

AgencyPortal 5.2

Contents

AgencyPortal 5.2.....	18
Getting Started	19
About AgencyPortal.....	20
Release Notes.....	21
Performance Report	22
Upgrade Guide	23
Upgrading from 5.1 to 5.2	23
Framework Overview	26
Boot Time Initialization	26
Front Servlet Architecture	26
Security	27
Dynamic Transaction Rendering/Behaviors	27
Change Management.....	27
Built-in Field Validation	28
Terminology	29
System Requirements.....	33
Tech Stack.....	34
Third Party Software and Licenses	36
Server	36
Client.....	40
Developer Setup.....	42
Prerequisites.....	43
Release Distribution.....	45
Setting Up Your AgencyPortal Project.....	46
Importing the Project into Eclipse	47
Configuring Integration Kits	49
Installing LOB Templates	50
Deploying Project Locally via Maven	51
Database Configuration.....	52
Initializing the Database	52
Local Solr Deployment	54

Required Properties.....	55
Setting Up a Tomcat Server.....	56
Running the App	58
Wrapping Things Up.....	59
Working with SDK Web Artifacts.....	59
Working with Web Sources	59
Importing the Project into SCM	60
Web-Based Installer.....	61
Prerequisites.....	61
Using the Installer.....	61
Deployment Information.....	64
WebSphere 8.5	65
AgencyPortal Web Application Archive Requirements	65
WebSphere Configuration Requirements	66
Additional Information	68
Configuring the Server.....	68
Setting Generic JVM Arguments	69
Configuring JNDI	71
Creating a JNDI JDBC Provider and Data Source	71
Deploying the WAR.....	73
Application Level Class Loader Settings.....	74
Module Level Class Loader Settings	75
Server-Level Classloading Settings.....	76
Additional Info for DB2 Users.....	78
DB2 Issue.....	78
Tomcat	80
Extra Dependencies.....	80
Additional Configuration.....	80
WebLogic.....	82
Solr Setup.....	84
Setting Up Solr in AgencyPortal	85
Using the Embedded Solr Engine	85
Using an External Solr Instance.....	85
Deploying Solr Under WebSphere	88
Tuning Solr Logging Under WebSphere	88
Deploying Solr Under WebLogic	89
Solr Cloud	90
SolrCloud and AgencyPortal	91
SolrCloud Cluster Composition	98

Backup and Recovery	99
Re-Indexing (Batch Index).....	99
User Interface	101
Browser Support	102
Browser Testing.....	102
Style Guide.....	104
Overall Color Palette	104
Navigation Bar	104
Home Page	104
LOB Icons	105
Data Entry	106
Add Buttons.....	107
Account Cards	108
Account Information	108
Button Guide	109
Button Types.....	109
Alignment.....	109
Drop-down Buttons	110
Button Grouping	110
Using Icon vs. Button.....	111
Button Sequencing.....	111
Buttons vs. Links	111
Buttons with Input Fields	112
Modal Guide	113
General Modal Standards	113
Action Modals.....	113
Other Modals	113
What not to do.....	114
Developer Guide	115
Data	116
Agencyport XML Engine (AXE).....	117
APDataCollection Properties	117
Data Schemas and Data Schema Resources.....	117
Characteristics of AXE's XPath Syntax.....	118
ID Attribute Generation	118
XML Schema	119
Data Schemas and Data Schema Resources.....	119
Creating a XSD Schema	120
Referencing an XML Schema from the TDF.....	120
Faster XML Parsing	121
Advanced Data Management	122
Views	122
Group ID Processing.....	125
Referential Integrity.....	125

ACORD Simplified Programming	128
Description of Existing Special Field Helper Functionality	129
Basic Goal and Premise of ACORD Simplified	130
Step-by-Step to ACORD Simplified.....	131
Defining the ACORD Simplified Data Model.....	131
Registering the ACORD Simplified Model with AXE	132
Adjusting the TDF to Use the ACORD Simplified Model.....	132
Creating Transformation Processing Units	135
Configuring the Transformers Resource	135
Registering the Transfer Resource in the Product Database.....	136
APDataCollection	136
Debug Console.....	137
Sample Simplifications	138
Index Management.....	141
Index Management Configuration	145
Change Management.....	147
ACORD ModInfo Review	148
Merged Document (DOM).....	149
ModInfo Action Use Cases	150
Diff/Merge Algorithm.....	151
Change Management API.....	152
Agencyport XML Engine (AXE) API	153
Rendering Agencyport Policy Change Summary Format.....	155
PersonalAutoPolicyChangeSummarizer	155
CMDDisplaypolicyChangeSummary	156
persautoEndorse/policyChangeSummary.jsp	156
Policy Change PDF Manager	156
AXE Best Practices	159
Delete/Add versus Change.....	159
Support for Complex Roster Processing.....	161
Managing Multiple Document View Types on APDataCollection.....	162
Index Management	162
Direct Use of JDOM is Discouraged.....	163
ACORD Policy Change Rendering Technique	164
com.agencyport.domXML.changeManagement.ModInfo Group	164
ACORD Form 71 (Personal Auto Policy Change PDF) Rendering Specifics	166

AXE Java Source Code Samples	169
PAEndorsementAxeTestCase.java	169
Determining whether anything has changed on this vehicle	173
Determining which vehicle description fields to print on this vehicle	173
Determining whether the garaging location relationship has changed or the data associated with the garaging location has changed.....	173
Determining whether driver percentages have changed	173
Retrieve all of the additional interests for this vehicle	173
Additional Interest mod info group process loop.....	173
Determination of whether this additional interest has changed.....	173
Change Management Configuration	175
Product Definition.....	176
Transaction Definition/Page Library	177
Transaction Definition	177
Page Library.....	178
Application Properties.....	178
Index of TDF Elements.....	178
Option Lists.....	195
Dynamic Transaction Rendering / Behaviors	197
Behaviors Repository	197
Transaction Definition Provider	198
PreConditions	198
IntraPage Support Module.....	198
Behavior XML Syntax	199
<hotField>	199
<behavior>	200
Behaviors: Built-in Preconditions and Operators	207
Behaviors and Hotfields	209
Enabling IntraPage DTR.....	210
Operational Assumptions	210
Enhanced DTR Processing.....	214
PreCondition Configuration and Synchronization	214
Overriding Default Data Cleanup Behavior.....	224
Overriding Default Data Cleanup Behavior.....	226
Field Validation.....	228
Available Validation Methods	229

Built-in Field Validation Connector	233
Process Side Field Validations	233
Display Side Field Validations / Connectors	233
Custom Field Validation	235
Custom Field Validation Example #1	235
Custom Field Validation Example #2	236
Transaction Validation.....	237
Custom Transaction Validation	237
Transaction Validation Report (TVR)	239
Sample Transaction	239
Additional Notes	240
TVR Tracking and Impactful Fields	240
Connectors	242
How Connectors are Executed.....	242
Connectors In Detail.....	244
XARC Rules	246
Technical Notes for XARC Rules.....	247
Workflow Management	248
Connector Outcome	248
Workflow Processing.....	248
Workflow Management Database Updates	249
Views	250
Types of Views.....	250
View Classes.....	252
Views in Java.....	252
View File.....	256
Example View File.....	260
Product Database.....	262
Creating a Product Database	264
Product Definition Memory Conservation.....	265
Field Level Messaging	266
Simple vs Detailed Messaging	266
Inserting the Value into a Message.....	266
Changing the Icon.....	266
Tooltips	266
Work Items.....	268
Work Item Management	269

ACORD to Work Item Mapping.....	270
Work Item Management Example	271
Extending the DDL.....	271
Extending the WorkItem Class.....	271
Extending the WorkItemManager Class.....	271
Work Item Status Management.....	273
Work Item Record Locking	275
Autosave	276
Enforcing Account Creation	277
Percent Complete	278
Customizing the Calculation	278
Overriding the Default Calculation.....	278
Disabling the Progress Bar.....	278
Removing the Calculated Percent	278
Improved Access to Work Item/Account Data	279
Account Management	280
Configuring Account Management	280
Account Management Infrastructure	280
Personal and Commercial Account Templates	281
Enforcing Account Creation	283
Solr Configuration.....	284
Solr Architecture.....	284
Solr Configuration Files	286
SolrManager.....	289
Using SolrManager.....	289
Indexing	289
Deleting	289
Searching	289
SearchManager.....	291
Solr Batch Indexing	292
Solr Batch Index Processing.....	292
Working with Solr Batch Indexers.....	293
Accessing Batch Indexers	295
framework.properties	295
General Setup for Access	295
Index Mapping	296
Index Mapping Schema	296
Resource Type.....	296
Compiled Runtime Representations	297

Index Data Mapper (Synchronization of Search Indices).....	297
Communication and Invocation	297
Solr Security.....	299
Token Validity.....	299
Token Generation	299
SolrSecurityServlet	300
Adding a New Field to an Existing Solr Core	301
Adding a New Solr Core (Index)	302
Work Item Assistant.....	304
Work Item Assistant Configuration	304
Email Notifications	307
Timeline	309
Timeline Configuration.....	310
Timeline Technical Overview.....	311
Data Model.....	311
Backward Compatibility	312
Event Collection.....	313
Work Item Timeline View	318
Access to Work Item Timeline	322
My Defaults.....	323
Save to Defaults	323
Apply Defaults	323
Available Fields	323
Work List	325
Work in Progress Feature	326
Work in Progress Columns	326
Work List Architecture	327
Server Side Architecture	327
Work List View Provider Interface	328
Work List View Provider	328
Work List Search Provider Interface	328
Solr Work List Search Provider	329
Compiled Work List View	329
comagencyport.api.worklist.pojo.WorkListView	329
Get Account Work List View.....	332
Get LOB Work Item Work List View	332
Save Account Work List View.....	333
Save Work Item Work List View	333

Delete Account Work List View	333
Delete Work Item Work List View	334
Query Account Index.....	334
Query Work Item Index.....	334
Client-Side Architecture	335
Server Side Work List Configuration and Customization	337
Work List Definition	337
Supporting Data Structures	337
Customizing Work List.....	343
Client-Side My Work and My Account Configuration	347
Adding New Fields to the UI.....	348
Adding a New Work List.....	348
Adding a Custom LOB Link.....	348
Work Item Quotes	350
Work Item Queues	351
Work List Customizations	354
Custom Work Item Actions	354
Security	355
Effective Security Programming.....	356
Security Programming Terminology	357
Step-by-Step to ACSI.....	358
Sample Security Package.....	358
Setting Up an acsi.properties File	358
Fleshing Out the ACSI Security Model	359
Making the Case for Fine-Grained Permissions.....	361
How to Introduce Fine-Grained Permissions.....	361
Implementing the Security Provider	364
Numeric Identity Provisioning	365
Extending the Security Profile Manager.....	366
Implementing the Security Provider	368
Extending the Security Profile	371
Upgrading from 3.x	372
Checklist	374
ACSI	375
Application Layer	377
Security Adaptor Layer	378

Security Provider.....	381
Security Reference Implementation.....	382
Additional Security Considerations (OWASP Top 10)	383
SQL Injection	383
Broken Authentication and Session Management.....	383
Cross-Site Scripting (XSS, aka Safe Strings)	383
Insecure Direct Object References.....	384
Security Misconfiguration	384
Sensitive Data Exposure	384
Missing Function Level Access Control.....	385
Cross-Site Request Forgery (CSRF).....	385
Using Components with Known Vulnerabilities	388
Unvalidated Redirects and Forwards	388
Content Security Policy	389
Encryption.....	390
Keystore Files	390
Field Security	392
Architecture	392
Server Side Configuration	394
Session Timeout.....	396
Request Processing	397
Transaction Page Processing.....	397
Displaying and Processing HTML Form Layouts	397
Supporting a Standard Sequence of Controlling Tasks	397
Front Controller Pattern	398
Determining the Class Names	399
Customizing the Standard Sequence of Processing Tasks.....	401
Database	403
DatabaseAgent.....	403
DatabaseResourceAgent.....	404
Property Reference	405
Data Model.....	407
Client-Side Development.....	408
Working with SDK Web Artifacts.....	408
Working with Web Sources	408
Altering the HTML Markup	409
Styling the User Interface	410
Client-Side Programming	413
Base Classes.....	414
Base Functions	414
Hookpoints	419
Third Party Libraries	421
PDF Generation.....	423
Legacy PDF Generation Utility	424
Transformation Matrix.....	424

PDF Generation Utility	426
Form Data Mapping	426
PDF Form Processing	426
Form Population.....	428
Form Hyperlinks	428
Resources and Artifacts	428
PDF Generation Utility Configuration.....	431
Establish All Forms Needed for Each LOB	431
Retrieve and Review eForms	431
Map Form Data	432
XPath	437
TDF Source	437
Custom Java Class.....	437
Creating a Custom IFieldValueGetter Interface	438
When to Create Resource Files	439
Simple Repeat	439
Double Repeats	440
Field Repeat on Repeat	440
Add JSP Tag to Policy Summary Pages	447
Integration	449
Connect5	450
Features and Architectures.....	450
Connect5 Best Practices	451
Connect5 Server Configuration	452
All Servers.....	452
WebSphere.....	452
Tomcat	452
Weblogic	452
JBOSS.....	452
Connect5 Eclipse Setup	453
Web Project (Required)	453
Integration Kits (Optional).....	453
Module Project (Optional).....	453
Outcome	453
Connect5 Report Configuration	455
XML Report Parameter Model.....	455
Report Properties	455
Web Reports	455
Runtime Logging	456
Connect5 Database Definitions	458
General Technology	458

SDK Tables	459
Connect5 Deployment Configuration	464
Build Artifacts	464
Deployment Diagram.....	464
Properties.....	464
Database.....	466
Turnstile Integration.....	468
History of Turnstile	468
Integration with Connect5.....	468
Turnstile Configuration.....	468
Using Turnstile with AgencyPortal	468
Turnstile Technical Workflow	470
Turnstile PDF Processing.....	471
Form Sever Response Processing	471
Email.....	475
Integration Kits	477
Integration Kit Architecture	478
Application Tier	478
Service Broker Tier	479
Application Properties.....	480
Property Concatenation.....	480
Property Substitution.....	480
Customizing Framework Base Classes	480
Property File Reference	483
Framework Properties	483
Security Properties	494
Application Logging Properties	496
Cache Properties	497
Localization Properties	498
Version Properties	498
Email Notification Properties	500
LOB Specific Properties	501
Upload Writer Programming	503
ImportBaseProcessor and UploadDataManager Classes	504
ImportBaseProcessor.....	504
UploadDataManager	505
Upload Message Mapper	507
Agency Connect Status Message	511
Sample Messaging for Validation Events	512
Sample Messaging for Correction Events	513
Technical Solution	514

Class Model.....	514
Sequence Diagrams	517
Upload High-Level	517
Transform Details	518
Custom Alter Import Data	519
Update Input Data Collection	520
Configuration Options.....	521
Best Practices for an Effective Upload Writer Development Exercise	524
Real Time Rating (RTR).....	528
Architecture	528
Supporting Data Structures	529
Resources and Artifacts	529
RTR Configuration	530
Logging Configuration.....	534
Logging Destination & Configuration Properties	536
Application Logging Configuration	536
Logging Packages and Classes	540
Configuration Files	541
Application Properties File	541
Application Logging Configuration File	541
Debug Console.....	544
Custom Debug Panels	545
Application Bootstrap	547
Bootservice Providers	548
Environment Specific Resources	549
Event Audit	551
Data Model.....	552
Event Display	554
Event Audit API.....	557
Localization.....	559
Product Definition.....	560
Property Structure	561
Component Descriptions	561
Localization of AgencyPortal Product Definitions	567
Localizing Your Content.....	570
Performance Logging.....	575

PerfObject	575
SDK Performance Package	576
Performance "Math"	577
JMeter Instruction Guide	579
Installing JMeter	580
Recording a Script.....	581
Reports	593
Report Basics	593
Report Configuration	598
Configuring a Report.....	598
User Workflow.....	602
Getting Started	603
Home Page	603
Session Timeout.....	605
Work List	607
Page View	607
Work Item Actions	609
Search and Filter	611
Searching for Accounts and Work Items	611
Sorting and Filtering Accounts and Work Items	615
Searching for Accounts within Other Components of the Application	619
Filterlist Search and Filter	621
Work Items.....	623
About Work Items.....	624
Transaction Types	625
Page Types	626
Messages	627
Work Item Links	627
Summary Pages	628
Page Elements	630
Fields.....	631
Field Validation.....	632
Field Helpers.....	633
Navigation Menu.....	635
Autosave	637
Work Item Assistant.....	638
Comments	638
File Attachments	639

Other Active Users	642
Secure Fields.....	643
Uploaded and Endorsed Work Items	643
Read Only Fields.....	644
Timeline	645
Timeline Events	645
Event Details.....	647
My Defaults.....	648
Saving Defaults	648
Applying and Managing Defaults	649
Help with Defaults.....	650
Creating a Work Item.....	651
Percent Complete	653
Data Pre-fill.....	654
Using Work Item Assistant.....	655
Navigating through the Transaction Pages	655
Managing Work Items	656
Modifying a Work Item.....	656
Linking a Work Item to an Account.....	658
Moving a Work Item.....	660
Copying a Work Item.....	661
Deleting a Work Item.....	661
Accounts	662
Page View	662
Account Types.....	663
Account Detail View	663
Creating an Account.....	665
Adding a Work Item within an Account.....	666
Down-Filling into a Work Item.....	667
Adding Account Level Locations and Contacts	668
Managing Accounts	671
Modifying Account Details	671
Merging an Account.....	672
Deleting an Account.....	674
Reports	675
Upload	677
Turnstile Integration	677
Field Level Upload Messaging	686
Mismatch	686
Automatic Corrections	686
Validations	688
Manual Update	688
PDF Spotlight.....	690

External References	691
JavaDoc	692
JSDoc	693

AgencyPortal 5.2

Welcome to Agencyport's online documentation for AgencyPortal 5.2. The topics in this documentation are organized by the following sections:

- **Getting Started**

This section includes information a software developer needs to get started with AgencyPortal, including requirements and set up procedures. The targeted audience of this section is a software developer who is assigned the task of implementing AgencyPortal.

- **User Interface**

This section details the out of the box styles used for AgencyPortal, as well as information on the buttons and modals that are present throughout the application. The targeted audience of this section is a front-end developer assigned the task of customizing AgencyPortal.

- **Developer Guide**

This section includes detailed reference information and examples to help a developer configure and maintain the components of AgencyPortal. The targeted audience of this section is a software developer assigned with the task of customizing AgencyPortal.

- **User Workflow** This section details how to use AgencyPortal and its functionality. The targeted audience of this section is a business analyst or other non-technical analyst who wants to know more about how the features within AgencyPortal work.

- **External References**

This section includes additional information and hyperlinks to external resources that can help a developer with their AgencyPortal project. The targeted audience of this section is a software developer assigned the task of implementing AgencyPortal.

These sections target developers and non-technical analysts' documentation needs, and provides detailed information on how to install, set up, configure, maintain and use the [AgencyPortal](#) application and its features. For release information related to the AgencyPortal 5.2 release, refer to the [Release Notes](#) topic in the Getting Started section.

Getting Started

AgencyPortal consists of several software and hardware components provided by different sources. This section provides details on how to set up the AgencyPortal web application using the AgencyPortal software development kit (SDK), and is geared towards experienced software developers who are assigned the task of implementing an AgencyPortal project.

You will learn:

- [about AgencyPortal](#)
- what's included in each [release](#)
- the results of [performance testing](#)
- what's needed to [upgrade](#) AgencyPortal 5 to 5.1
- details about the [AgencyPortal framework](#)
- the [terminology](#) used in this documentation and throughout the application and SDK
- the [technology](#) used in AgencyPortal
- about [3rd party software and licenses](#) for server and client portions of the application
- how to [install, set up and customize](#) AgencyPortal
- how to [set up your environment](#) to deploy AgencyPortal
- how to [set up Solr](#) and all the components associated with the AgencyPortal [search and filter](#) feature

The above items only highlight what is included in this section. Before you begin the setup process, make sure you thoroughly read through the topics included in the [Developer Setup](#) section.

The following sections also include additional information to use as references for your project:

- For more information on configuring the various components of your project, refer to the [Developer Guide](#) section.
- For non-technical descriptions of how various features work by default, refer to the [User Workflow](#) section.

About AgencyPortal

AgencyPortal is a web-based platform that allows P&C insurance carriers to build web applications that execute key business transactions quickly and easily with their agents, brokers, MGAs or policy holders. AgencyPortal includes all components necessary to extend the workflow of a carrier's policy processing environment out to multiple distribution channels.

The back-end of AgencyPortal is written in Java and the front-end is standard web technology, including HTML, CSS and JavaScript.

AgencyPortal is not a one-size-fits-all application. It is shipped as a framework that includes all the key components you'll need to get started, but will allow you to support the unique requirements of your business and your insurance products. AgencyPortal can be tailored by an insurance carrier to build:

- agent and consumer point of sale portals for commercial, personal and specialty lines
- quoting for new business and renewals
- agent and consumer self-service for billing, endorsements or claims

Features of AgencyPortal include:

- ready-made templates based on ACORD standards to use as a starting point for your project
- the Toolkit - a web-based interface that helps you build and maintain your portal pages and business rules yourself
- the ability to export business rules from AgencyPortal in an easily-readable format to ensure compliance with underwriting rules
- an architecture designed for integration, including [Connect5](#) for integration with core systems, enterprise web applications, document management, email and more
- support for online collaboration with all parties involved in the transaction
- multilingual abilities

Release Notes

The following includes hyperlinks to view release note documentation related to the AgencyPortal 5.2 release:

- [AgencyPortal 5.2 Release Guide](#)
- [AgencyPortal 5.2.0-1 Release Notes](#)
- [AgencyPortal 5.2.0-2 Release Notes](#)
- [AgencyPortal 5.2.0-5 Release Notes](#)
- [AgencyPortal 5.2.0-6 Release Notes](#)

Refer to the [Upgrade Guide](#) topic for instructions on upgrading from AgencyPortal 5.1 to 5.2.

Performance Report

With each release of AgencyPortal, the Product Development team runs several performance tests on the application. The following report provides details on the AgencyPortal5.2 results:

- [AgencyPortal 5.2 Performance Test Summary](#)

Upgrade Guide

Upgrading from 5.1 to 5.2

This section serves as a guide for application developers upgrading their applications from AgencyPortal framework version 5.1 to version 5.2. If the application being upgraded is currently on a pre-5.0 version, developers must familiarize themselves with all of the prior upgrade guides before attempting to upgrade directly from 5.2.

While it is always our intention to minimize the impact of product upgrades, there are inevitably areas where changes are necessary to existing implementation code and configuration to provide the required enhanced functionality. This guide is intended to provide a comprehensive list of those changes and serves as a checklist for the person(s) performing the AgencyPortal framework upgrade.

Note

We advise that application developers familiarize themselves with the entirety of this guide before attempting to perform an upgrade.

Third Party Library Changes

Several of the third party libraries have been updated to a newer version. Please refer to the list below to replace the JARs in your project with the new ones:

Library Name	Old Version	New Version
jackson-annotations.jar	2.5.2	2.6.3
jackson-core.jar	2.5.2	2.6.3
jackson-databind.jar	2.5.2	2.6.3
jackson-jaxrs-json-provider.jar	2.5.2	2.5.2
jackson-jaxrs-base.jar	2.5.2	2.5.2
solr-core.jar	4.4	4.10.4
solr-solrj.jar	4.4	4.10.4
jersey-container-servlet-core.jar	2.17	2.22.1
jersey-server.jar	2.17	2.22.1
jersey-media-json-jackson.jar	2.17	2.22.1
jersey-media-json-processing.jar	2.17	2.22.1

Solr Version Changes

We have upgraded Solr to version 4.10.4. If you have AgencyPortal instances that work with standalone Solr installations, these Solr installations should be upgraded to version 4.10.4.

Similarly, any Apache ZooKeeper installations in place for Solr Cloud should be upgraded to version 3.4.6, since this is now the version of ZooKeeper that Solr 4.10.4 mandates.

Database Changes

SQL Server 2008

If you're using SQL Server 2008, you must switch the `database_agent_class_name` application property to point to the new `LegacySQLDatabaseAgent` since the standard `SQLDatabaseAgent` now relies on database sequences that are not supported in SQL Server 2008: `database_agent_class_name=com.agencyport.database.LegacySQLDatabaseAgent`

Then run the following update script:

[upgrade-5.2-sqlserver-2008.sql](#)

SQL Server 2012 +

The standard `SQLDatabaseAgent` for SQL Server 2012 now relies on sequences instead of the `NextKeyValue` table and `GetNextKeyValue` stored procedure. The following updated script contains sequence creation statements for the standard tables; however, you must modified these statements so that the sequences start with a key integer high enough to account for the records in your specific database. Additionally, sequence statements should be added for any other non-standard tables or custom code that relies on keys that are managed by the `NextKeyValue` table (i.e., there should be a sequence for each and every record in the `NextKeyValue` table).

[upgrade-5.2-sqlserver-2012.sql](#)

MySQL

Run the following update script:

[upgrade-5.2-mysql.sql](#)

Oracle

The standard `OracleDatabaseAgent` now relies on sequences instead of the `NextKeyValue` table and `GetNextKeyValue` stored procedure. The following update script contains sequence creation statements for the standard tables; however, you must modify these statements so that the sequences start with a key integer high enough to account for the records in your specific database. Additionally, sequence statements should be added for any other non-standard tables or custom code that relies on keys that are managed by the `NextKeyValue` table (i.e., there should be a sequence for each and every record in the `NextKeyValue` table).

[upgrade-5.2-oracle.sql](#)

DB2-9

Run the following update script:

[upgrade-5.2-db2-9.sql](#)

Web Content File Changes

Modified files:

- /account/accountDetail.jsp
- /dynamicpage/dynamicpage.jsp
- /policychange/changeRequests.jsp
- /shared/accountEntities.jsp
- /shared/lightboxes/default/tcr/display.jsp
- /shared/work_item_menu.jsp
- /site/timer.jsp
- /workitemassistant/modals.jsp
- /worklist/partials/modals/pleaseWait.mdl.jsp
- /worklist/partials/modals/workitem-approveWorkitem.mdl.jsp

Added Files:

- /account/lossSummaryHelper.jsp

Web Source File Changes

Modified Files:

- /js/ap_javascript_objects.js
- /js/queues/worklist.recentWorkItemsQueue.ctrl.js
- /less/agencyportal.less
- /less/cards.less
- /less/forms.less
- /less/modals.less
- /less/variables.less
- /less/agencyportal.less
- /less/agencyportal.less

Added Files:

- /js/ap_fieldsecurity_handler.js
- /js/snippets/ap_snippets.js
- /lib/imageMapster-master/

Web.xml File Changes

The following is added to the web.xml file:

```
<servlet-mapping> <servlet-name>SnippetImageRenderer</servlet-name> <url-pattern>/SnippetImageRenderer</url-pattern> </servlet-mapping> <servlet> <description>This servlet handles the request for Turnstile Snippets</description> <display-name>SnippetImageRenderer</display-name> <servlet-name>SnippetImageRenderer</servlet-name> <servlet-class>com.agencyportturnstile.SnippetImageRenderer</servlet-class> </servlet>
```

Property File Changes

Added Properties

enable_turnstile_snippets property is added to the framework.properties file. This property defaults to true.

Upload Interface Changes

The following URL was changed:

Previous URL: <http://host:port/agencyportal/UploadWriter>

New URL: <http://host:port/agencyportal/api/uploadwriter>

Framework Overview

The AgencyPortal framework is a Java based solution using XML, servlet and JSP technologies running in a JEE application server. No proprietary application server APIs or components are used. An agency portal application built on the AgencyPortal framework will run in a variety of web application containers and servers, including Apache Tomcat, IBM WebSphere, Oracle WebLogic and JBoss.

Boot Time Initialization

When the JEE application server hosting AgencyPortal starts up, several standard framework services/subsystems, as well as any custom services/subsystems, must be initialized. JEE ServletContextListener and associated context parameter entries handle this bootstrap processing. The [Application Bootstrap](#) topics describes the ApplicationBootstrap routine in detail.

Front Servlet Architecture

Typically, after a user is logged on, they are directed to a page generated by the JSP home.jsp. home.jsp is included with AgencyPortal for reference, but this JSP is expected to be customized.

On home.jsp, you can find a URL that looks similar to the following:

```
"FrontServlet?PAGE_NAME=generalInfo&TRANSACTION_NAME=quickQuoteBOP&WORKITEMID=1528&METHOD=Display&TRANSACTIONSTARTPOINT=true&rnd=0.1905311247662278&CSRF_TOKEN=8173c9f3aab4f0cf47a1339156f482d7"
```

FrontServlet is the "front end" of the framework and normally receives all browser input. The FrontServlet takes the rest of the parameters from the URL and constructs the proper command, or target servlet, to run. With rare exceptions, the target servlet will either be a servlet found in the com.agencyport.servlets.base package or in the "user supplied" packaged for this transaction.

In this example, the generated target servlet will be the following:

```
com.agencyport.servlets.bop.CMDDisplaygeneralInfo
```

This is the name of a Java class. FrontServlet then does a lookup to see if this command should actually be handled by a base servlet (a built-in servlet) or if the name represents a user supplied servlet (the base class needed to be extended for some reason).

The parameter FIRST_TIME=true is a special framework parameter required when invoking the first page of an application (transaction) for the first time. After the framework locates the correct servlet, it executes and the first page of the application displays. At a high level, the framework servlet for the first page of an application does the following:

- creates a new work_item ID
- creates the menu control file for the work item
- obtains and forwards the page object to dynamicpage.jsp, which generates the HTML that shows up on the user's browser

Assume that the user fills in some data on this first page and presses submit. The HTML for that first page contains a hidden field - PAGE_NAME - that contains the name found in the TDF for the page.

Example

```
<transaction title="Workers Compensation" target="WorkCompPolicyQuoteInqRq"> <page id="generalInfo" title="General Information" type="dataEntry">
```

When the user submits the HTML from the FrontServlet, the framework will use the name generalInfo to manufacture the target servlet name.

The target servlet will process the data. A servlet is automatically "wired" by the framework to a class named DataManager. This class directs the movement of data into and out of the physical data store. By default, the store is an APDataCollection whose contents are written as a BLOB to a database server. APDataCollection is a wrapper class around an XML structure. A very high level overview of the data management component is as follows:

- The framework's properties file contains an entry(ies) that names the data schema(s) to use for storing the data. The schemas are loaded at startup time.
- The transaction definition file names one of the schemas listed in the properties file as the target for the data.

Example

```
<transaction title="Watercraft" target="WatercraftPolicyRequest"/>
```

The syntax employed in a page definition provides the mapping between the HTML field and the target field in the data store. If, on the HTML page, there is a text box labeled "Carrier Name" and the contents of the text box are to be stored in the XML element given by the WatercraftPolicyRequest > PriorPolicy > Carrier path, the definition for the HTML field would look similar to the following:

Example

```
<fieldElement type="text" id="PriorPolicy.Carrier" label="Carrier Name" ... />
```

Note

Since `WatercraftPolicyRequest` is the parent of all elements in the document, it only needs to be specified once and not per `fieldElement`.

Finally, after the data has been stored away, the servlet updates the menu control file to indicate that the page has been "visited" and, as a result, can be "jumped to" via the Navigation Menu. It then determines the next page to display and forwards that information to the `FrontServlet`.

If the page must perform additional tasks besides data entry, there are two mechanisms available:

- service "plug-ins" (commonly referred to as connectors)
- overriding default base class behavior with a custom class (commonly referred to as APCommand extensions)

Service "plug-ins" are typically used to accomplish tasks, such as the following:

- business rules
- rating (quoting)

These services can be configured to run either on the display or process sides and are specified in the TDF as part of the page definition.

Developers create connectors to services, following a lightweight specification that allows the framework to automatically call any connectors configured for a page.

Security

Each inbound HTTP request is expected to go through a security filter before execution reaches the `FrontServlet` class. The security filter provides the following functions:

1. Authenticates that the request is coming from a trusted source.
 - Interfaces with the carrier back end during initial SSO handshake operations to authenticate the request.
 - Orchestrates the creation of a security profile from which client indicative information, functional permissions and work group associations are stored. This is done initially once per browser session.
2. Verifies that the current user has the authority to execute the target action of the request.
3. Verifies that the current user has the authority to access the work item on the request.

The security filter is part of a pluggable architecture called ACSI, which provides a common secure mechanism for authenticating and authorizing each HTTP request. Refer to the [Security](#) topic in the Developer Guide section for more information on security.

Dynamic Transaction Rendering/Behaviors

Dynamic transaction rendering (DTR) provides the necessary infrastructure in the framework that serves as the basic foundation for applications to support the concept of reflexive questioning. This provides the ability to conditionally display or alter one or more page entities in a transaction given the state of one or more precondition values. Refer to the [Dynamic Transaction Rendering/Behaviors](#) topic in the Developer Guide section for more information.

Change Management

AgencyPortal framework provides support for change management transactions by way of the Change Management Engine. Refer to the [Change Management](#) topic in the Developer Guide section for more information on this component.

The change management infrastructure included with AgencyPortal provides additional user experience elements to simplify the process of making changes to existing policy data and implements an industry standard mechanism for representing change requests in the form of the ACORD ModInfo aggregate.

Built-in Field Validation

The AgencyPortal framework includes a field validation sub-system driven by TDF configuration; thereby, eliminating the need to program many field validations in a rules engine. Refer to the [Field Validation](#) topic in the Developer Guide section for more information on this sub-system.

Terminology

The following key terms are central to development at Agencyport and are used frequently throughout documentation. Review these terms to get familiar with our terminology before getting started with the installation process.

Term	Definition
ACSI	Acronym for Agencyport Common Security Interface.
Agencyport XML Engine (AXE)	The Agencyport XML Engine (AXE) is used to describe the components of the framework developed to manage the complexities of working with ACORD XML as a data format. AXE has its own XPath type syntax, which closely resembles that of full W3C XPath.
Aggregates	The term aggregate is used to describe a set of XML nodes that exist as a group and is reused in multiple places within the schema. ACORD uses the term extensively.
APDataCollection	APDataCollection is a term used to describe the ACORD XML business data instance. The term APDataCollection is used in two contexts. APDataCollection refers to a JDOM based XML object that is wrapped by the class APDataCollection; in this context it "is" the application data.
Base Classes	The SDK associates pre-built base classes with each of the page types. In normal situations, the base classes can fully process the page type. The base classes are designed for extension to handle special situations. These classes are also often referred to as commands.
Behavior	A behavior describes to the DTR subsystem the nature of a specific dynamic alteration.
CLOB/BLOB	The default physical storage mechanism employed by the SDK is to store the data as a CLOB (Character Large Object) or BLOB (Binary Large Object) in a database. In databases where support is provided, XML column types are used in place of the CLOB or BLOB types.
Change Management Engine	The Agencyport framework provides support for change management transactions (endorsements, renewals, etc) by way of the change management engine. The change management infrastructure included with AgencyPortal provides additional user experience elements to simplify the process of making changes to existing policy data and implements an industry standard mechanism for representing change requests in the form of the ACORD ModInfo aggregate.
Codelists	Codelist files contain one or more sets of name/value pairs used by the application. Codelist files can be used to drive the values of fieldElements selectlists, filterlists and radio buttons.
Commands	The SDK associates pre-built commands with each of the page types. In normal situations, the base classes can fully process the page type.
Compound Key	The mechanism used to identify a page entity to DTR. The format of a compound key is XPath like. It follows the following format: Transaction id/Page id/ Page Element Id/Field Element id.
Connectors	Connectors are the mechanism by which the framework integrates with third party services or the way to apply business specific logic. They can be engaged on either the display or process sides.
Custom Page	A type of page used to implement a custom JSP. Typically, custom JSP pages are used only to implement quote results and submission summary pages, and are not used for data collection.
Data Entry	A type of page used to collect basic non-repeating data. For example, 'general information' might be collected on a data entry page.
Data Schema	The data schema file describes the data store. The data schema is an XSD file. The data elements transferred to/from the physical data store and the HTML pages are in a sense managed by this schema. The default schemas are defined by ACORD, one of the primary developers of insurance standards. In general, each line of business has a specific schema (entities common across lines of business are of course shared). The

Term	Definition
	HTML element names found in the TDF "map" directly to the element names found in the data schema and are, as a result, ACORD standard data field names. The effort saving and speed to market advantages of utilizing an ACORD foundation are considerable: the logical data model is already defined, ACORD provides an extensive off-the-shelf data dictionary, defining all the data fields, and re-using is heavily promoted by standardization; there is little rational for analysts or developers to "reinvent the wheel" as each new line of business comes along.
Debug Console	The AgencyPortal debug console is designed as a development aide and provides useful information about the application. The debug console is made up of panels, each of which provides a piece of functionality.
Display	The term display-side and process-side (or simply display and process) refer, respectively, to the logic necessary to display a page to the user and to process any data entered on a page after the page has been submitted.
Display-Side	The terms display-side and process-side (or simple display and process) refer, respectively, to the logic necessary to display a page to the user and to process any data entered on a page after the page has been submitted.
Dynamic Transaction Rendering (DTR)	The term Dynamic Transaction Rendering (DTR) is used to describe the components in the application that make a transaction dynamic by dropping, adding or altering page entities. DTR is implemented through behaviors.
FieldElement	A fieldElement is a TDF representation of a single application field.
Framework	The core software components that comprise AgencyPortal. Often used interchangeably with SDK.
Front Servlet	The Front Controller Pattern centralizes request logic by presenting a single point of entry for the web application. A front controller typically consists of a request handler and a catalog of dynamically invoked commands that process the web request.
Hot field	A predetermined, targeted field on a page whose state change triggers an event that could result in a change to the current page's composition.
HTMLDataContainer	The term HTMLDataContainer is used to describe the collection of data coming off a single page. This container is instantiated on the process-side only.
Interest Level	A message sent to the page subsystem from the transaction definition provider that instructs the page subsystem whether to include, exclude or alter a TDF page entity.
IntraPage DTR (IPDTR)	DTR as it applies to a page governed by page entities on that same page. Changing the composition of a web page in reaction to a change that the user has applies to one of the fields on that page itself. Intra-page DTR can apply to both data entry and roster based SDK pages.
Origin hot field	The hot field that triggers an event to re-render the composition of a page. Although there can be multiple hot fields associated with a page, there is one and only one hot field that triggers the specific event at a single time.
Page	The content displayed in the user's browser window is referred to in this document simply as a page. There are a small number of supported page types in the framework, including data entry (used for basic non-repeating data entry pages), roster (used in the case where a repeating group is being collected) or page display (in the case where a custom JSP page is required). Typically, custom JSP pages are used only to implement quote results and submission summary pages.
Page Entity	A TDF is comprised of page entities. Examples of page entities include the page, pageElement and fieldElement physical entities. A page entity is a general term that can be used to refer to one of the aforementioned physical entities.
Page Libraries	Common pages can be defined in files called page libraries. Pages defined in page libraries can be reused across transactions or even within a single transaction for applications that support interstate.

Term	Definition
PageElement	A pageElement is a page entity in the TDF and is a way to group items within a page. pageElements contain individual fieldElements and are used to create fieldsets, rosters, questionnaires, etc within the application.
Precondition	A condition that is discovered before the first page of a transaction is displayed.
Process	The terms display-side and process-side (or simply display and process) refer, respectively, to the logic necessary to display a page to the user and to process any data entered on a page after the page has been submitted.
Process-Side	The terms display-side and process-side (or simply display and process) refer, respectively, to the logic necessary to display a page to the user and to process any data entered on a page after the page has been submitted.
Roster	A special type of page used to collect repeating data. For example, a list of workers compensation class codes might be collected in a roster.
SDK	The core software components that comprise AgencyPortal and the software development kit that is used to implement them. Often used interchangeably with "the framework".
Submission Navigation Menu	The term submission navigation menu is used to describe the left hand menu presented to the user within a transaction.
Transaction	A series of pages is termed a transaction. The notion of a transaction is central to AgencyPortal and it defines a unit of work for the end user. For example, a personal auto new business application is implemented as a transaction.
Transaction Definition File (TDF)	The term transaction definition file (TDF) refers to an XML file that describes the content of each of the pages in the traversal (the "screen flow"). The TDF, while page specific in nature, can be thought of as the driving force behind the web application and, in a sense, the framework as well. The term transaction, therefore, refers (depending on the context) either to the series of pages making up the web application or a web application known to and "functioning" within the framework.
Transaction Definition Provider	A DTR support module responsible for returning interest level messages to the page subsystem for specific page entities and carrying out alteration/modification upon those page entities it desires to alter.
Transaction Definition Behavior Repository	A collection of behavior metadata files that relates how interest levels and alterations apply to a set of page entities under various preconditions/conditions.
Transaction Validation Report (TVR)	A compilation of all of the current outstanding field validation issues associated with a work item.
Views	Views are used to support complex data relationships when there is not a one-to-one relationship between a field on the page and a target in the data store.
Work in Progress (WIP)	The work in progress queue is a page that displays a list of work items that are currently active for a given user and often gives them the ability to open, delete or, otherwise, manage any given item.
Work Item	A single, unique submission and the primary unit of work in AgencyPortal. A work item can typically be thought of as an individual insurance application. Many work items may exist for a given customer if they have multiple active quotes or have applications in progress for a variety of lines of business. Examples of a work item include an individual quick quote, new business or endorsement submission.
XML Column	In databases where support is provided, XML column types are used to store the XML application data. This option provides a way to query XML documents for discreet information using XQuery.
XPath	XPath is a language for finding information in an XML document. It uses path expressions to select nodes or node-sets in an XML document.

Term	Definition
XQuery	In RDBMS where support is provided for it, XQuery is used to query and report against data in the data store.

System Requirements

The AgencyPortal framework is a Java based solution using XML, servlet and JSP technologies running in a JEE application server. No proprietary application server APIs or components are used. An agency portal application built on the AgencyPortal framework will run in a variety of web application containers and servers, including Apache Tomcat, IBM WebSphere, and Oracle WebLogic.

Hardware and software requirements for the AgencyPortal framework in a production environment are as follows:

Suggested typical heap sizes are 128min/1024max in Microsoft Windows based environments and 128min/2048 max in environments (i.e., Unix/Linux), which can allocate larger amounts of memory to a single process. Heap tuning greatly depends on the application profile and number of concurrent users.

AgencyPortal Server	
Hardware	<ul style="list-style-type: none">• 2GB or Greater of System Memory• Dual to quad CPU configuration• Multiple Disks to Support a RAID Configuration
Operating System	As a Java web application, AgencyPortal is operating system agnostic. Any OS (Windows, Linux, Unix, ZOS, iSeries, etc) that can run a Java application server (i.e., WebSphere, WebLogic, JBoss, etc) is capable of running AgencyPortal.
Application Server	<ul style="list-style-type: none">• IBM WebSphere 8.5 or Greater (with the IBM JDK 1.7)• WebLogic 12c or greater• Apache Tomcat 7 or greater• JBoss 6.1 or greater
Database Server	<ul style="list-style-type: none">• Microsoft SQL Server 2008/2012/2014• DB2 8, DB2 9, DB2 9, DB2 10 with PureXML• Oracle 9i, Oracle 10g, Oracle 11i, Oracle 11g, Oracle 12c• MySQL 5.x
Java Runtime Environment	The AgencyPortal SDK is compiled on version 1.7.x of the JDK

AgencyPortal Client	
Browsers	<ul style="list-style-type: none">• IE9+• Chrome (current -1)• Firefox (current -1)• Safari 5.1+• Microsoft Edge

Tech Stack

The following includes the tech stack for AgencyPortal 5.2.

Application

JEE 6	Enterprise Java computing platform
JDK 1.7	Java Development Kit

3rd Party Libraries

Apache Axis	Exchanging file attachments between AgencyPortal and AgencyConnect4
Apache Commons	Turnstile file upload
Apache Solr™	Application wide search
Apache PDFBox	PDF generation tool
Apache POI	Spreadsheet import and export functionality in toolkit
Annovention	Annotation discovery
Jackson 2.x	Supports JSON object serialization/deserialization
JAXB 2.0	Supports XML object serialization/deserialization
Jersey 2.x	Supports JAX-RS
JDOM2	XML processing
JaxenXPath Engine	Executing XPath on XML documents
XML Schema Object Model (XSOM)	Parse XML schema documents

Front End

AngularJS	JavaScript framework used for client-side application development
jQuery 1.9	JavaScript library
Bootstrap 3	Front end framework

Database

MySQL	
Oracle	
DB2	
SQL Server	System database

Testing and Code Quality Tools

JUnit	Writing and executing unit tests
JaCoCo	Generates code coverage reports
JMeter	Volume and performance testing
Selenium	Automated regression testing
Zed Attack Proxy	Security vulnerability testing
Sonar	Static code analysis for compliance

Third Party Software and Licenses

This section details the third party software and licenses for [server](#) and [client](#) portions of AgencyPortal 5.2.

Server

GroupID	ArtifactID	Version	License
axis	axis	1.4	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
com.fasterxml.jackson.core	jackson-annotations	2.6.3	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt
com.fasterxml.jackson.core	jackson-core	2.6.3	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt
com.fasterxml.jackson.core	jackson-databind	2.6.3	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt
com.fasterxml.jackson.jaxrs	jackson-jaxrs-json-provider	2.6.3	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt
com.fasterxml.jackson.jaxrs	jackson-jaxrs-base	2.6.3	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt
com.spartial4j	spartial4j	0.3	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
com.sun.xsom	xsom	20140925	CDDL v1.1 / GPL v2 dual license http://glassfish.java.net/public/CDDL+GPL_1_1.html repo
commons-codec	commons-codec	1.7	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
commons-fileupload	commons-fileupload	1.2.1	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
commons-io	commons-io	2.1	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
commons-lang	commons-lang	2.6	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
commons-logging	commons-logging	1.1.3	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
dom4j	dom4j	1.6.1	BSD http://dom4j.sourceforge.net/dom4j-1.6.1/license.html
javax	javaee-web-api	6.0	CDDL + GPLv2 with classpath exception https://glassfish.dev.java.net/nonav/public/CDDL+GPL.html repo A business-friendly OSS license
jaxen	jaxen	1.1.6	BSD - http://spdx.org/licenses/BSD-3-Clause
joda-time	joda-time	2.2	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt repo

GroupID	ArtifactID	Version	License
junit	junit	4.11	Common Public License Version 1.0 http://www.opensource.org/licenses/cpl1.0.txt
org.apache.ant	ant	1.8.3	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
org.apache.ant	ant-launcher	1.8.3	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
org.apache.axis	axis-saaj	1.4	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
org.apache.geronimo.specs	geronimo-stax-api_1.0_spec	1.0	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
org.apache.httpcomponents	httpclient	4.2.3	Apache License LICENSE.txt repo
org.apache.httpcomponents	httpcore	4.2.2	Apache License LICENSE.txt repo
org.apache.httpcomponents	httpmime	4.2.3	Apache License LICENSE.txt repo
org.apache.lucene	lucene-analyzers-common	4.4.0	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt
org.apache.lucene	lucene-analyzers-kuromoji	4.4.0	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt
org.apache.lucene	lucene-analyzers-phonetic	4.4.0	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt
org.apache.lucene	lucene-codecs	4.4.0	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt
org.apache.lucene	lucene-core	4.4.0	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt
org.apache.lucene	lucene-grouping	4.4.0	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt
org.apache.lucene	lucene-highlighter	4.4.0	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt
org.apache.lucene	lucene-memory	4.4.0	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt
org.apache.lucene	lucene-misc	4.4.0	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt
org.apache.lucene	lucene-queries	4.4.0	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt
org.apache.lucene	lucene-queryparser	4.4.0	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt

GroupID	ArtifactID	Version	License
org.apache.lucene	lucene-spatial	4.4.0	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt
org.apache.lucene	lucene-suggest	4.4.0	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt
org.apache.pdfbox	frontbox	1.8.3	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
org.apache.pdfbox	jempbox	1.8.3	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
org.apache.pdfbox	pdfbox	1.8.3	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
org.apache.poi	poi	3.11	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
org.apache.poi	poi-ooxml	3.11	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
org.apache.poi	poi-ooxml-schemas	3.11	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
org.apache.solr	solr-core	4.10.4	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt
org.apache.solr	solr-solrj	4.10.4	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt
org.apache.xmlbeans	xmlbeans	2.3.0	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
org.apache.zookeeper	zookeeper	3.4.6	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
org.codehaus.woodstox	wstx-asl	3.2.7	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
org.glassfish.jersey.containers	jersey-container-servlet-core	2.22.1	CDDL v1.1 / GPL v2 dual license http://glassfish.java.net/public/CDDL+GPL_1_1.html
org.glassfish.jersey.core	jersey-server	2.22.1	CDDL v1.1 / GPL v2 dual license http://glassfish.java.net/public/CDDL+GPL_1_1.html
org.glassfish.jersey.media	jersey-media-json-jackson	2.22.1	CDDL v1.1 / GPL v2 dual license http://glassfish.java.net/public/CDDL+GPL_1_1.html
org.glassfish.jersey.media	jersey-media-json-processing	2.22.1	CDDL v1.1 / GPL v2 dual license http://glassfish.java.net/public/CDDL+GPL_1_1.html
org.hamcrest	hamcrest-core	1.3	New BSD License http://www.opensource.org/licenses/bsd-license.php repo
org.javassist	javassist	3.15.0-GA	MPL 1.1 http://www.mozilla.org/MPL/MPL-1.1.html OR LGPL 2.1 http://www.gnu.org/licenses/lgpl-2.1.html OR Apache License 2.0 http://www.apache.org/licenses/

GroupID	ArtifactID	Version	License
org.jdom	jdom2	2.0.6	<p>Similar to Apache License but with the acknowledgment clause removed https://raw.github.com/hunterhacker/jdom/master/LICENSE.txt repo /*--</p> <p>Copyright (C) 2000-2012 Jason Hunter & Brett McLaughlin. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:</p> <ol style="list-style-type: none"> 1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution. 3. The name "JDOM" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact <request_AT_jdom_DOT_org>. 4. Products derived from this software may not be called "JDOM", nor may "JDOM" appear in their name, without prior written permission from the JDOM Project Management <request_AT_jdom_DOT_org>. <p>In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgement equivalent to the following: "This product includes software developed by the JDOM Project (http://www.jdom.org/)."</p> <p>Alternatively, the acknowledgment may be graphical using the logos available at http://www.jdom.org/images/logos.</p> <p>THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JDOM AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.</p> <p>This software consists of voluntary contributions made by many individuals on behalf of the JDOM Project and was originally created by Jason Hunter <jhunter_AT_jdom_DOT_org> and Brett McLaughlin <brett_AT_jdom_DOT_org>. For more information on the JDOM Project, please see <http://www.jdom.org/>. */</p>
org.glassfish.jersey.containers	jersey-container-servlet	2.17	CDDL license http://glassfish.java.net/public/CDDL-GPL_1_1.html
org.json	json	090211	Provided without support or warranty http://www.json.org/license.html
org.mortbay.jetty	jetty	6.1.14	Apache License Version 2.0 http://www.apache.org/licenses/LICENSE-2.0
org.mortbay.jetty	jetty-util	6.1.14	Apache License Version 2.0 http://www.apache.org/licenses/LICENSE-2.0
org.mortbay.jetty	servlet-api-2.5	6.1.14	CDDL 1.0 https://glassfish.dev.java.net/public/CDDLv1.0.html
org.noggit	noggit	0.5	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
org.ow2.asm	asm	4.1	BSD http://asm.objectweb.org/license.html
org.ow2.asm	asm-analysis	4.1	BSD http://asm.objectweb.org/license.html

GroupID	ArtifactID	Version	License
org.ow2.asm	asm-commons	4.1	BSD http://asm.objectweb.org/license.html
org.ow2.asm	asm-tree	4.1	BSD http://asm.objectweb.org/license.html
org.ow2.asm	asm-util	4.1	BSD http://asm.objectweb.org/license.html
org.slf4j	slf4j-api	1.6.6	MIT License http://www.opensource.org/licenses/mit-license.php repo
stax	stax-api	1.0.1	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo
tv.cntt	annovention	1.2	The Apache Software License, Version 2.0 http://www.apache.org/licenses/LICENSE-2.0.txt repo

Client

Library	Version	License
amcharts	3	amcharts
angularjs	1.3.0	MIT License https://github.com/angular/angular.js/blob/master/LICENSE
angular-bootstrap	0.12.1	MIT License http://angular-ui.github.io/bootstrap/
angular-loader	1.3.0	MIT License https://github.com/angular/angular.js.git
angular-mocks	1.3.0	MIT License http://angularjs.org
angular-resource	1.3.0	MIT License http://angularjs.org
angular-route	1.3.0	MIT License http://angularjs.org
angular-sanitize	1.3.0	MIT License http://angularjs.org
angular-ui-router	1.3.0	MIT License http://angularjs.org
animate.css-master	3.1.1	MIT License http://www.opensource.org/licenses/mit-license.php
bootstrap	3.1.1	MIT License http://www.opensource.org/licenses/mit-license.php
bootstrap-datepicker-master	1.4.1	Apache 2 http://www.apache.org/licenses/LICENSE-2.0.txt
es5-shim	3.1.1	MIT License http://www.opensource.org/licenses/mit-license.php
formatmask		MIT License http://www.opensource.org/licenses/mit-license.php
front-awesome	4.4.0	MIT License http://www.opensource.org/licenses/mit-license.php
html5shiv	3.6.2	MIT License http://www.opensource.org/licenses/mit-license.php
imagemapster	1.2.14-beta1	MIT License http://www.opensource.org/licenses/mit-license.php

Library	Version	License
jquery	1.10.2	MIT License http://www.opensource.org/licenses/mit-license.php
jquery ui	1.10.4	MIT License http://www.opensource.org/licenses/mit-license.php
jquiqery-ui-timepicker	0.3.3	MIT and GPL Licenses http://jquery.org/license
jquery file upload	5.40.1	MIT License http://www.opensource.org/licenses/mit-license.php
jquery-visible-master		MIT License http://www.opensource.org/licenses/mit-license.php
modernizer	2.7.1	MIT License http://www.opensource.org/licenses/mit-license.php and BSD http://asm.objectweb.org/license.html
respond	1.3.0	MIT License http://www.opensource.org/licenses/mit-license.php
select2	3.5.2	Apache License OR the GPL License http://www.apache.org/licenses/LICENSE-2.0 http://www.gnu.org/licenses/gpl-2.0.html
spin	2.0.1	MIT License http://www.opensource.org/licenses/mit-license.php
idletimer	0.9.2	MIT License http://www.opensource.org/licenses/mit-license.php
typeahead	0.10.1	Twitter License https://github.com/twitter/typeahead.js/blob/master/LICENSE

Developer Setup

This section takes you through the installation and setup process needed to start implementing AgencyPortal, and is geared only towards software developers working on a portal implementation.

In this section, you will learn how to:

- get up and running with the necessary [prerequisites](#)
- use the components of [release distribution](#)
- [deploy](#) a project locally via Maven
- [wrap things up](#)
- use the [web-based installer](#)

After reviewing these procedures, you will be prepared to modify and customize your AgencyPortal application using Eclipse.

Continue on to the [Deployment Information](#) section for information on how to set up your application server environment.

Note

Refer to the [Developer Guide](#) section for more information on setting up and customizing particular features within the application.

Prerequisites

To get you up and running with an AgencyPortal development environment, you must install the following tools:

- Eclipse
- Maven
- GruntJS
- a DBMS
- a SCM System
- Tomcat

Install Eclipse

Download and install the Java EE edition of Eclipse 4.4 or above from the [Eclipse Download](#) site if you are using Eclipse IDE. If that is not your IDE of choice, feel free to run the gamut with your preferred IDE.

Install Maven

You need to set up your machine so that you can run Maven commands on the terminal or via the Windows command prompt. Our distribution contains ANT and Maven based artifacts, but we recommend the Maven based setup since it enables developers with a speedy local deployment setup via Eclipse m2e-wtp.

Perform the following to install Maven:

1. Go to the Maven website (<http://maven.apache.org/download.cgi>).
2. Install Maven using the command line. Follow the instructions in the installation guide to add Maven to your path.

Install GruntJS

If you're doing any client-side development, you'll need to have GruntJS set up so that you can use it to minify, aggregate and generate source maps for JavaScript, as well as compile LESS CSS files.

The following includes the instructions to download the prerequisites (refer to the [Client-Side Development](#) topic for information on how to set up GruntJS for a project).

Since the GruntJS lives under the node package management system, you'll also need to download and install Node.js from the [Node.js](#) site.

Install a DBMS

AgencyPortal supports communication over JDBC with MySQL, SQL Server, Oracle and DB2. Make sure you have one of these running somewhere that your local environment can reach.

1. Create a database for AgencyPortal if you don't already have one.
2. Set up a user account for the portal. If you're using the automatic database installer feature (refer to the Initializing the Database section in the [Database Configuration](#) topic), make sure the user has been granted permissions for read, write and create access.

Install a SCM System

If you're using Subversion or Git as your Source Code Management (SCM) system, you should install the necessary tools to communicate with your repository.

If you're using Subversion, we recommend using both of the following:

1. Eclipse - it can't be used for 100% of cases, but Eclipse has a pretty good Subversion integration plugin called Subclipse. You can download a copy of Subclipse (at least version 1.10.x) from [here](#).
2. Tortoise SVN - Since IDE integration for source control isn't always adequate; you'll need a subversion client that runs outside of your IDE. For Windows machines, we recommend Tortoise SVN (version 1.9 and up).

Note

Versions is a good subversion client for Mac OS machines.

Download Tomcat

We recommend that developers use Tomcat for running AgencyPortal on their local machine. Even though, sometimes, it can be prudent to locally run the AppContainer that you'll be running in production, heavy containers like WebSphere and WebLogic can often grind your local environment to a halt.

Download the ZIP distribution of Tomcat (version 7+). No need to do anything with the Tomcat installation at this moment; simply extract the file to a location of your choice.

Release Distribution

There are three parts to the SDK 5 distribution:

1. **AgencyPortal SDK Core**

The SDK is made up of JAR files, web content, SQL packages and other configuration files. The SDK is distributed as a zip file; however, these artifacts are made available to you in your Maven repository:

- apbase.jar
- apsecurity.jar
- apsecurityreference.jar (only use this as a quick start; it should never go to production)
- apwebapp.jar
- toolkit.jar
- apwebcontent.war

2. **AgencyPortal Integration Kits** (template integration source code, along with some configuration snippets)

By default, the intkit core source code is something projects will need to get up and running; other intkits can be brought in as needed.

3. **AgencyPortal Templates** (product definition templates with source code)

By default, the template core artifact (a bundle with shared commercial and account templates) is needed to get up and running. Other templates can be brought in as needed.

To get an implementation team up and running quickly, we also distribute sample project ZIP files. There is a Maven based sample and an Ant based sample. These files have a standard directory structure with the necessary components listed above (SDK, intkit core and template core), along with a functional build script (build.xml or pom.xml).

This section includes the following procedures to set up these components of the SDK:

- [Setting Up Your Portal 5 Project](#)
- [Importing the Project into Eclipse](#)
- [Configuring Integration Kits](#)
- [Installing LOB Templates](#)

Setting Up Your AgencyPortal Project

Note

We expect that only one member of the team will go through the setup process and subsequent publish to SVN/Git. Ideally, this should be performed at the start of a project. Other team members will check out the project from your SCM system and import it into Eclipse as a Maven project. Refer to the [Importing Maven Projects](#) procedure on the Sonatype website for more information.

This topic details what you'll need to do to set up your AgencyPortal project. The artifacts you'll need are made available to you via your FTP site. The quickest way to get set up is to download the sample Maven project. You'll find this under:

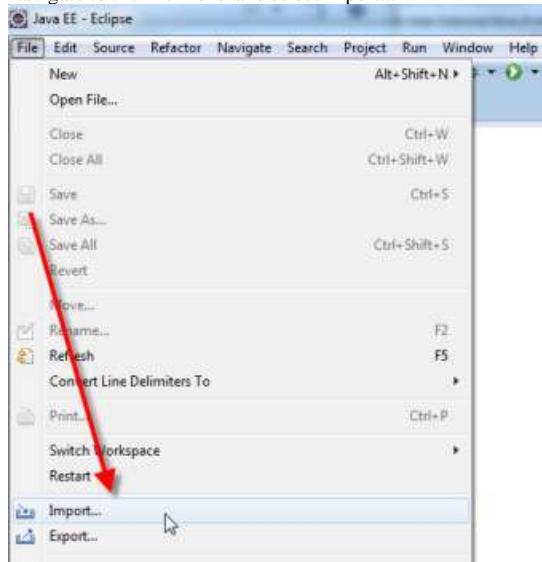
AgencyPortal_Templates/Core_Templates

1. Download and extract the sample-project-{version}-maven.zip file and extract it to a location of your choice.
2. Open the provided pom.xml file and update the values for groupId, artifactId and version to values of your choice. Refer to the [Guide to Naming Conventions on GroupId, ArtifactId and Version](#) topic on the Maven website for more information on Maven's conventions for naming your new project artifacts.
3. Within the pom.xml file, configure the repository location for retrieving agencyportal artifacts. Refer to the repositories section of the pom file.
4. Your project is now ready to import into Eclipse. Refer to [Importing the Project into Eclipse](#) to complete this task.

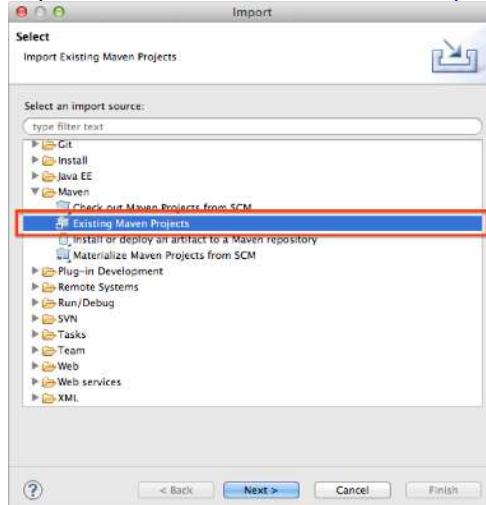
Importing the Project into Eclipse

Perform the following to import the project into Eclipse as a Maven project:

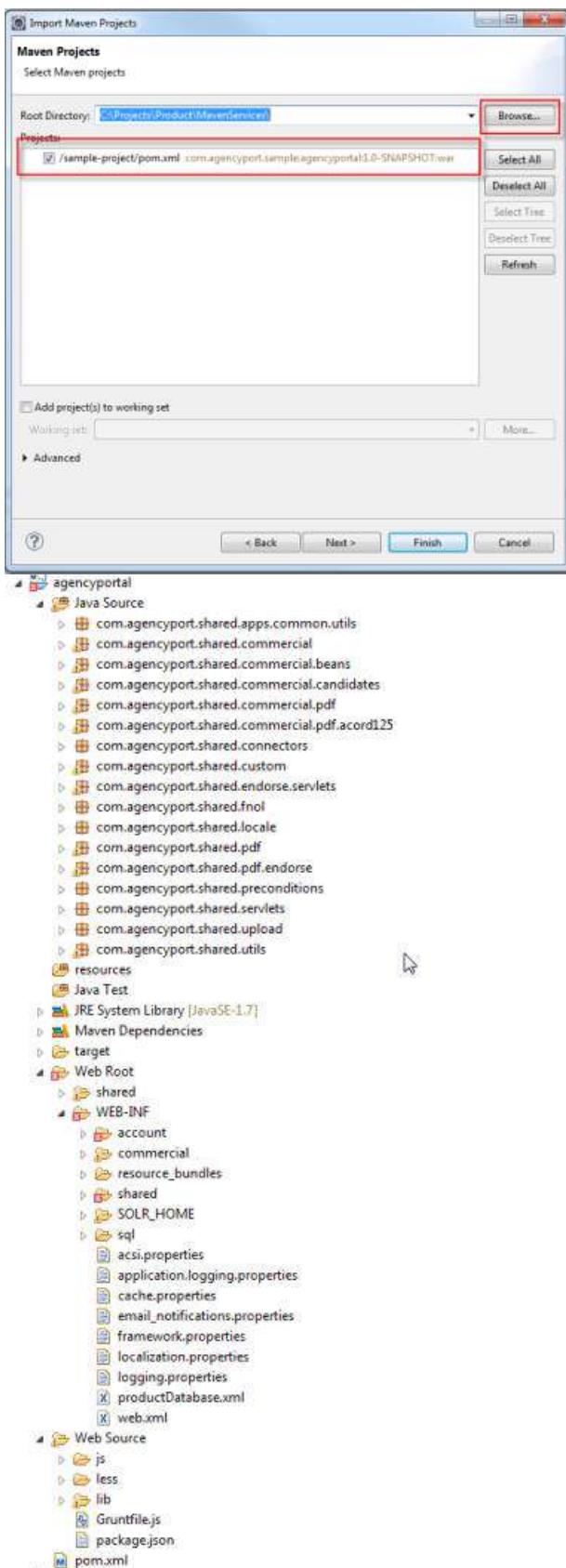
1. Open Eclipse in a workspace of your choice.
2. Navigate to the File menu and select Import...



3. Select Existing Maven Projects and click Next (if you don't see Existing Maven Projects, make sure you have the correct version of Eclipse Java EE edition installed. Refer to the [Prerequisites](#) topic for more information.).



4. Click the Browse button to browse to the location where you created your project (this is the location where you previously extracted the project).
5. Check the check box next to the project and click Finish.



Configuring Integration Kits

Out of the box, the sample AgencyPortal5.2 project has a dependency on some core integration kit source files, which have not been included. If you're going to integrate with Connect5, we recommend you perform the following:

1. Download the integration kit reference implantation assets core artifact from your distribution (`intkit_ria-<version>-core.zip`).
2. Add the source code from the zip file to your new project's Java Source folder.

Alternatively, if you're not ready to work with integration kit assets, then you can modify `web.xml` to remove/comment out the dependencies on classes from the `com.agencyport.intkit.core.*` package.

Installing LOB Templates

AgencyPortal templates (other than shared, commercial and account) are not packaged into the sample project. This is to allow developers to selectively include the line of business (LOB) templates of their choice.

Perform the following to install and include a LOB template(s):

1. Download the template zip files from your FTP site.
2. Extract the zip files into a temporary directory outside of your project. Template zip files contain three main things: source code, web artifacts and a productDatabase.xml fragment. Import these items into the project manually.

Note

Template zip files can be imported at runtime by the web-based installer; however, this approach would be for validation purposes only since template source code is not imported.

1. Copy the contents of the template's src folder into your project's Java Source folder.
2. Open the productDatabase.xml fragment from the template zip file.
3. Copy the <product/> definition and paste it into your project's productDatabase.xml file (located under Web Root/WEB-INF).
4. Move the remaining folders (which are Web Content folders) in your template zip file into your project's Web Root folder.

Note

Some templates come with SQL files found under the WEB-INF directory. These files can be executed manually or by using the automatic_database_insaller.

3. Refresh your workspace in Eclipse.

Deploying Project Locally via Maven

We recommend you deploy your newly packaged project locally for validation/testing before importing the project into Source Code Management (SVN/Git).

Perform the following to get the app up and running on a local Tomcat server via Eclipse's WTP plugin:

- [Configure the Database](#)
- [Deploy Solr Locally](#)
- [Set Required Properties](#)
- [Set Up Tomcat Server](#)
- [Run the App](#)

Database Configuration

Perform the following to point your application to your database after you choose your database vendor and set up a portal database that you can access from your machine:

Note

The prerequisite for this step is that you've already created a database for AgencyPortal. If you're using the automatic database installer feature described below, make sure there is a user on the database server with read, write and create access.

1. Before the deployed application can communicate with the database, you need to set the following application properties in framework.properties:
 - The db_table_prefix needs to be modified if your JNDI user is different than the owner of the database tables.
 - The datasource_prefix needs to be verified if you are not using Tomcat.
 - Set the database_agent_class_name property appropriately. Update database_agent_class_name to whichever one of the values that the commented section above corresponds to the type of database you are using (e.g., if you're using SQL server, you want to set it to com.agencyport.database.SQLDatabaseAgent).

Note

If the properties you're trying to set apply to your environment, but not everybody else's, you can create local property override files that will be applied only to you at runtime. Create a properties file prefixed with your username (`${username}.framework.properties`) under WEB-INF and add property overrides that apply to your environment only.

2. Configure Tomcat with a JNDI resource named agencyportal (this default name can be changed in framework.properties) and add the connection parameters for your database.

- a. Create a META-INF folder under Web Root and add a context.xml file, such as the following:

```
<Context reloadable="false" allowLinking="true">

    <!-- MySQL Server -->
    <Resource name="agencyportal"
        auth= "Container"
        type= "javax.sql.DataSource"
        description="MySQL database for
        AgencyPortal"
        maxActive="100"
        maxIdle= "30"
        maxWait= "10000"
        username= "dbusername"
        password= "dbpassword"
        driverClassName="com.mysql.jdbc.Driver"
        url=
        "jdbc:mysql://hostname:3306/mymodernportaldb" />
</Context>
```

3. Add the appropriate JDBC drivers to your Tomcat/lib folder.

Note

When working with a SqlServer database, please use the sqljdbc4 JAR provided by Microsoft.

Initializing the Database

You can optionally use the automatic_database_installer to create the agencyportal database tables during application startup. The following properties are available in the framework.properties to enable this feature.

Property	Description	Default Value	Recommended Setting (if any)
automatic_database_installer_enabled	Indicates whether to create the agencyportal tables during application startup.	false	
base_sql_directory	The directory where SQL files known to the installer live.	none	
strict_datasource_lookup		none	
ordered_sql_package_list	A list of SQL "packages" for the installer to use, specified in execution order. A "package" is a sub-directory of the base_sql_directory and must contain a sub-directory for the database type, as well as a sqlOrder.properties file that specifies the files to execute with the orderedSqlScriptList property.	none	

All of the SQL scripts needed to initialize the database have been packaged into SQL packages within the WEB-INF/sql folder. The base SDK packages are:

- apwebapp
- apsecurityreference
- apdashboard

A SQL package is just a folder with a bunch of SQL files and a `sqlOrder.properties` file that lists the order that the SQL files need to be executed in.

The automatic database installer will inspect the `ordered_sql_package_list` property in `framework.properties` and run the packages configured there. By default, this property is configured to include the base SDK packages; however, in addition to the above SDK packages, some templates ship with SQL packages too. These packages would be dropped into that directory during template import. Any templates that ship with SQL packages also append to the `ordered_sql_package_list` property in their LOB properties file. Therefore, the automatic database installer should be able to run all SDK and template SQL packages without requiring any additional configuration.

If you choose to enable the `automatic_database_installer`, it will be engaged during application startup if these two things are true:

1. The application can establish connectivity to the database, as configured (via the JNDI datasource or directly through application properties if the `force_use_jdbc` property is set to true).
2. If the application fails to run a simple select query against the `work_item` table.

If project teams need to create their own SQL packages and have the `automatic_database_installer` execute them, they only need to create the SQL package folder, its `sqlOrder.properties` file and update the `ordered_sql_package_list` property.

Of course, you can do all of this manually by running the SQL scripts located within each SQL package in the correct order:

- apwebapp/<db>/create_tables.sql
- apwebapp/<db>/GetNextkeyValue.sql
- apsecurityreference/<db>/create_acsi_tables.sql
- apsecurityreference/<db>/load_acsi_nextkey.sql
- apsecurityreference/<db>/load_acsi_tables.sql
- apsecurityreference/<db>/load_acsi_user.sql
- apdashboard/<db>/create_tables.sql
- apdashboard/<db>/dashboard_insert.sql

Local Solr Deployment

Before we deploy to Tomcat, you need to set up a home for the Solr index. The project already contains the necessary Solr configuration files to get started under the following directory:

```
solr/SOLR_HOME
```

All you need to do is copy this SOLR_HOME directory to somewhere on the filesystem outside of the project codebase and outside of the deployment directory. There are multiple ways to register the SOLR_HOME directory, but for now, just use framework.properties.

Set the solr_home property in framework.properties (or \${username}.framework.properties) to something like the following:

```
solr_home=C:/projects/example-prj/SOLR_HOME
```

Note

For more information on setting up Solr, refer to the [Solr Setup](#) topic.

Required Properties

There are some additional required properties that must be set before you can deploy the application. For various reasons, such as security, upload integration and Turnstile integration, the application needs to be aware of its own URL at runtime. The URL registers by setting a few properties:

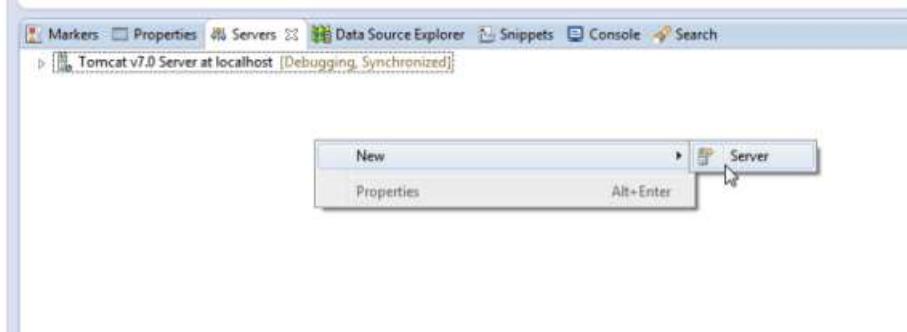
my_portal_app	This is the master property containing the entire URL. The default is set to be the sum of the parts below. Example <code>my_portal_app=\${my_portal_app_protocol}://\${my_portal_app_domain}:\${my_portal_app_port}/\${APPLICATION_NAME}</code> Instead of altering this property, consider changing the different URL parts below.
my_portal_app_protocol	Usually 'http' or 'https'
my_portal_app_domain	The domain with any prefixes without any port number. Example developer.agencyport.com
my_portal_app_port	Whatever port your application container will be listening to. The default for Tomcat is '8080'.
APPLICATION_NAME	The context of the application. Usually matches the WAR file name, but is configurable on most application containers. For the purposes of this setup guide, 'agencyportal' is used.

Using this set up you'll also need to set the `output_dir` property to point to some other location outside of the `webapps` folder. This is because the Maven build will be prevented from doing automatic redeploys due to a file systemlock.

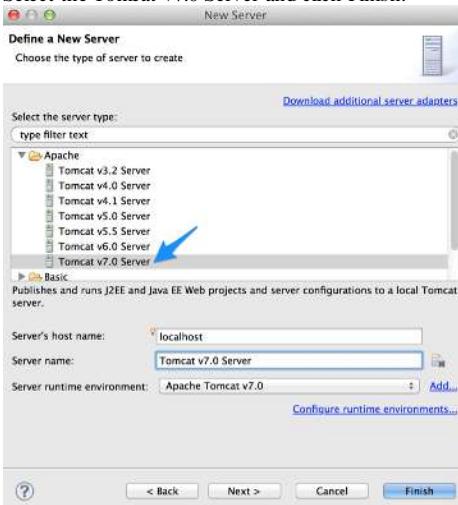
Setting Up a Tomcat Server

Perform the following to set up a new server and point it to a local Tomcat instance:

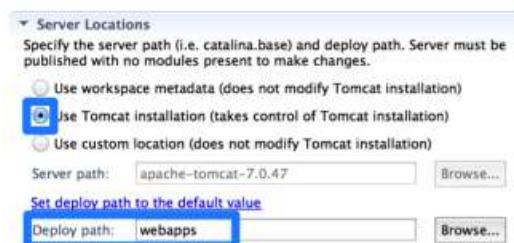
1. Make sure your local Tomcat installation has the necessary JDBC jars made available.
2. Set up a new server within Eclipse and point it to your Tomcat installation:
 - a. Right click on the Servers view of the Java EE perspective.
 - b. Select New.
 - c. Click Server.



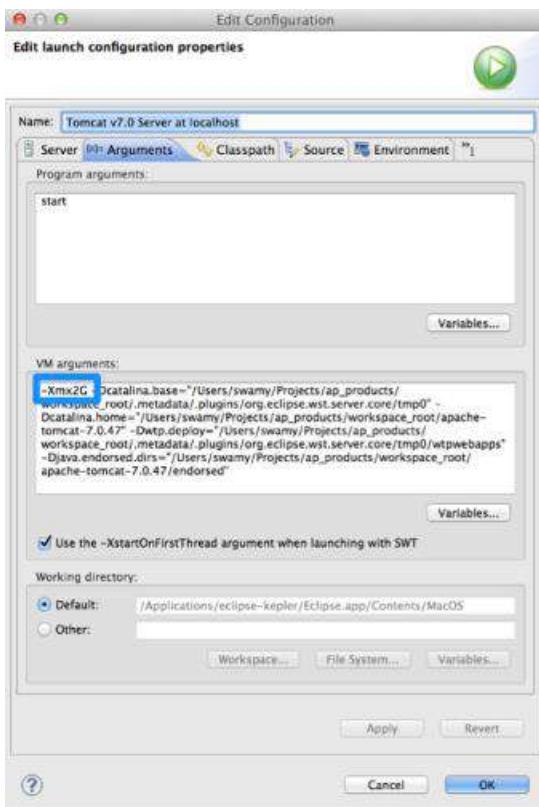
3. Select the Tomcat v7.0 Server and click Finish.



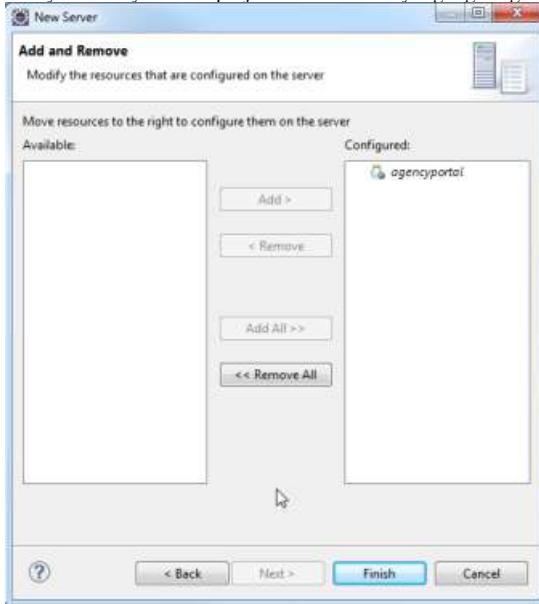
4. Double click the new server you created to edit its properties. Select the Use Tomcat installation (takes control of Tomcat installation) option in the server settings and change the Deploy path to webapps.



5. Increase the memory by adding -Xmx2G to the VM arguments.



6. Add your newly created project to the server by highlighting an available resource and clicking the Add button.



7. Click Finish.

Running the App

Perform the following to run the app:

1. Start your server.
2. Test the server at http://localhost:8080/<your_project_name>/DisplayLogonForm.
3. Enter your username and password (start with agent/password).

Wrapping Things Up

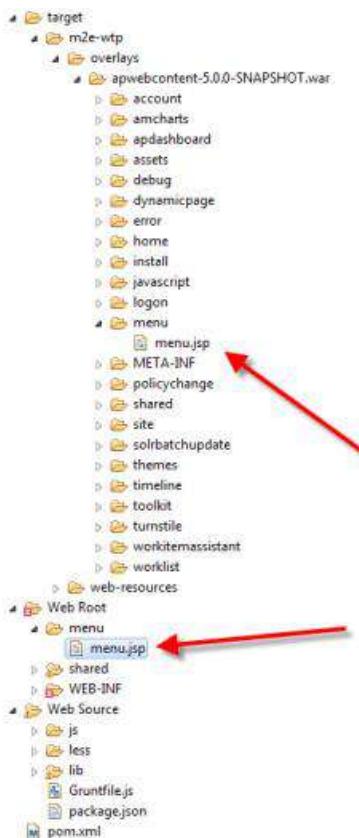
Now that the application is running locally, there are a few things to cover before we wrap things up. In this topic, we'll discuss how to work with the SDK's web artifacts, how to work with the web sources we've distributed and, finally, describe what needs to go into your SCM system.

Working with SDK Web Artifacts

The Maven build pulls in the SDK's web artifacts from the repository at build time. You can view these artifacts within the target folder after a build, but any changes you make to these files are blown away after the next build.

The way to overwrite the SDK's version of a file is to create your own version of the file under the Web Root folder with a matching directory structure and file name. This way, when build occurs, your version of the file wins the conflict. This practice is commonplace for files, such as site/site_shell.jsp, home/home.jsp and menu/menu.jsp.

Example



Working with Web Sources

The SDK ships with web sources, which live in their own directory and never directly make it into the WAR. These web sources are JavaScript files that are concatenated together and minified, as well as LESS files that are compiled to CSS.

We have provided you with the means to generate these combined/compiled files so you can add your own files into the mix.

Note

You should not modify any of the existing files under Web Sources (apart from Gruntfile.js). You should treat the files provided to you as you would any third party library code (i.e., if you modify and we change it, then you'll have trouble upgrading to our latest code). As a best practice, when adding new files, add them to a new directory under Web Source called custom.

The source files are combined/compiled by a tool called GruntJS (<http://gruntjs.com/>). The real power of this tool is its ability to generate HTML5 source maps (<http://www.html5rocks.com/en/tutorials/developertools/sourcemaps/>).

You should have already downloaded nodejs as a [prerequisite](#), so you should be able to open a terminal/command prompt and navigate to the Web Source of your project and then run the following two commands to install GruntJS:

```
npm install
```

```
npm install -g grunt-cli
```

Note

By default, this command will drop some files into a directory called `node_modules`. You should not commit this directory to SVN (instead, add it to your SVN or Git ignore list).

Now that GruntJS is installed, you should be able to run the following command to run the build:

```
grunt
```

By default, the Grunt build is set up to output the following files into your project's Web Root/assets/directory:

- js/agencyportal.js
- js/agencyportal.js.map
- js/agencyportal-body.js
- js/agencyportal-body.js.map
- themes/agencyportal/agencyportal.css
- themes/agencyportal/agencyportal.css.map

Since these files are dropped into the Web Root directory, Maven takes care of getting them into the WAR file for us. For this reason, the Grunt build does not need to know anything about where or how you're deploying your code (locally or otherwise). One drawback to this approach is that you'll need to commit these build artifacts to SVN/Git. Otherwise, somebody will have to run the Grunt build, even if they're not working on JavaScript or LESS.

Note

You may have noticed that these map files can get quite large. The good news is that their presence in the WAR file will not have any negative impact on page load times for regular users. This is because the browser knows not to download source map files until the developer console is opened.

For more information, refer to the [Client-Side Development](#) topic for more information.

Importing the Project into SCM

At this point, we're ready to commit this project to a SCM system so that other team members can work on the project. The process is pretty straightforward for both Subversion or Git if you're familiar with the tool.

1. Import the entire project you've created into your SCM repository.
2. Checkout the project from the repository and place into a different location.
3. In the newly checked out project, delete the following (so you can see the files and directories whose names begin with '.', refer to your operating system's instructions for showing hidden files and folders):
 - .settings folder
 - target folder
 - .project file
 - .classpath file
 - Web Source/node_modules folder
4. Commit the delete.
5. Add those directories to the SVN or Git ignore list.

Web-Based Installer

AgencyPortal includes a web-based installer feature that can be used to verify and explore the distribution and validate that it works in your deployment environment. If you are planning on making permanent substantive changes, then you need to follow the [Developer Setup](#) procedures to create a project before we get started.

Prerequisites

The AgencyPortal installer provides easy application setup with a pre-built war file. Prior to installing the AgencyPortal application with the installer, the following tasks must be completed.

1. A database instance must be available to the application. The database instance cannot have any existing AgencyPortal tables defined or the install will fail. Read, write and create permissions must be granted to allow the installer to create and populate AgencyPortal tables.
2. An application server where the AgencyPortal war file can be deployed is required. The application server must have a JNDI resource named `agencyportal` configured to communicate with the database.
3. Download the line of business templates you need from your FTP site. You'll need these zip files once you start walking through the installer. AgencyPortal 5.2 version templates are recommended.
4. Download the `agencyportal.war`, which has been prepred for your container, from the FTP site. Deploy `agencyportal.war` on your application server. When deploying, ensure that the context root for the web application is set to 'agencyportal':
 - on some containers, this must be configured explicitly
 - on others, you only need to ensure that you do not rename the war file

To access the installer and finish setting up your application, use the following URL:

`http://<host name>:<port number>/agencyportal/Install`

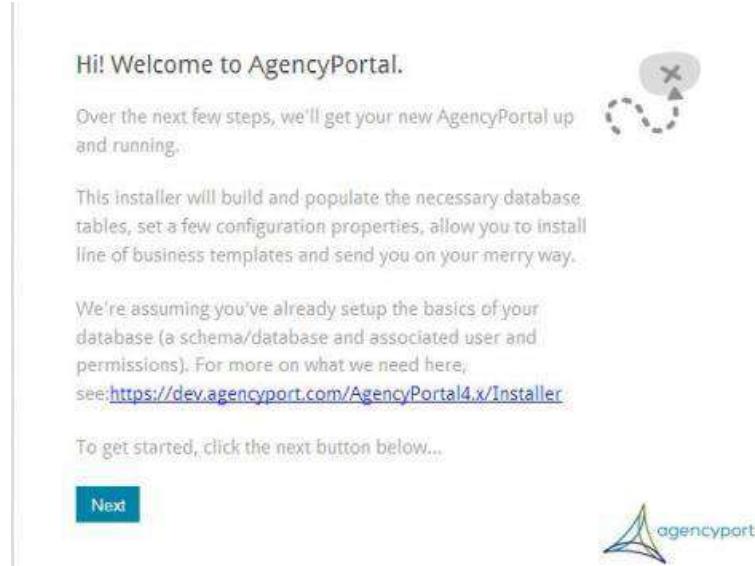
Tip

The installer will walk you through four screens of setup information and tasks. Do not quit the installer until you have launched your AgencyPortal application. After you are finished with the installer, and have launched your portal, you can log in using agent/password.

Using the Installer

The screen shots below show the sequence you will follow to use the installer and complete the setup of your application.

Getting Started Screen



Database Setup Screen

Let's Build A Database!



Now we'll build the database tables needed by AgencyPortal and populate them with some initial requisite configuration info. In order to make sure this runs the way it's supposed to, select the type of database you're using.

Note: The database user you've configured in your JNDI settings for this web app must have permission to create tables for this to work. If this sounds like nonsense, see: <https://dev.agencyport.com/AgencyPortal4.x/Installer>



Template Installation Screen

Tip

Add multiple lines of business by clicking "Browse" and then "Go" for each template you want to install. You should not leave the Template Installation Screen until you have installed all of your templates. Any required SQL scripts per template will be run against the database at this phase of the install. Additionally, if you are installing the Commercial Package template, there are some guidelines to follow. You must install all of the following lines of business before you install the Commercial Package template: Commercial Property, General Liability, Commercial Auto, Commercial Crime, and Inland Marine.

Line of Business Templates



Cool, the database is good to go.

Now, let's upload any line of business templates (i.e. personal auto, commercial package, etc) you want to install on this AgencyPortal. If you haven't already downloaded them, you can find any templates you've licensed on the [Dev Zone](#)

Import Template

Already Installed:	
Product	Version



Wrap Up Screen

Tip

You will need to restart your application server before checking the check box.

Almost There!



Congratulations! AgencyPortal has been installed. Before you launch the application, **please restart your application server**. Once finished, return here and **launch your application**.

If you need to back up and [install more templates](#) you can do so now. Templates can always be installed later with the AgencyPort Toolkit.

OK, I've Restarted



Deployment Information

After you install and set up the components of AgencyPortal outlined in the [Developer Setup](#) section, it's time to start setting up your application server environment. AgencyPortal supports the following application server environments:

- [WebSphere 8.5](#)
- [Tomcat](#)
- [WebLogic](#)
- JBoss

This section includes detailed requirements and settings needed to set up your environment for application deployment. The information included in this section is targeted to software developers working on an AgencyPortal implementation project.

WebSphere 8.5

Since AgencyPortal is packaged as a JEE compliant web application, it is compatible with any JEE compliant web container. However, to get the application up and running in WebSphere, you must prepare a WebSphere 8.5 environment and ensure the WAR/EAR file and container are properly configured. This is due to the characteristics of WebSphere, such as its default classloading policies, default boot strap process, and bundled IBM JDK.

Refer to the following requirements below for a comprehensive list of the changes you must make:

- [AgencyPortal Web Application Archive Requirements](#)
- [WebSphere Configuration Requirements](#)

AgencyPortal Web Application Archive Requirements

The following are the requirements for the AgencyPortal web application archive:

- Some of AgencyPortal's JAR file dependencies, which are typically distributed as part of the product, are not needed since they collide with JARs provided as part of the container.
 - o Do not include the following JAR artifacts in any AgencyPortal WAR or EAR file that you are about to deploy:
 - saaj
 - geronimo-stax-api_1.0_spec-1.0
 - stax-api
 - activation
 - taglibs-standard-impl
 - javax.mail

Note

Depending on the method employed to get to a WAR file, some (or all) of these JAR files may already be excluded. The sample pom files contain mutually exclusive profiles, which take care of these exclusions.

- Ensure your database_prefix application property is set appropriately. While some application containers bind JNDI names in the java:comp/env/ namespace by default, WebSphere supports empty prefixes.
- The cryptography library provided as part of IBM's JDK mandates the usage of an alternate keystore file within our application. The SDK's apbase.jar is packaged with this alternate keystore file (ibmacsi.key) so all you have to do is point to it within application properties (e.g., acsi.properties) as follows:
 - security.id_store.keystore_filename=(asResource)keystore/ibmacsi.key
 - com.agencyport.security.encryption.EncryptionHelper.<encryption-type>.keystore_filename={security.id_store.keystore_filename} (where <encryption-type> is the encryption algorithm used, such as Blowfish).If you are using security reference tables, you must also regenerate the epassword column values in the subject table using the following command line utility. Our security reference uses the login ID as the salt value.

```
C:\TomcatRoot\tomcat1-7.0.50\webapps\aptemplates\WEB-INF\lib>java -cp apbase-5.2.0-SNAPSHOT.jar com.agencyport.security.encryption.BuiltinEncryptionAgent -h
```

Usage:java com.agencyport.security.encryption.BuiltinEncryptionAgent

- mode [decrypt] or [encrypt] or [create] or [read], encrypt is the default if not supplied, use [create] to create keystore for application instance, use [read] to extract the key value from the key store.
- keystore <file name of key store file>; default is (asResource)keystore/acsi.key
- salt <salt value>; no default, must be used for encrypt mode
- value <value to encrypt or decrypt>; no default, when in create mode this is the user supplied encryption key value that will be stored in the keystore file
- encryption_type <encryption algorithm>, default is Blowfish
- cipher_transformation <cipher transformation>; optional, defaults to the algorithm of choice with the default mode and padding arguments implemented by the cipher provider
- terse <true or false>; default is false, only applies to modes of encrypt and decrypt
- password <password value>; default supplied if not entered, applies to all modes

Example

The following is an example of a generated epassword value for the login ID 'agent'.

```
C:\TomcatRoot\tomcat1-7.0.50\webapps\apttemplates\WEB-INF\lib>java -cp apbase-5.2.0-SNAPSHOT.jar com.agencyport.security.encryption.BuiltinEncryptionAgent -mode encrypt -keystore (asResource)keystore/acsi.key -value password -salt agent
```

Reading key store file: (asResource)keystore/acsi.key

password encrypted is: 8S1AUtuOHF4ZRsekmr94YGCul0d/59+y YkME8jPdPUQ= (salt value = agent)

In this example, you would need to replace the `-keystore` parameter with the `ibmcsi.key` keystore and make sure you are running within the IBM JDK.

It is very important you generate your own keystore file for security reasons. Refer to the [Encryption](#) topic in the Security section for more information.

There are some additional required properties that must be set before you can deploy the application. For various reasons, such as security, upload integration and Turnstile integration, the application needs to be aware of its own URL at runtime. The URL registers by setting a few properties:

my_portal_app	<p>This is the master property containing the entire URL. The default is set to be the sum of the parts below.</p> <p>Example</p> <pre>my_portal_app=\${my_portal_app_protocol}://\${my_portal_app_domain}:\${my_portal_app_port}/\${APPLICATION_NAME}</pre> <p>Instead of altering this property, consider changing the different URL parts below.</p>
my_portal_app_protocol	Usually 'http' or 'https'
my_portal_app_domain	<p>The domain with any prefixes without any port number.</p> <p>Example</p> <pre>developer.agencyport.com</pre>
my_portal_app_port	Whatever port your application container will be listening to. The default for Tomcat is '8080'.
APPLICATION_NAME	The context of the application. Usually matches the WAR file name, but is configurable on most application containers. For the purposes of this setup guide, 'agencyportal' is used.

WebSphere Configuration Requirements

The following is an overview of the requirements needed for WebSphere configuration:

- AgencyPortal 5.2 requires JDK 1.7. As such, you must ensure that the “IBM WebSphere SDK Java Technology Edition Version 7.x” is made available to your WebSphere container (via the IBM Installation Manager).
- AgencyPortal uses Apache Solr to provide enhanced search capabilities.
 - A SOLR_HOME directory that contains some configuration files and ultimately houses the search index is provided to you alongside the product.
 - The SOLR_HOME directory needs to be placed somewhere on the Web Container’s machine in a directory that will not be wiped out when a new version of a WAR/EAR file is deployed.
 - The deployed AgencyPortal application must have write access into this directory.
For more detailed procedures, refer to the [Solr Cloud](#) topic in the Solr Setup section.
- AgencyPortal’s Solr request servlet-filter will not work correctly unless the container invokes its init function during application bootstrap. By default, WebSphere will not do this; however, you can configure this by setting a [custom property](#) at the “Web Container” level:
 - Navigate to the custom properties menu in the WebSphere admin console (via Servers > Application Servers > server_name > Web Container settings > Web Container > customproperties).
 - Set the following property: `com.ibm.ws.webcontainer.invokeFilterInitAtStartup=true`.
Refer to the Configuring the Server section in the [Additional Information](#) topic for more detailed information.

- To ensure that AgencyPortal's embedded JAX-RS implementation is used instead of WebSphere's implementation, set the following property to disable WebSphere's implementation:
 - Navigate to the custom properties menu in the WebSphere admin console (via Servers > Application Servers > **server_name** > Java and Process Management > Process Definition > Java Virtual Machine > Custom properties).
 - Set the following property: `com.ibm.websphere.jaxrs.server.DisableIBMJAXRSEngine=true`.
- Set the parent last classloading at the server level [via Servers > Application Servers > **server_name** > Class loading mode > Classes loaded with local class loader first (parent last)].
- Set the following systemproperties to get the expected behavior (via Servers > Application Servers > **server_name** > Java and Process Management > Process definition > Java Virtual Machine > Generic JVM arguments):
 - `-Djava.util.logging.configureByLoggingPropertiesFile=true` – this is necessary on WebSphere for the AgencyPortal logging framework to perform as expected.
 - `-Dagencyport.env=some_string` – (optional) use this so the application load environment has specific resources (i.e., specific to this container only). The most common use case for this feature is to provide an environment specific properties file.

Note

You can configure the systemproperty name to use `agencyport.env`, by default, in `web.xml`.

- `-Dsolt.solr.home=path_to_solr_home` – (optional) this is one of a few different ways to tell the application where to find the `SOLR_HOME` directory.
- Set up a JNDI datasource that points to a database containing the AgencyPortal tables:
 - This is a standard JNDI setting for most database vendors except for DB2, which requires you to change the `resultSetHoldability` property to DB2's default of 1 (`HOLD_CURSORS_AT_COMMIT`) [WebSphere's default is 2 (`CLOSE_CURSORS_AT_COMMIT`)]. Refer to the Configuring JNDI section of the [Additional Information](#) topic for more detailed information.
 - You can make the property change via Resources > JDBC > Data Sources > `db2_datasource_name` > Custom Properties > `resultSetHoldability`.

Note

If this is an XA datasource, define a new customproperty on the data source (where `property_name='downgradeHoldCursorsUnderXa'` and boolean value = true).

Refer to the [Additional Info for DB2 Users](#) topic for more detailed information.

- During the AgencyPortal archive deployment process, bind the application to the aforementioned JNDI datasource.
- Before starting the application, configure the classloader settings at the application and module level:
 - via WebSphere enterprise applications > `application_name` > Class loading and update detection > Classes loaded with local class loader first (parent last).
 - via WebSphere enterprise applications > `application_name` > Manage Modules > `module_name` > Class loader order > Classes loaded with local class loader first (parent last). Refer to the Class Loader Settings section in the [Additional Information](#) topic for more detailed information.

Additional Information

If additional information is needed to accompany the configuration procedures, refer to the following detailed instructions:

- [Configuring the Server](#)
- [Setting Generic JVM Arguments](#)
- [Configuring JNDI](#)
- [Creating a JNDI JDBC Provider and Data Source](#)
- [Deploying the WAR](#)
- [Application Level Class Loader Settings](#)
- [Module Level Class Loader Settings](#)
- [Server-Level Classloading Settings](#)

Configuring the Server

The following section contains additional information on configuring the server mentioned in the WebSphere Configuration Requirements section of the [WebSphere 8.5](#) topic. Perform the following to configure the server:

1. Navigate to the appropriate section (see sample below) and select the server to configure.

The screenshot shows the 'Application servers' page. On the left, there's a sidebar with 'Guided Activities', 'Servers' (selected), 'Server Types' (with 'WebSphere application servers' expanded), 'Applications', 'Services', and 'Resources'. The main area has a heading 'Application servers' with the sub-instruction 'Use this page to view a list of the application servers in your environment'. It includes a 'Preferences' section with icons for '+' and '-' and a table titled 'You can administer the following resources:' showing 'server1' and 'test-app01portal5' with a total of 1 item.

2. Set the web container settings under "Custom properties" as indicated in the screen shots below:

The screenshot shows the 'Configuration' tab for 'server1'. It has tabs for 'Runtime' (selected) and 'Configuration'. Under 'General Properties', fields include 'Name' (server1), 'Node name' (test-app01portal5), 'Run in development mode' (unchecked), 'Parallel start' (checked), 'Start components as needed' (unchecked), and 'Access to internal server classes' (set to 'Allow'). Under 'Container Settings', there are sections for 'Session management', 'SIP Container Settings', 'Web Container Settings' (selected), 'Web container' (highlighted with a green box), 'Web container transport chains', 'Portlet Container Settings', 'EJB Container Settings', 'Container Services', and 'Business Process Services'. The 'Applications' section is also visible at the bottom.

Application servers > server1 > Web container
Use this page to configure the web container.

Configuration

General Properties

- Default virtual host: default_host
- Enable servlet and command caching.
- Disable servlet request and response pooling

Asynchronous Servlet Properties

Additional Properties

- Asynchronous Request Dispatching
- Custom properties**
- Web container transport chains
- Session management

- Set the properties to com.ibm.ws.webcontainer.invokeFilterInitAtStartup=true and as indicated in the screen shots below:

Application servers > server1 > Web container > Custom properties > New...
Use this page to specify an arbitrary name and value pair. The value that is specified for the name and value pair is a string that can set internal system configuration properties.

Configuration

General Properties

+ Name: com.ibm.ws.webcontainer.invokeFilterInitAtStartup
* Value: true

Properties specified above

Apply OK Reset Cancel

Setting Generic JVM Arguments

Perform the following to set generic JVM arguments:

- Set the argument as indicated in the screen shots below:

Guided Activities

Servers

Server Types

- WebSphere application servers**
- WebSphere MQ servers
- Web servers

Applications

Services

Resources

Calendar

Application servers

Use this page to view a list of the application servers in your environment. Click a server to change the status of a specific application server.

Preferences

You can administer the following resources:

server1	test-app01portal5
---------	-------------------

Total 1

Runtime Configuration

General Properties

Name: server1
Node name: test-app01portal5
 Run in development mode
 Parallel start
 Start components as needed
Access to internal server classes: Allow

Server-specific Application Settings

Classloader policy: Multiple
Class loading mode: Classes loaded with parent class loader first

Buttons: Apply, OK, Reset, Cancel

Container Settings

- Session management
- SIP Container Settings
- Web Container Settings
- Portlet Container Settings
- EJB Container Settings
- Container Services
- Business Process Services

Applications

- Installed applications

Server messaging

- Messaging engines
- Messaging engine inbound transports
- WebSphere MQ link inbound transports
- SIB service

Server Infrastructure

- Java and Process Management
 - Class loader
 - Process definition Process definition
 - Process execution

Application servers

Application servers > server1 > Process definition

Use this page to configure a process definition. A process definition defines the command line information necessary to start or initialize a process.

Configuration

General Properties

Executable name:
Executable arguments:
Start command:
Start command arguments:

Additional Properties

- Java Virtual Machine
- Environment Entries
- Process execution
- Process Logs
- Logging and tracing

- Add the following arguments:
 - Djava.util.logging.configureByLoggingPropertiesFile=true
 - Dagencyport.env=some_string
 - Dsolr.solr.home=path_to_solr_home



Configuring JNDI

Refer to the following screen shot to configure JNDI:

The screenshot shows the JNDI Data sources configuration page. It lists two data sources: DefaultDatasource and agencyportal. The agencyportal data source is selected and highlighted with a green border.

Name	JNDI name	Scope	Provider	Description
DefaultDatasource	DefaultDatasource	Node= test-app01portals5.Server=server1	Derby JDBC Provider	Data source for the WebSphere Default Application
agencyportal	agencyportal	Node= test-app01portals5.Server=server1	Microsoft SQL Server JDBC Driver	Data source for the Microsoft SQL Server JDBC Driver. This data source type is configurable

If you are using DB2 as a provider, you must perform some additional configuration steps. Refer to the [Additional Info for DB2 Users](#) topic for more information.

Creating a JNDI JDBC Provider and Data Source

Perform the following to create a JNDI JDBC provider and data source:

1. Create a new provider.

The screenshot shows the creation of a new JDBC provider named DB2 Universal JDBC Driver Provider. The provider is listed in the JDBC providers table.

Name	Scope	Description
DB2 Universal JDBC Driver Provider	Node= test-app01portals5.Server=server1	One-phase-commit DB2 JDBC provider that supports JDBC 3.0. One session that uses the provider supports one 1-phase commit processing, unless you use driver type 2 with the application server for z/OS. If you use the application server for z/OS, driver

2. Add a data source.

The screenshot shows the 'Data sources' configuration page in the WebSphere Application Server V8 interface. The left sidebar lists various management categories like Welcome, Guided Activities, Servers, Applications, Services, Resources, Resource Adapters, Asynchronous beans, Cache Instances, Mail, URL, Resource Environment, Security, Environment, and System administration. Under the 'Resources' section, 'JDBC' is expanded, and 'JDBC providers' is selected, revealing 'Data sources' (which is highlighted with a green box). The main panel displays the 'Data sources' table with two entries: 'DefaultDatasource' and 'agencyportal'. The 'agencyportal' row is also highlighted with a green box. Below the table is a 'Test connection' button. The right side of the screen shows 'General Properties' and 'Additional Properties' sections, and a 'Related Items' section containing a link to 'JAAS - J2C authentication data' (which is also highlighted with a green box).

Select	Name	JNDI name	Scope	Provider	Description	Category
	DefaultDatasource	DefaultDatasource	Node>test-app01Node06Cell>nodes>test-app01portal5>server>server1	Derby JDBC Provider	Data source for the WebSphere Default Application	
	agencyportal	agencyportal	Node>test-app01portal5>server>server1	Microsoft SQL Server JDBC Driver	Data source for the Microsoft SQL Server JDBC Driver. This	

General Properties

- Scope**: cells>test-app01Node06Cell>nodes>test-app01portal5>server>server1
- Provider**: Microsoft SQL Server JDBC Driver
- Name**: **agencyportal**
- JNDI name**: agencyportal
- Use this data source in container managed persistence (CMP)

Description: Data source for the Microsoft SQL Server JDBC Driver. This data source type is configurable in version 6.1.0.15 and later nodes.

Additional Properties

- Connection pool properties
- WebSphere Application Server data source properties
- Custom properties

Related Items

- JAAS - J2C authentication data

Data sources > agencyportal > JAAS - J2C authentication data

Specifies a list of user identities and passwords for Java(TM) 2 connector security to use.

Prefix new alias names with the node name of the cell (for compatibility with earlier releases)

Apply

Preferences

Select	Alias	User ID	Description
<input type="checkbox"/>	test-app01portal5/agencyportal5	pos	
<input type="checkbox"/>	test-app01portal5/portal5	portal5	

Total 2

Data sources > agencyportal > JAAS - J2C authentication data > New...

Specifies a list of user identities and passwords for Java(TM) 2 connector security to use.

General Properties

* Alias:

* User ID:

* Password:

Description:

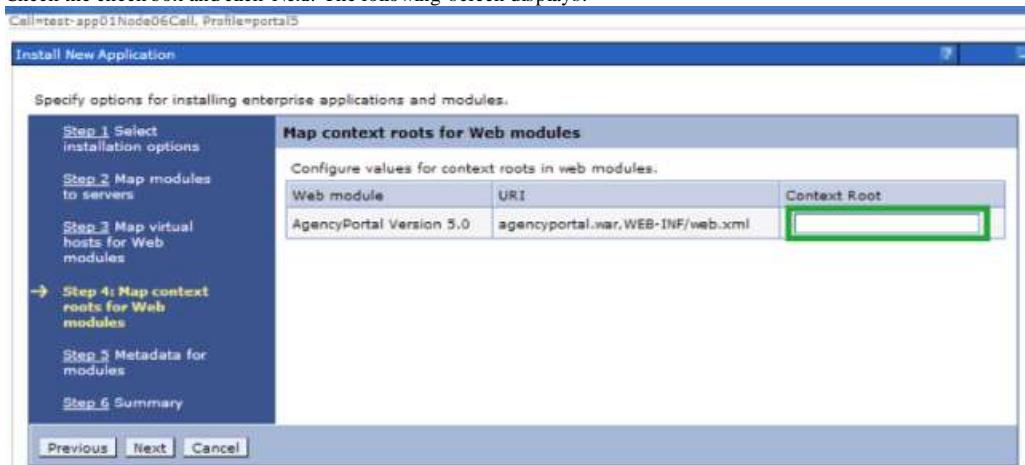
Buttons: Apply | OK | Reset | Cancel

Deploying the WAR

Perform the following to deploy the WAR:

1. Log in to the WAS Admin console.
2. Navigate to Applications > New Application > New Enterprise Application.
3. Select the WAR file to deploy.
4. Click Next.
5. Wait while the WAR file is uploaded to the server (this can take a couple of minutes).
6. Select Fast Path and click Next.
7. Check the “Precompile JavaServer Pages files” field.
8. Change agencyportal_war to “agencyportal” under the Application Name.
9. Click Next.
10. Check the check box and click Next.
11. Check the check box and click Next.

12. Check the check box and click Next. The following screen displays.



13. Enter "agencyportal" into the Context Root.
 14. Click Next.
 15. Click Finish and wait until deployment.
 16. Click "Save (directly to the master configuration").

Note

You may not receive any notification that the deployment was successful. Before you start the application, make sure you configure your [application](#) and [module](#) level classloader settings

Application Level Class Loader Settings

The following screen shots describe the settings needed for the application level class loader:

The top screenshot shows the 'Enterprise Applications' list:

- Start, Stop, Uninstall, Update, Rollout Update, Remove File, Export, Export SOD, Export File
- Select: Name: agencyportal.war
- Application Status:
- Resources: DefaultApplication, agencyportal, agencyportal.war, jobs, users
- Total: 5

The bottom screenshot shows the 'Configuration' page for 'agencyportal.war':

- General Properties: Name: agencyportal.war, Application reference validation: Issue warnings
- Detail Properties: Target specific application status, Startup behavior, Application binaries, **Class loading and update detection** (highlighted in red), Request dispatcher properties, JASPER provider
- Modules: Manage Modules, Metadata for modules, Display module build logs
- Web Module Properties: Session management, Context Root For Web Modules, Initialize parameters for servlets, JSP and JSTL options, Virtual hosts
- Enterprise Java Bean Properties

Configuration

General Properties

Class reloading options

Override class reloading settings for Web and EJB modules

Polling interval for updated files
 Seconds

Class loader order

Classes loaded with parent class loader first

Classes loaded with local class loader first (parent last)

Module Level Class Loader Settings

The following screen shots describe the settings needed for the module level class loader:

Welcome

- Selected Activities
- Servers
- Applications**
 - New Application
 - Application Types**
 - WebSphere enterprise applications
 - Business-level applications
 - Assets
 - Global deployment settings
- Services
- Resources
- Security
- Environment
- System administration
- Users and Groups
- Monitoring and Tuning

Enterprise Applications
 Use this page to manage installed applications. A single application can be deployed onto multiple servers.

Start Stop Install Uninstall Update Rollout Update Remove File Export Export DDL Export File

Select	Name	Application Status
<input type="checkbox"/>	DefaultApplication	
<input type="checkbox"/>	agencyportal	
<input type="checkbox"/>	agencyportal_web	
<input type="checkbox"/>	idbag	
<input type="checkbox"/>	sunx	

Total 5

Enterprise Applications > agencyportal
 Use this page to configure an enterprise application. Click the links to access pages for further configuring of the application or its modules.

Configuration

General Properties

Name
 agencyportal

Application reference validation
 Issue warnings

Detail Properties

- [Target specific application status](#)
- [Startup behavior](#)
- [Application binaries](#)
- [Class loading and update detection](#)
- [Request dispatcher properties](#)
- [JASPI provider](#)

Modules

- [Manage Modules](#)
- [Metadata for modules](#)
- [Display module build logs](#)

Metadata for modules

Web Module Properties

- [Session management](#)
- [Context Root For Web Modules](#)
- [Initialize parameters for servlets](#)
- [JSP and JSF options](#)
- [Virtual hosts](#)

Enterprise Java Bean Properties

[Enterprise Applications](#) > [agencyportal](#) > Manage Modules

Manage Modules

Specify targets such as application servers or clusters of application servers where you want to install the modules that are contained in your application. Modules can be installed on the same application server or dispersed among several application servers. Also, specify the Web servers as targets that serve as routers for requests to this application. The plug-in configuration file (`plugin-cfg.xml`) for each Web server is generated, based on the applications that are routed through.

Clusters and servers:

WebSphere:cell=test-app01Node06Cell,node=test-app01portal5,server=server1	<input type="button" value="Apply"/>
---	--------------------------------------

Select	Module	URI	Module Type	Server
<input checked="" type="checkbox"/>	AgencyPortal Version 5.0	agencyportal.war,WEB-INF/web.xml	Web Module	WebSphere:cell=test-app01Node06Cell,node=test-app01portal5,server=server1

[Enterprise Applications](#) > [agencyportal](#) > Manage Modules > agencyportal.war

Use this page to configure an instance of a deployed web module in the application. This page contains deployment-specific for a web module and session management settings.

Configuration

General Properties	Additional Properties
<ul style="list-style-type: none"> + URI agencyportal.war Alternate deployment descriptor * Starting weight 10000 * Class loader order Classes loaded with local class loader first (parent last) 	<ul style="list-style-type: none"> View Module Class Loader Custom properties Target specific application status View Deployment Descriptor Session Management

Server-Level Classloading Settings

Set the parent last classloading at the server level [via Servers > Application Servers > **server_name** > Class loading mode > Classes loaded with local class loader first (parent last)] as follows:

Guided Activities

Servers

Server Types

- WebSphere application servers
- WebSphere MQ servers
- Web servers

Applications

Services

Resources

Security

Application servers

Use this page to view a list of the application servers in your environment page to change the status of a specific application server.

Preferences

<input type="button" value=""/>	<input type="button" value=""/>
Name	Node
You can administer the following resources:	
server1	test-app01portal5
Total 1	

Runtime Configuration

General Properties

Name: server1
Node name: test-app01portals5
 Run in development mode
 Parallel start
 Start components as needed
Access to internal server classes: Allow

Server-specific Application Settings

Classloader policy: Multiple
Class loading mode: Classes loaded with parent class loader first

Container Settings

- Session management
 - SIP Container Settings
 - Web Container Settings
 - Portlet Container Settings
 - EJB Container Settings
 - Container Services
 - Business Process Services

Applications

- Installed applications

Server messaging

- Messaging engines
- Messaging engine inbound transports
- WebSphere MQ link inbound transports
- SIB service

Server Infrastructure

- Java and Process Management
 - Class loader
 - Process definition
 - Process execution
- Administration
 - Java SDKs

Communications

Apply OK Reset Cancel

Additional Info for DB2 Users

DB2 Issue

If you receive the following exception and are using a DB2 database, you may experience the DB2 issue described in this section.

```
com.agencyport.security.provider.RoleProvider getRoles Standard SQL Exception Info for exception
at level 0 - SQL State: 'null'; SQL Error Number: '0'; SQL Error Text: 'DSRA9110E: ResultSet is
closed.';
```



Setup for a DB2 database is different than other vendors.

- The issue that causes the error is that, by default, the application server configures the `resultSetHoldability` custom property with a value of 2 (`CLOSE_CURORS_AT_COMMIT`). This property causes DB2 to close its `resultSet/cursor` at transaction boundaries. Despite DB2's default `resultSetHoldability` value of 1 (`HOLD_CURSORS_OVER_COMMIT`), the application server retains its own default value of 2 to avoid breaking compatibility with previous versions of the application server.
- You can change the default if the need arises (you must change the value of 2 to 1 via the WebSphere application server console).

Refer to the following to make these changes:

A screenshot of the WebSphere Application Server Data Sources configuration interface. The left pane shows the navigation path: Data sources > agencyportal_old_db2. The main pane displays the configuration for the datasource "agencyportal_old_db2".

General Properties

- Scope: cells:test-app01Node06Cell:nodes:test-app01portal5:servers:server1
- Provider: DB2 Universal JDBC Driver Provider
- Name: agencyportal_old_db2
- JNDI name: agencyportal_old_db2

Additional Properties

- Connection pool properties
- WebSphere Application Server data source properties
- Custom properties** (highlighted)

Related Items

- JAAS - J2C authentication data

<input type="checkbox"/>	fullyMaterializeLobData	true	This setting can affect whether LOB data is fully materialized in the JDBC driver when a row is fetched, or whether LOB data is retrieved in pieces as needed. The actual behavior depends on whether the database server supports progressive streaming. Please refer to DB2 documentation for this property. The default value is true.	false
<input checked="" type="checkbox"/>	resultSetHoldability	1	Determine whether ResultSets are closed or kept open when committing a transaction. The possible values are: 1 (HOLD_CURSORS_OVER_COMMIT), 2 (CLOSE_CURSORS_AT_COMMIT).	false
<input type="checkbox"/>	currentPackageSet		This property is used in conjunction with the DB2binder - collection option which is given when the JDBC/CLI packageset is bound during installation by the DBA.	false

Note

This is a XA datasource; define a new customproperty on the datasource where `property_name = downgradeHoldCursorsUnderXA` and boolean value = true.

Tomcat

Deploying AgencyPortal under Tomcat is usually very straightforward. It should be as simple as adding a few extra dependencies, as well as configuring your JNDI datasource, SOLR_HOME location and a few other required properties.

Extra Dependencies

Since Tomcat is particularly lightweight, as far as application servers go, you'll need to add the following additional dependencies to get things up and running:

- javax.servlet:jstl:1.2
- org.apache.taglibs:taglibs-standard-impl:1.2.1
- javax.mail:mail:1.4.1

These dependencies need to be included in either your WAR or Tomcat/lib directory in addition to your JDBC driver of choice.

Note

The sample pom file provided takes care of including these three dependencies. Reference the default profile. You'll still need to take care of the JDBC driver yourself.

Additional Configuration

JNDI Configuration

Add a JNDI datasource to your Tomcat context file. In the out of the box framework properties file, the datasource name is set to agencyportal and the datasource prefix is set to java:comp/env/. The following is a sample JNDI resource for a MySQL database. Refer to Tomcat's documentation for more information.

```
<!-- MySQL Server --> <Resource name="agencyportal" auth="Container" type="javax.sql.DataSource" description="MySQL database for AgencyPortal" maxActive="100" maxIdle="30" maxWait="10000" username="dbusername" password="dbpassword" driverClassName="com.mysql.jdbc.Driver" url="jdbc:mysql://hostname:3306/mymodemportaldb" />
```

Solr Deployment

You must copy the SOLR_HOME to a location on the AppServer's filesystem that is outside of the deployment directory. There are multiple ways to register the SOLR_HOME directory to Tomcat; however, for now just use framework properties.

Set the solr_home property in framework.properties or \${username}.framework.properties to a value similar to the following:

```
solr_home=C:/servers/tomcat_root/runtime/SOLR_HOME
```

As long as the container has read and write permissions in that directory, the application should be able to read the configuration in SOLR_HOME at runtime and create the index files in that location.

Note

Refer to the [Solr Setup](#) section for more information.

Required Properties

There are some additional required properties that must be set before you can deploy the application. For various reasons, such as security, upload integration and Turnstile integration, the application needs to be aware of its own URL at runtime. The URL registers by setting a few properties:

my_portal_ap p	This is the master property containing the entire URL. The default is set to be the sum of the parts below.
-------------------	---

	<p>Example</p> <pre>my_portal_app=\${my_portal_app_protocol}://\${my_portal_app_domain}:\${my_portal_app_port}/\${APPLICATION_NAME}</pre> <p>Instead of altering this property, consider changing the different URL parts below.</p>
my_portal_ap p_protocol	Usually 'http' or 'https'
my_portal_ap p_domain	<p>The domain with any prefixes without any port number.</p> <p>Example</p> <p>developer.agencyport.com</p>
my_portal_ap p_port	Whatever port your application container will be listening to. The default for Tomcat is '8080'.
APPLICATION_NAME	The context of the application. Usually matches the WAR file name, but is configurable on most application containers. For the purposes of this setup guide, 'agencyportal' is used.

WebLogic

AgencyPortal is packaged as a JEE compliant application and maintains compatibility with WebLogic with a minimal amount of configuration.

- Some of AgencyPortal's JAR file dependencies, which are typically distributed as part of the product, are not needed since they collide with JARs provided as part of the container.
 - Do not include the following JAR artifacts in any AgencyPortal WAR or EAR file that you are about to deploy:
 - saaj
 - geronimo-stax-api_1.0_spec-1.0
 - stax-api
 - activation
 - taglibs-standard-impl
 - javax.mail

Note

Depending on the method employed to get to a WAR file, some (or all) of these JAR files may already be excluded. The sample pom files contain mutually exclusive profiles, which take care of these exclusions.

- Ensure your `database_prefix` application property is set appropriately. While some application containers bind JNDI names in the `java:comp/env/` namespace by default, WebLogic supports empty prefixes.
- WebLogic added a JAX-RS implementation in version 12.1.3. However, AgencyPortal relies on its own Jersey based JAX-RS implementation, which is typically embedded within an AgencyPortal WAR file. To instruct WebLogic's classloader to server up the embedded JAX-RS implementation when called upon by the application, the following `weblogic.xml` can be used:

```
<?xml version="1.0" encoding="UTF-8"?> <weblogic-web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-web-app/1.7/weblogic-web-app.xsd" xmlns="http://xmlns.oracle.com/weblogic/weblogic-web-app"> <container-descriptor> <prefer-application-packages> <package-name>javax.ws.rs.*</package-name> <package-name>org.glassfish.*</package-name> <package-name>org.jvnet.*</package-name> <package-name>jersey.*</package-name> <package-name>javax.inject.*</package-name> </prefer-application-packages> <show-archived-real-path-enabled>true</show-archived-real-path-enabled> </container-descriptor> <context-root>agencyportal</context-root> </weblogic-web-app>
```

Simply include this `weblogic.xml` file under the WEB-INF directory in the AgencyPortal WAR file.

There are some additional required properties that must be set before you can deploy the application. For various reasons, such as security, upload integration and Turnstile integration, the application needs to be aware of its own URL at runtime. The URL registers by setting a few properties:

my_portal_ap_p	<p>This is the master property containing the entire URL. The default is set to be the sum of the parts below.</p> <p>Example</p> <pre>my_portal_app=\${my_portal_app_protocol}://\${my_portal_app_domain}:\${my_portal_app_port}/\${APPLICATION_NAME}</pre> <p>Instead of altering this property, consider changing the different URL parts below.</p>
my_portal_ap_p_protocol	Usually 'http' or 'https'
my_portal_ap_p_domain	The domain with any prefixes without any port number. <p>Example</p> <pre>developer.agencyport.com</pre>
my_portal_ap_p_port	Whatever port your application container will be listening to. The default for Tomcat is '8080'.

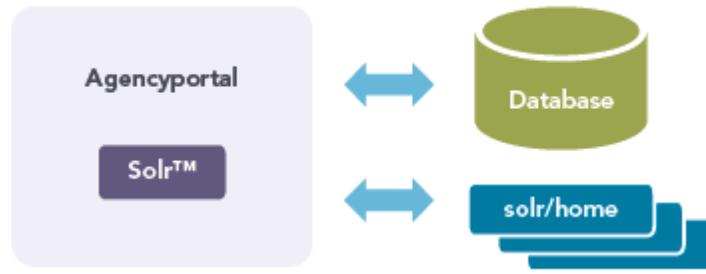
APPLICATION_NAME	The context of the application. Usually matches the WAR file name, but is configurable on most application containers. For the purposes of this setup guide, 'agencyportal' is used.
------------------	---

Solr Setup

Out of the box, AgencyPortal 5.2 uses Solr™ 4.10.4, an open source enterprise search platform from the Apache Lucene™ project, for its search and filter functionality. This allows you to search through all account and work item data in a scalable way instead of having to tie a search value to a particular data column on the database.

The following diagram illustrates the layered architecture and the various components that constitute each tier when using Solr for search and filter within AgencyPortal:

Portal Runtime



After you install and set up the major components of AgencyPortal that are outlined in the [Developer Setup](#) and [Deployment Information](#) sections, set up and configure Solr to use the search and filter feature within AgencyPortal. This section includes the following technical details for a software developer assigned with the task of setting up AgencyPortal:

- [how to set up Solr in AgencyPortal](#)
- [how to deploy Solr under WebSphere](#)
- [how to deploy Solr under WebLogic](#)

Also included is information on [Solr Cloud](#), which allows you to set up a cluster of Solr servers to distribute certain index and search functionality.

Refer to the [Solr Configuration](#) topic in the Developer Guide section for more information on customizing Solr and the components of this feature.

Setting Up Solr in AgencyPortal

The sections included in this topic detail how to set up Solr in AgencyPortal.

Using the Embedded Solr Engine

AgencyPortal contains an embedded Solr engine. Using the embedded Solr engine can simplify the WAR deployment process, for both local and clustered environments since you only need to deploy one WAR to get everything up and running. There are only three steps to get AgencyPortal's embedded Solr engine up and running for a non-clustered environment.

1. Copy the SOLR_HOME folder from the SDK zip file into a folder where you want to store the account and work item index data.

Note

SOLR_HOME contains configuration files for the Solr search engine and will, ultimately, house the search index data, too. You don't want to lose search index data when deploying later versions of an application WAR; therefore, SOLR_HOME is typically not bundled into application WAR. Instead, place the folder in a directory on the server where the application has read and write access.

2. Provide the application server where AgencyPortal is running with the location of the SOLR_HOME directory where the Solr configuration is housed. You can do this in one of the following ways:

1. You can configure the location of SOLR_HOME using framework.properties.

Example

Add the following to agencyportal's framework.properties file:

```
solr_home=/projects/portal/runtime/SOLR_HOME
```

2. Add a systemproperty named solr.solr.home with a file path that points to the SOLR_HOME directory.

Example

Add the following JVM argument:

```
-Dsolr.solr.home=/projects/portal/runtime/SOLR_HOME
```

3. Add a JNDI entry for Java: comp/env/solr/home that points to the SOLR_HOME directory.

Example

In Tomcat, you can add the following entry to context.xml:

```
<Environment name="solr/home" type="java.lang.String"  
value="/parentfolder/SOLR_HOME" override="true" />
```

The application needs to be aware of its URL so that it can know the base_solr_url to use when sending HTTP requests into its own embedded Solr engine. For flexibility purposes, base_solr_url is a discreet application property in and of itself; however, this is preconfigured to be derived from my_portal_app* set of application properties. Therefore, you only need to correctly set the following properties in the environment specific framework.properties file:

- my_portal_app_protocol=http
- my_portal_app_domain=localhost
- my_portal_app_port=9999
- APPLICATION_NAME=prestigeportal

Using an External Solr Instance

AgencyPortal does not require you to use its embedded Solr engine. In fact, pointing to a Solr instance that is running externally is seamless. There are some benefits to using an external Solr instance. First, under this approach, Solr will be running on its own dedicated JVM. Second, when it comes to allocating clustered environments, the external approach provides greater flexibility because you're no longer constrained to having a one-to-one ratio of AgencyPortal nodes to Solr nodes. Third, you get to take advantage of the full Solr admin console UI. The drawback, however, is that deploying these two applications separately is more work on the infrastructure/operations side. To get started with this approach, perform the following steps:

Note

When Solr runs outside AgencyPortal in a SolrCloud configuration, it is important to ensure that there is a load balancer sitting between the AgencyPortal instances and the Solr instances. Otherwise, each AgencyPortal application will be configured to hit a specific Solr instance and the application will cease to be fault tolerant.

1. Download Solr from the Apache website and rename the solr-4.10.4.war file to solr.war. The important parts of the zip file are:
 1. dist/solr-4.10.4.war
 2. example/lib/ext/*
 3. example/cloud-scripts/skcli.sh (or .bat)

Note

AgencyPortal currently supports version 4.10.4. Future versions of AgencyPortal may require different versions of Solr.

2. When you first download the Solr distribution, one of the first things you must do as part of the initial deployment is set up the logging framework. Refer to [this](#) article for more information.
3. To enable the Solr admin console for AgencyPortal's indexes, add an admin request handler to the solrconfig.xml files located in SOLR_HOME/account/conf/ and SOLR_HOME/workList/conf/. The following XML snippet can be added alongside the other requestHandler elements located within the solrconfig.xml files:

```
<requestHandler name="/admin/" class="solr.admin.AdminHandlers" />
```
4. Provide the application server where the Solr instance is going to run with the location of the SOLR_HOME directory where the Solr configuration is housed. You can do this in one of the following ways:
 1. Add a systemproperty named solr.solr.home with a file path that points to the SOLR_HOME directory.

Example

Add the following JVM argument:

-Dsolr.solr.home=/projects/portal/runtime/SOLR_HOME

2. Add a JNDI entry for Java: comp/env/solr/home that points to the SOLR_HOME directory.

Example

In Tomcat, you can add the following entry to context.xml:

```
<Environment name="solr/home" type="java.lang.String" value="/parentfolder/SOLR_HOME" override="true" />
```

5. A Solr instance needs to be aware of its own host, hostPort and hostContext. Provide this information to Solr in one of the following ways:
 1. Adjust the environment specific entries in the solr.xml file located in the SOLR_HOME folder. For reference, the following table describes some of the configuration options within the solr.xml file.

Node	Description	Default Value
Host	The hostname Solr uses to access cores.	localhost
hostPort	The port Solr uses to access cores.	8080
hostContext	The servlet context path.	agencyportal/solr
zkClientTimeout	A timeout for connection to a ZooKeeper server. It is used with SolrCloud.	1500ms
genericCoreNodeNames	If TRUE, node names are not based on the address of the node, but on a generic name that identifies the core. When a different machine takes over serving the core, things will be much easier to understand.	true

2. You can change the default values highlighted in the following example:

```
<solr> <solrcloud> <str name="host">${host:localhost}</str> <int name="hostPort">${portal.port:8080}</int> <str name="hostContext">${hostContext:agencyportal/solr}</str> <int name="zkClientTimeout">${zkClientTimeout:15000}</int> <bool
```

- ```

name="genericCoreNodeNames">>${genericCoreNodeNames:true}</bool> </solrcloud> <shardHandlerFactory
name="shardHandlerFactory" class="HttpShardHandlerFactory"><int name="socketTimeout">${socketTimeout:0}</int>
<int name="connTimeout">${connTimeout:0}</int> </shardHandlerFactory></solr>
```
3. Use JVM arguments, such as the following, to override the default values applied by solr.xml:
    - -Dhost=dev-app10
    - -Dhost=dev-app10
    - -Dhost=dev-app10
  6. Deploy solr.war and verify by browsing to the Solr admin console (an HTTP GET request to http://yourhost:yourport/yoursolrcontext/ should be directed to the admin page).

#### Note

If the concerned environment needs a cluster of Solr instances, then you need to set up a SolrCloud instance. Refer to the [Solr Cloud](#) topic for more information.

To disable AgencyPortal's embedded Solr instance and point it to an external Solr instance, the following changes are all that is required:

1. Set or override the base\_solr\_url property in framework.properties to point to the external Solr instance's URL.
2. Modify web.xml and remove the following two servlet filters and their filter-mappings:
  - SolrFilter
  - SolrRequestFilter
3. Optionally, if you're using the Maven build system, you can modify your project's pom.xml file and add an exclusion for the solr-core dependency to the apwebapp dependency. This will prevent the inclusion of Solr JARs in the WAR file, which would otherwise be lying dormant. The following is an example of how you can add an exclusion:

```

<dependency>
 <groupId>com.agencyport.sdk</groupId>
 <artifactId>apwebapp</artifactId>
 <version>${agencyport.sdk.version}</version>
 <exclusions>
 <exclusion>
 <groupId>org.apache.solr</groupId>
 <artifactId>solr-core</artifactId>
 </exclusion>
 </exclusions>
</dependency>
```

#### Note

Removing the SolrFilter and SolrRequestFilter from web.xml and removing the Solr dependencies from pom.xml may not be practical if you plan on running Solr externally for some environments while running it in embedded mode for some others (e.g., as part of your local config).

# Deploying Solr Under WebSphere

You may need to perform the following modification for AgencyPortal embedded Solr engine, or a standalone instance of Solr, to run in WebSphere. By default, ServletFilters are not initialized on during server startup under WebSphere (<http://www-01.ibm.com/support/docview.wss?uid=swg1PM62909>). This is an issue because the SolrDispatchFilter does a lot of work during the init phase. To work around this issue, there is a customproperty that can be added to force WebSphere to invoke ServletFilter.init.

1. Navigate to the custom properties menu in WebSphere admin console: Servers > Application Servers> server\_name > Web Container Settings > Web Container > Custom Properties.
2. If not already present, add the following new property:  
`com.ibm.ws.webcontainer.invokeFilterInitAtStartup=true`.

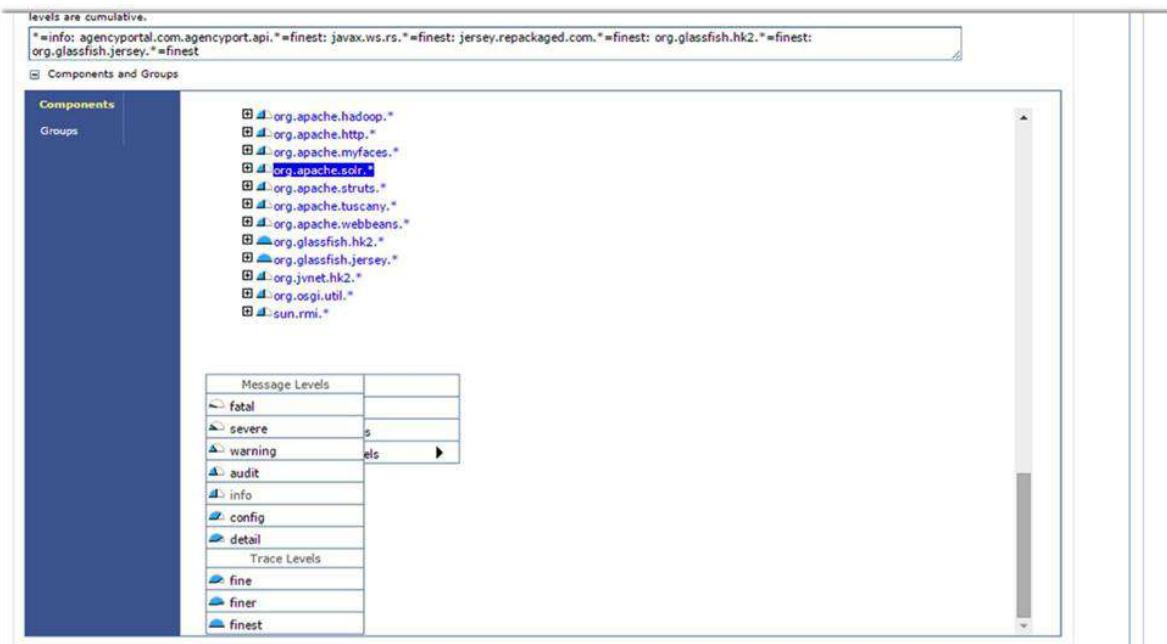


A secret key is used to validate the authenticity of any inbound Solr request when AgencyPortal's embedded Solr engine is used. The cryptography library provided as part of IBM's JDK mandates the usage of an alternate keystore file within our application. The SDK's apbase.jar is packaged with this alternate keystore file so all you have to do is point to it within application properties [e.g., framework.properties or acsi.properties: `solr_keystore_filename=(asResource)keystore/ibmsolr.key`].

## Tuning Solr Logging Under WebSphere

To adjust the logging level for Solr under WebSphere, you can navigate to the following dialog in the administration console:

Go to Troubleshooting > Logs and trace > <server> > Diagnostic Trace > Change log detail levels > Components and Groups > org.apache.solr.\*



# Deploying Solr Under WebLogic

Perform the following if you want to provide the location of SOLR\_HOME using an environment variable under WebLogic:

1. Add the following to the web.xml file:

```
<env-entry><env-entry-name>solr/home</env-entry-name> <env-entry-type>java.lang.String</env-entry-type> <env-entry-value>c:\project\SOLR_HOME</env-entry-value> </env-entry>
```

The above hardcoded value can be replaced by the application server deployment configuration file: plan.xml. plan.xml is produced when you define the configuration using the admin console.

2. Perform the following in the plan.xml file:

- Define variable definition:

```
<variable-definition> <variable> <name>SOLR_HOME</name> <value>c:\project\SOLR_HOME</value> </variable> </variable-definition>
```

- Define a module override to replace the value from web.xml:

```
<module-descriptor external="false"> <root-element>web-app</root-element> <uri>WEB-INF/web.xml</uri> <variable-assignment> <name>SOLR_HOME</name> <!-- this links to the variable-definition name --> <xpath>/web-app/env-entry[env-entry-name="solr/home"]</env-entry-value></xpath> </variable-assignment> </module-descriptor>
```

# Solr Cloud

Solr includes the ability to set up a cluster of Solr servers that combines fault tolerance and high availability. This feature is called SolrCloud.

SolrCloud provides the capabilities to distribute index and search functionality to support the following features:

- central configuration for the entire cluster
- automatic load balancing and fail-over for queries
- ZooKeeper integration for cluster coordination and configuration.

Let's start by defining some of the terms used in the SolrCloud world:

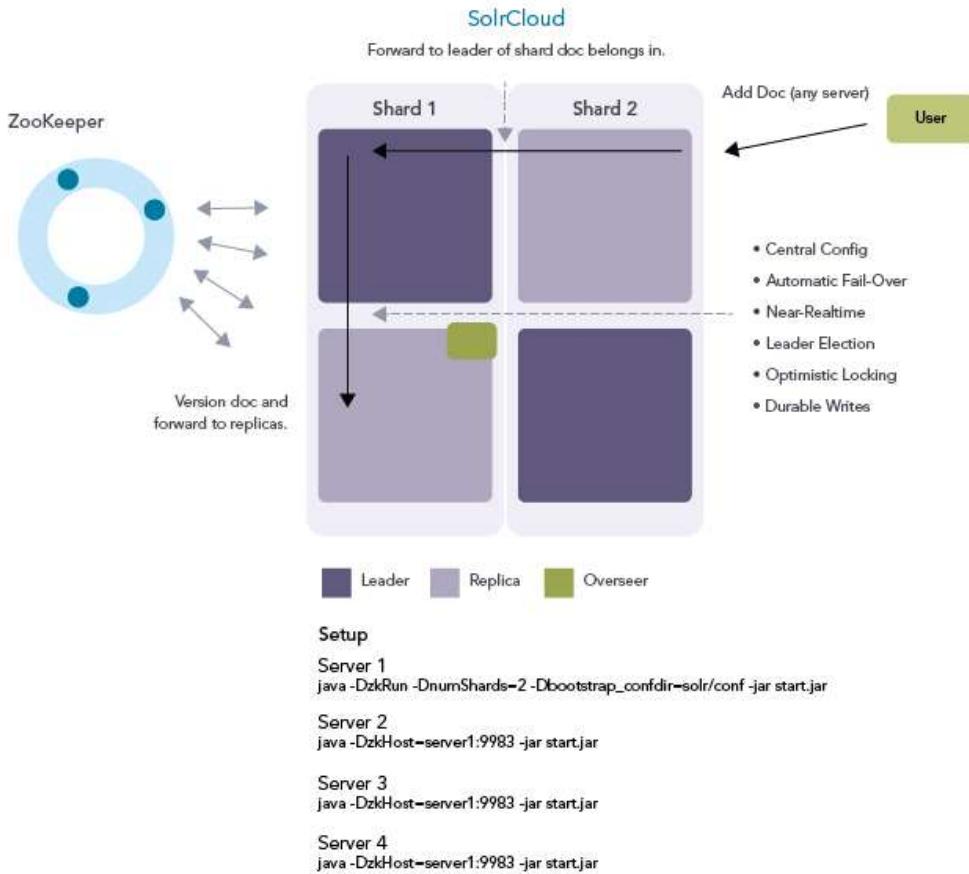
- **Collection** - A collection is a complete logical index in a SolrCloud cluster. All of the nodes in a SolrCloud cluster, the configuration sets and the data in the index all make up the collection.
- **Shard** - Every sole node in the SolrCloud cluster has its own data store located in the file system. A shard is a partition of the data in the collection. If SolrCloud is configured with one shard, then each node in the cluster will have a synchronized copy of the entire index stored locally. This means that any single node can fully handle a search request since each node has all of the data. However, since individual nodes have 100% of the data under their purview and are responsible for handling search requests (and other operations) on their own, then performance will suffer if the index is very large. This is why having multiple shards becomes beneficial. When there are two shards, the data is divided in two and shared amongst the nodes. The nodes allocated to shard 1 will manage one half of the data and the nodes allocated to shard 2 will manage the other half.

## Example

If your Solr cluster has two shards and two nodes, that means that roughly half of the indexed documents are stored in one node and the other half is stored in the other node. Since each node in the cluster only manages half of the data, a search request can be handled by a node in shard 1 and a node in shard 2 concurrently. It varies from case to case; however, it is unlikely that an AgencyPortal implementation will need more than one shard unless it gets into the multi-million document range. Refer to the [SolrCloud Cluster Composition](#) section for more information on this topic.

- **Leader** - Each shard is made up of one or more nodes. One of these nodes is elected as a leader. Leader nodes maintain the canonical copy of the data for the shard and therefore have a few extra responsibilities, such as distributing index/update requests to the other nodes in the shard to keep them in sync.
- **Replica** - If a shard has more than one node, then the nodes that aren't the leader are said to be "replica nodes". Replica nodes maintain a synchronized copy of the leader node's content. They can handle search requests on their own, but index/update requests get forwarded to the leader node, which ensures that all of its replica nodes are kept in sync. Leader nodes are replicas, too!
- **Core** - A core is a related set of data upon which we want to search and its corresponding configuration files that define the data schema and other options. AgencyPortal's out of the box Solr configuration defines two cores: one for account data, named "account", and one for work item data, named "worklist".
- **ZooKeeper** - ZooKeeper is an Apache product that SolrCloud uses to coordinate state and configuration across the individual Solr nodes in the cluster.
- **ZooKeeper Ensemble** - A cluster of ZooKeeper servers working in tandem in a cluster to provide fault tolerance.

SolrCloud is flexible and distributes search and indexing without a master node to allocate nodes, shards and replicas. Instead, Solr uses ZooKeeper to manage these locations, depending on configuration files and schemas. Documents can be sent to any Solr node and SolrCloud will use ZooKeeper to figure it out.



SolrCloud is designed to provide a highly available, fault tolerant environment that can index your data for searching. It's a system in which data is organized into multiple pieces, or shards, that can be housed on multiple machines with replicas providing redundancy for both scalability and fault tolerance, and a ZooKeeper server that helps manage the overall structure so that both index and search requests can be routed properly.

This section explains SolrCloud and its inner workings in detail; however, before you dive in, it's best to have an idea of what it is you're trying to accomplish. This page provides a simple tutorial that describes how SolrCloud works within AgencyPortal on a practical level and how to take advantage of its capabilities. We'll use simple examples of configuring SolrCloud on a single machine, which is obviously not a real production environment that includes several servers or virtual machines. In a real production environment, you'll also use the real machine names instead of "localhost," which we use in these examples.

For more detailed information on SolrCloud, refer to the [SolrCloud](#) Apache site.

## SolrCloud and AgencyPortal

AgencyPortal ships with an embedded Solr engine. The embedded Solr engine also contains an embedded ZooKeeper engine. As such, you have all of the components necessary to set up a working SolrCloud with just one agencyportal WAR file. However, deploying this kind of configuration in a production environment is discouraged. For the highest availability, we recommend you [deploy a ZooKeeper ensemble](#) independently. You should also consider deploying Solr independently if you want to take advantage of SolrCloud's admin UI. Refer to the "*Using an External Solr Instance*" section in the [Setting Up Solr in AgencyPortal](#) topic for more information.

To set up a SolrCloud cluster using the AgencyPortal's embedded Solr engine, you only need to configure a few application properties and some system properties (JVM arguments) on each node.

### Register the Location of SOLR\_HOME

Each node must have its own SOLR\_HOME configuration. The location of the SOLR\_HOME directory where the Solr configuration is housed must be provided. You can do this in one of the following ways:

1. You can configure the location of SOLR\_HOME by setting the "solr\_home" application property on each node's environment specific framework.properties file.

### Example

Add the following application property:

```
solr_home=/projects/portal/runtime/SOLR_HOME
```

2. Add a systemproperty named solr.solr.home with a file path that points to the SOLR\_HOME directory.

### Example

Add the following JVM environment:

```
-Dsolr.solr.home=/projects/portal/runtime/SOLR_HOME
```

3. Add a JNDI entry for Java (comp/env/solr/home) that points to the SOLR\_HOME directory.

### Example

In Tomcat, you can add the following entry to context.xml:

```
<Environment name="solr/home" type="java.lang.String" value="/parentfolder/SOLR_HOME"
override="true" />
```

## Make Solr Nodes Aware of their URLs

Each Solr node must be aware of its own URL so that it can tell ZooKeeper how other nodes can contact it. There are application properties that you can set on each node's environment specific framework.properties file or you can use system properties (JVM arguments) or modify the solr.xml file within each node's SOLR\_HOME. Either way, the URL is specified in three parts:

	<b>Application Property</b>	<b>System Property (JVM Argument)</b>
Hostname	<code>solr_cloud.node.host=dev-app10</code>	<code>-Dhost=dev-app10</code>
Port	<code>solr_cloud.node.hostPort=8080</code>	<code>-Dport=9999</code>
Host Context	<code>solr_cloud.node.hostContext=prestiegeportal/solr</code>	<code>-DhostContext=prestiegeportal/solr</code>

### Note

As a convenience, agencyportal prefills a built-in systemproperty named `application.hostname`, which you can use to dynamically assign the hostname for each environment in application properties. For example, adding `solr.cloud.node.host=${application.hostname}` to framework.properties should appropriately configure the hostname for any and all nodes.

If you don't want to use application properties or systemproperties to configure each Solr node's URL information, you can alternatively adjust the environment specific entries in each node's corresponding solr.xml file located in it's SOLR\_HOME directory. For reference, the following table describes some of the configuration options within the solr.xml file.

<b>Node</b>	<b>Description</b>	<b>Default Value</b>
Host	The hostname Solr uses to access cores.	localhost
hostPort	The port Solr uses to access cores.	8080
hostContext	The servlet context path.	agencyportal/solr
zkClientTimeout	A timeout for connection to a ZooKeeper server. It is used with SolrCloud.	15000ms

Node	Description	Default Value
genericCoreNodeNames	If TRUE, node names are not based on the address of the node, but on a generic name that identifies the core. When a different machine takes over serving that core, things will be much easier to understand.	true

You can change the default values highlighted in the following example:

#### Example

```
<solr>
 <solrcloud>
 <str name="host">${host:localhost}</str>
 <int name="hostPort">${portal.port:8080}</int>
 <str name="hostContext">${hostContext:agencyportal/solr}</str>
 <int name="zkClientTimeout">${zkClientTimeout:15000}</int>
 <bool
name="genericCoreNodeNames">${genericCoreNodeNames:true}</bool>
 </solrcloud>
 <shardHandlerFactory name="shardHandlerFactory"
 class="HttpShardHandlerFactory">
 <int name="socketTimeout">${socketTimeout:0}</int>
 <int name="connTimeout">${connTimeout:0}</int>
 </shardHandlerFactory>
</solr>
```

#### Bootstrap Configuration for Multiple SolrCores

AgencyPortal ships configuration to use two SolrCores: "account" and "worklist." Many of the SolrCloud examples on the [Apache Solr documentation site](#) have only one core and illustrate how to get Solr to upload the core's configuration to ZooKeeper with the `bootstrap_confdir` systemproperty (where `bootstrap_confdir` is set to the directory of the one and only core within the node's `SOLR_HOME`). Since we have multiple cores, you only need to set the alternate `bootstrap_conf` systemproperty to true. This tells Solr to read `SOLR_HOME/solr.xml` and upload the configuration set for each core that it finds. You only need to upload Solr configuration to ZooKeeper once (the first time you start up the first Solr node in the SolrCloud).

`-Dbootstrap_conf=true`

#### Specify the Number of Shards

Similar to the `bootstrap_conf` argument described above, the `numShards` argument needs to be provided once, during the initialization of the first Solr node, the first time you start up the SolrCloud. This argument tells Solr the number of logical partitions you plan on splitting the index into (for larger indexes, splitting the search index across multiple shards can improve search response times).

`-DnumShards=2`

#### Set Up a ZooKeeper Ensemble

Download the appropriate version of ZooKeeper from Apache (Solr 4.10.4 depends on ZooKeeper 3.4.6). Extensive documentation of ZooKeeper is outside the scope of our documentation, but the following is some high-level information:

#### Note

When running Windows, the 'start' and 'zoo.cfg' parameters should be omitted. On Windows, ZooKeeper only works with the CFG file when it has the default name of `zoo.cfg`.

- ZooKeeper uses a quorum voting system. For an ensemble of ZooKeeper servers to handle any request, a majority of ZooKeeper servers must be active and available to vote. Refer to the [Determining the Number of ZooKeeper Servers](#) section below for more information.

- The ZooKeeper distribution comes with a zoo\_samples.cfg file under its conf directory. You can take this file as a starting point and create your own zoo.cfg file.
- Each ZooKeeper server needs to listen on three ports. One port is used for listening to external clients, such as Solr; the other two are used for internal ZooKeeper server to server communication (leader election and voting). These are specified for each server in the zoo.cfg file.
- Each ZooKeeper server needs to be aware of every other ZooKeeper server in the ensemble. This information is also provided within each server's zoo.cfg file.
- Each ZooKeeper server will need a location on the filesystem to store its runtime data. These "dataDir" will need to be seeded with a text file named "myid," which should contain the numerical ID (integer value between 1 and 25) that you have assigned for each of the servers in your ZooKeeper ensemble.

For a ZooKeeper ensemble that has three servers, this is what one of the zoo.cfg files might look like:

```
tickTime=2000
dataDir=/var/lib/zookeeperdata/1
clientPort=2181
initLimit=5
syncLimit=2
autopurge.purgeInterval=8
server.1=10.20.30.40:2888:3888
server.2=10.20.30.40:2889:3889
server.3=10.20.30.40:2890:3890
```

The parameters are as follows:

- **tickTime** - Part of what ZooKeeper does is to determine which servers are up and running at any given time, and the minimum session time out is defined as two "ticks". The tickTime parameter specifies, in milliseconds, how long each tick should be.
- **dataDir** - This is the directory in which ZooKeeper will store data about the cluster. This directory should start out empty.
- **clientPort** - This is the port on which Solr will access ZooKeeper. This is one of the three ports mentioned above that each ZooKeeper server will need to listen on.
- **initLimit** - Amount of time, in ticks, to allow followers to connect and sync to a leader. In this case, you have 5 ticks, each of which is 2000 milliseconds long, so the server will wait as long as 10 seconds to connect and sync with the leader.
- **syncLimit** - Amount of time, in ticks, to allow followers to sync with ZooKeeper. If followers fall too far behind a leader, they will be dropped.
- **autopurge.purgeInterval** - This is the interval (in hours) on which the autopurge task will trigger. Based on the above example, the task will fire every eight hours and will delete all but the most recent three log files (the default value for autopurge.snapRetainCount is 3) and corresponding snapshots.
- **server.X** - These are the IDs and locations of all servers in the ensemble (including the current server) and the ports upon which they communicate with each other. The server ID must additionally be stored in the <dataDir>/myid file and be located in the dataDir of each ZooKeeper instance. The ID identifies each server, so in the case of this first instance, you would create the file /var/lib/zookeeperdata/1/myid with the content "1". In the example above, you'll also notice the two port number notation (delimited by the semi-colon). We mentioned earlier that each ZooKeeper server listens on three ports, the clientPort and two other ports for internal ZooKeeper server to server communication. This is how those two other port numbers are specified.

You can use the bin/zkServer.sh script to start a ZooKeeper server on Unix systems. You can pass the name of the config file to use, such as the following:

```
sh bin/zkServer.sh start zoo.cfg
```

To start multiple ZooKeeper servers on the same machine, using the same ZooKeeper distribution, simply run the above command multiple times while passing in a different config file name each time.

## Note

Parts of the above documentation are based on documentation provided by Apache Solr, which is available [here](#). You can also refer to [ZooKeeper's Getting Started](#) documentation for more information.

## Integrate Solr with ZooKeeper

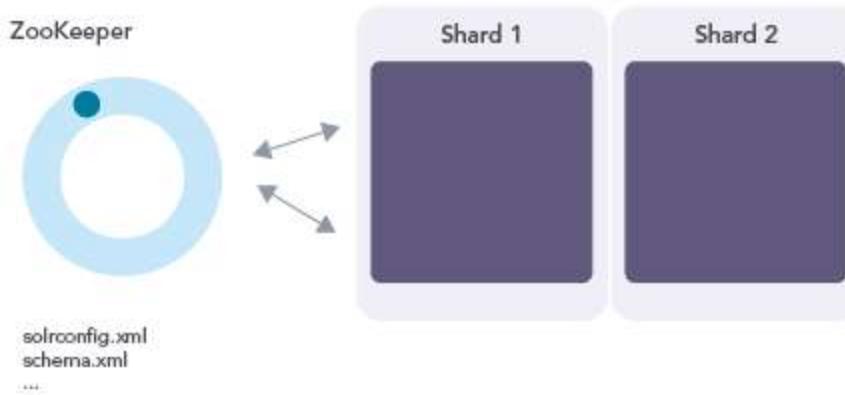
Each Solr node needs to be aware of the hostname and client port number for each ZooKeeper node in the ZooKeeper ensemble. This information can be provided to Solr by using the `solr_cloud.zookeeperHosts` application property or the `zkHost` system property. Either way, this is a comma-separated list of addresses for each node in your ZooKeeper ensemble, which is specified as follows:

- as an application property:  
`solr_cloud.zookeeperHosts=10.20.30.40:2181,10.20.30.40:2182,10.20.30.40:2183`
- or, as a JVM startup argument:  
`-DzkHost=10.20.30.40:2181,10.20.30.40:2182,10.20.30.40:2183`

Review the following examples to complete the configuration:

### Example 1

Simple Two-Shard Cluster on the Same Machine



### Creating a Cluster with Multiple Shards

Perform the following to create a cluster with multiple shards:

1. Set up and start the ZooKeeper server.
  1. Download and extract the ZooKeeper distribution (`zookeeper.tar.gz`) from the Apache site.
  2. Create a file named `zoo.cfg` under the `conf` directory with the following contents:

```
tickTime=2000 dataDir=/var/lib/zookeeperdata clientPort=2181 initLimit=5 syncLimit=2 server.1=localhost:2888:3888
```

  3. Create a directory that matches the `dataDir` property specified in `zoo.cfg`.
  4. Create a file named `myid` in the `dataDir` directory created above. The file should only contain the character "1". This value corresponds to the "server.1" ID listed in the `zoo.cfg` file.
  5. Start the ZooKeeper server by navigating to the ZooKeeper directory and running the following Unix command:

```
sh bin/zkServer.sh start zoo.cfg
```

2. On each node's `solr.xml`, change the values for `host`, `hostPort`, `hostContext`, or, alternatively, use the application properties or JVM arguments described immediately before this example.

```
<solr>
 <solrcloud>
 <str name="host">${host:10.21.39.153}</str>
 <int name="hostPort">${portal.port:8080}</int>
 <str
 name="hostContext">${hostContext:agencyportal/solr}</str>
 <int name="zkClientTimeout">${zkClientTimeout:15000}</int>
 <bool
 name="genericCoreNodeNames">${genericCoreNodeNames:true}</bool>
 </solrcloud>
```

```

<shardHandlerFactory name="shardHandlerFactory"
 class="HttpShardHandlerFactory">
 <int name="socketTimeout">${socketTimeout:0}</int>
 <int name="connTimeout">${connTimeout:0}</int>
</shardHandlerFactory>
</solr>

```

- Start the first node (in our case, any one portal in the cluster - the order of the parameters should be maintained):

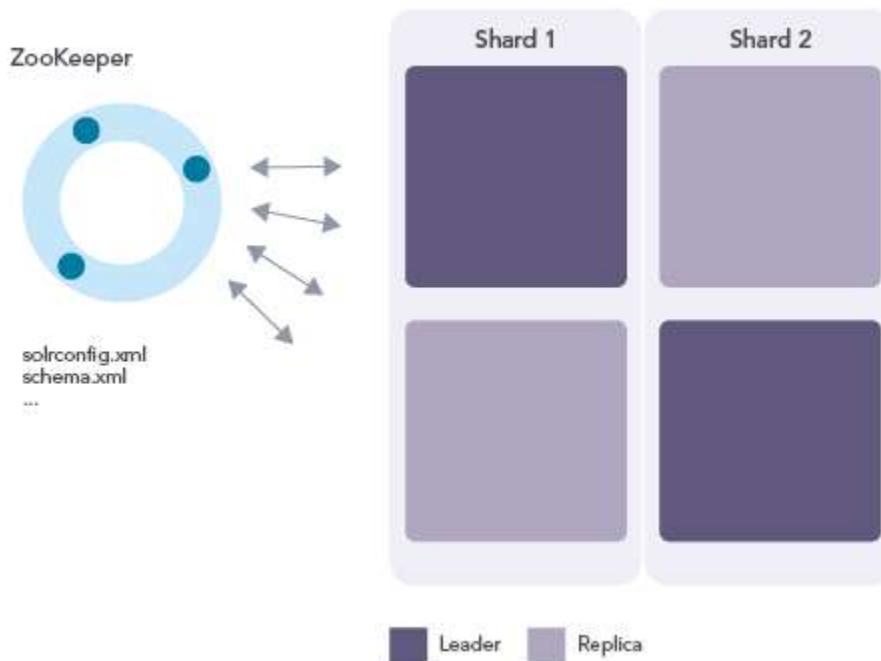
```
-Dsolr.solr.home=/data/application/overseernode/SOLR_HOME -Dbootstrap_conf=true -DnumShards=2
```

- DzkHost:** tells Solr how to communicate with ZooKeeper (use comma separation to specify multiple hosts).
- Dbootstrap\_conf=true:** since we don't have a config in ZooKeeper yet, this parameter causes the local configuration directory for each SolrCore under SOLR\_HOME to be uploaded to ZooKeeper. This is only necessary on the first Solr node to be started when there is no data in ZooKeeper or when newly updated schema info needs to be uploaded.
- DnumShards=2:** the number of logical partitions we plan on splitting the index into.

- Start any remaining shard nodes and point them to the running ZooKeeper.

## Example 2

Simple Two Shard Cluster with Shard Replicas and Three ZooKeeper Servers in an Ensemble



The problem with the previous example is that there is only one ZooKeeper server that contains the state of the cluster. If that ZooKeeper server crashes, distributed queries will still work since the Solr server remembers the state of the cluster last reported by ZooKeeper. The problem is that no new servers or clients will be able to discover the cluster state and no changes to the cluster state will be possible.

Running multiple ZooKeeper servers in concert (a ZooKeeper ensemble) allows for high availability of the ZooKeeper server. Every ZooKeeper server needs to know about every other ZooKeeper server in the ensemble and a majority of servers are needed to provide service.

A ZooKeeper ensemble of three servers allows any one server to fail with the remaining two constituting a majority to continue providing service. Five ZooKeeper servers are needed to allow for the failure of up to two servers at a time.

A cluster like the one listed above might be composed as follows:

Three ZooKeeper Servers

host1:

```
hostname: zk1
zoo.cfg entries:
dataDir=/data/zookeeperdata/
clientPort=2181
server.1=zk1:2888:3888
server.2=zk2:2888:3888
server.3=zk3:2888:3888
```

host2:

```
hostname: zk2
zoo.cfg entries:
dataDir=/data/zookeeperdata/
clientPort=2181
server.1=zk1:2888:3888
server.2=zk2:2888:3888
server.3=zk3:2888:3888
```

host3:

```
hostname: zk3
zoo.cfg entries:
dataDir=/data/zookeeperdata/
clientPort=2181
server.1=zk1:2888:3888
server.2=zk2:2888:3888
server.3=zk3:2888:3888
```

Four AgencyPortal/Solr nodes that start with the following JVM arguments

node1:

```
-Dsolr.solr.home=/data/application/node1/SOLR_HOME -Dbootstrap_conf=true -
DzkHost=zk1:2181,zk2:2181,zk3:2181 -DnumShards=2

node2:
-Dsolr.solr.home=/data/application/node2/SOLR_HOME -DzkHost=zk1:2181,zk2:2181,zk3:2181

node3:
-Dsolr.solr.home=/data/application/node3/SOLR_HOME -DzkHost=zk1:2181,zk2:2181,zk3:2181

node4:
-Dsolr.solr.home=/data/application/node4/SOLR_HOME -DzkHost=zk1:2181,zk2:2181,zk3:2181
```

## Note

Notice how the `bootstrap_conf` argument is only specified for the first node. You should only specify this argument the first time Solr is registering the configuration data with Solr or when the configuration data has changed and needs to be re-uploaded to ZooKeeper. This means that after we bring up the remainder of the cluster, we should bring the first node back down, remove its `bootstrap_conf` argument and bring it back up again. This ensures that we only upload this configuration data to ZooKeeper when necessary.

## SolrCloud Cluster Composition

When trying to determine the optimal SolrCloud cluster composition (i.e., the number of shards, replicas, ZooKeeper servers), there are many factors that will vary on a case by case basis. Unfortunately, there is no direct way to answer this. The following information describes some of the factors to consider.

### Determining the Number of Shards and Replicas

First, we must describe the differences between a shard and a replica.

- A shard is a partition of the data in the index. If your Solr cluster has two shards and two nodes, that means that roughly half of the indexed documents are stored in one node and the other half is stored in the other node. SolrCloud does the work of figuring out which documents to send to which shards when an index request comes in. Incoming search requests will be routed to both shards concurrently, regardless of which shard received the original request and, since each shard is only searching half the index, overall search response times should be faster.
- A replica maintains an exact copy of all of the data allocated to the shard. If you have two nodes and one shard, then this shard is said to have two replicas. One node will be nominated as the leader of the shard (a leader is a replica node with a few extra responsibilities), the other will act as a replica. If you have six nodes and two shards, then each shard would have three replicas. SolrCloud does the work of ensuring that a replica is kept in sync with its leader.

If either of these concepts seem unclear, refer back to the SolrCloud information above.

The amount of shards you should have depends on the amount of documents in the index (in our case, documents are made of work item and account records). An individual Solr node with 6000000 documents under its purview may take significantly longer to respond to a search request than a Solr node with 3000000 documents.

If you're in a situation where search requests are taking too long, having the index split across more than one shard will reduce the amount of time it takes one node in one shard to handle a search request. Of course, going from a one shard cloud to a two shard cloud doesn't exactly halve the response time.

There is additional overhead that Solr must undertake to route the search request to each applicable shard and then process, combine and weight the results for the client response. This may lead you to infer that having too few shards will lead to an oversized index in each shard, which will slow down the servicing of search and index requests. Conversely, having too many smaller shards will lead to too much overhead (i.e., there are diminishing marginal returns for each additional shard).

You must also consider that each shard must have a leader node and at least one replica node if high availability, fault tolerance and automatic backup and recovery is desired. Replicas also provide additional capacity for concurrent processing of requests since each replica can handle a search request on behalf of a shard, so that factor is worth considering too. All of this means that if you've decided that some replication is desirable, then resources (i.e., the total number of nodes that the infrastructure can support) will also become a limiting factor when determining the ratio of shards to replicas.

If you're planning on running a two node cluster, you will be limited to one shard, which is comprised of a leader and replica node. A four node cluster could support a maximum of two shards with a leader and replica node each, but you might be better served by a one shard, three replica configuration. Of course, once you step into the realm of having eight or more nodes in a cluster, your options become less limited.

### Note

The release of Solr 4.3 brought with it "shard splitting" functionality (which was made more robust in versions 4.3.1 and 4.4). Prior to shard splitting, there was no way to change the number of shards in a SolrCloud cluster without tearing down the cluster and re-indexing all documents under the new configuration. This meant that users would have to guesstimate the number of shards they might conceivably need in the future (while, naturally, erring on the side of too many). The shard splitting API allows you to have SolrCloud split an existing shard that has grown too large into two shards.

## Determining the Number of ZooKeeper Servers

The first thing to consider is that maintaining just one (or two) ZooKeeper servers would become a single point of failure for your entire SolrCloud infrastructure. ZooKeeper implements a [Paxos](#) algorithm to avoid split brain scenarios. A quorum of active ZooKeeper servers is needed to service any ZooKeeper requests (i.e., unless half the number of your ZooKeeper services plus one are up, the entire ZooKeeper ensemble will be down). This means that you would need at least three ZooKeeper servers in your ZooKeeper ensemble at a minimum.

The next thing to consider is that Solr does not heavily use ZooKeeper. Yes, ZooKeeper is an integral part of the SolrCloud architecture, but Solr is not transferring reams of data to and from ZooKeeper. Instead, it transfers only very minimal cluster state information. For this reason, it's unlikely that you'll see any significant performance gains from scaling up the number of ZooKeeper servers in your ensemble.

## Backup and Recovery

SolrCloud makes backup and recovery seamless. As previously described, a shard's replica nodes maintain an exact copy of the documents in a shard, and replicas keep in sync with each other and the leader of the shard automatically by SolrCloud. As long as each shard has some replica nodes in place, then SolrCloud has what it needs to automatically invoke its backup and recovery routine in the event of a node's failure. SolrCloud's backup and recovery routine works as follows:

- If a replica node goes down (i.e., it stops checking in with ZooKeeper), other nodes in the SolrCloud will automatically stop routing requests to it in favor of the other replicas in the shard (including the shard's leader node). This applies to requests that have been initially sent to Solr nodes that are active and are slated to be internally rerouted by SolrCloud's internal load balancer. For requests coming in from the outside, an external load balancer is needed to prevent this request traffic from going to the downed server. When the downed server is eventually brought back online, it will automatically recover by re-replicating itself against the leader. This replication will incorporate any newly indexed documents that have been added to the shard since the node went down.
- If the leader node of a shard goes down, SolrCloud will automatically use ZooKeeper to elect a new leader (choosing from the other replicas in the shard). The newly elected leader of the shard goes through a recovery process of replicating itself against all of the other replicas in the shard to ensure that it has the very latest data. You may ask yourself, '*if replica nodes maintain exact copy of the shard, then why would the newly elected leader need to go through a recovery process?*'. The answer is that the newly elected leader may not have the latest data in this case. If the leader node went down during the process of notifying the other replicas about a newly indexed document, then the replica that was elected may not have the latest document.
- If the ZooKeeper ensemble can't reach quorum, then the entire SolrCloud goes into read only mode. This alleviates potential split-brain scenarios and sync concerns. If AgencyPortal fails to write to Solr when indexing new accounts and work items or updating existing accounts and work items, it will not continue with making the equivalent change to the database to keep the index and the database in sync. In this case, users will see errors on screen so it is important to promptly bring a ZooKeeper quorum back online.
- If an entire shard goes down, then all search requests will fail (unless the `shards.tolerant` flag is set to true, but this is something that AgencyPortal does not support). Index and update requests will fail if they are applicable to the downed shard. This means that the integrity of the index will be maintained once a node from the downed shard comes back online.

In the event of a truly catastrophic event where all Solr nodes are lost, you can always fall back to re-establishing a clean SolrCloud and re-index the entire index against the database.

## Re-Indexing (Batch Index)

AgencyPortal does not use Solr as its "system of record." Although there is a growing movement to [use Solr as a NoSQL](#) database, AgencyPortal still relies on a traditional RDBMS to maintain the canonical copies of work item and account data. Keeping Solr's index in sync with the database is something that AgencyPortal does out of the box. For example, changes for new or updated work items will be persisted to Solr before being written to the database if the Solr update fails and processing does not continue. However, there are some scenarios when a full re-index of the Solr documents against the database is required.

A re-index or "Batch Index" involves first deleting all documents under Solr and subsequently recreating them all individually by querying the system of record that, in our case, is AgencyPortal's database. The most common scenario that necessitates a full re-index of Solr is a change to `schema.xml`, which causes the documents in the Solr index to no longer match the format specified in the Solr schema.

Other scenarios include data integrity issues where the data in Solr no longer matches the data in the system of record (AgencyPortal actively defends against these integrity issues, but they are, nonetheless, still possible). Or, an initial roll-out of Solr to an existing production database (e.g., a production deploy of an AgencyPortal 4.x to 5.x migration).

AgencyPortal exposes an extensible Solr Batch Index UI for invoking a batch re-index from a live AgencyPortal instance. This can be used as part of a production release process where a re-index is required. After the production WAR is deployed, an administrator would invoke the batch index process from the UI, wait for it to complete and then confirm that Solr search functionality is working. All of this would need to take place before real user traffic is routed to the newly deployed production environment.

For information about using and configuring AgencyPortal's built-in Solr batch index processor, refer to the [Solr Batch Indexing](#) section.

# User Interface

This section provides information on the AgencyPortal user interface, which includes a list of supported browsers and information on the many different design aspects of the application. This includes the following components:

- [Browsers](#)
- [Styles](#)
- [Buttons](#)
- [Modals](#)

The targeted audience of this section is a front-end developer assigned the task of customizing AgencyPortal.

# Browser Support

AgencyPortal 5.2 supports the following browsers:

- Internet Explorer (9+)
- Chrome (current, -1)
- Firefox (current, -1)
- Safari (5.1+)
- Opera (current, -1)
- IOS Safari (6+)
- Android Chrome (4+)
- Microsoft Edge

Supported browsers are determined during browser validation testing. We typically use the most recent browser versions plus, at a minimum, the prior two versions of that browser. We support later browser versions as long as the application interaction maintains its stability and is supported by product development.

## Browser Testing

With each release of AgencyPortal, we extensively test the application throughout multiple browsers to ensure that not only functionality works as expected, but also that all styles and other UI components display correctly.

We perform our cross-browser testing using a variety of resources, such as:

- Executing automated scripts
- Manually testing the app using a variety of laptops and desktops
- Using remote machines set up with varying operating systems and browsers
- Setting up remote productivity tools, such as Spoon.net, to perform localized testing via a QA URL across multiple browser versions without having to install and build a large variety of environments
- Testing with bad data

Automated script testing and using remote productivity tools to test across multiple browsers are mostly used to test application functionality. We make sure data entry, TVR and endorsement/renewal functionality work correctly, as well as searching and sorting, Work Item Assistant (including attachments) functionality and any PDF attachment and creation. We use a more manual testing approach to the later to ensure that everything appears as expected. For each browser version tested, we conduct a series of manual tests to validate that the following display, operate or function as expected:

- Field lengths
- Data entry input functions
- Field validations
- Standard rules processing
- Field population
- All pages are accessible via all normal means
- All pages are accessible at varying points in the workflow
- Links
- Font size consistency
- Browser behavior consistency [older browser versions remain consistent within the family of browsers up through the most recently released version (e.g. FF 27 through FF 25)]
- End of processing functions (i.e. continue buttons, save and exit selections, etc.)
- Page display size and format
- PDF display
- File attachments
- Search functionality
- Work item management functions (e.g., copy, assign, claim, open, etc.)
- Workflow processing functions

## Style and UI Testing

The browsers that AgencyPortal supports are leveraged from Bootstrap. Even with the responsive intelligence of our Bootstrap implementation, some browsers still interpret AgencyPortal differently. The addition of new functionality and third-party libraries creates new markup and styles. Sometimes those new features conflict with previously tested code. Therefore, every new version of AgencyPortal is tested thoroughly.

We expect visual elements to appear the same in our supported browsers and platforms. Those elements include fonts, colors, page layout and form actions and display on page templates. Page templates include the logon, Home, My Accounts, My Work, My Reports, data entry and pop-

up modal. Components on these pages consist of data entry, roster tables, summary pages and the list and card view of the account and work item work lists.

Our test is performed on Mac OS X, Windows and mobile devices to ensure that all styles and UI components are consistent.

- We perform Windows testing using Microsoft's Remote Desktop client. Agencyport has a variety of remote desktops dedicated just for browser testing.
- We use an iPhone and iPad for mobile device testing using both Windows desktop browsers (IE, Firefox, Chrome) and iOS devices (mobile Safari).

During style and UI testing we look at various pages throughout the application. This process is informal and manual, essentially navigating through the app to (a) make sure new features and bug fixes have no unintentional damage and (b) look for any new problems that might have been missed in the previous release.

The following pages are reviewed using our supported browsers/platforms:

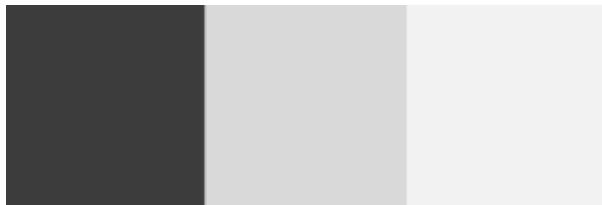
- Logon
  - log into the application as an agent, underwriter and administrator
  - check form position and background photo size/position
- Home
  - check visual layout of main navigation, cards UI, tables and footers
  - hover over cards to show hidden buttons
  - click on a full application/quick quote button to create a new work item with a data entry page
- Data Entry Page (new work item)
  - check visual layout of Submission Navigation, Data Entry Form, Work Item Menu and Work Item Assistant
  - fill out the form
  - check drop-downs, option lists and calendars
  - use deliberately wrong dates to trigger error messages
  - check roster pages
  - check policy summary page
- Submission Navigation
  - check clickable, current, disabled and problem states of links
- Work Item Assistant
  - add comments
  - add documents (with button and drag and drop)
  - add documents without declaring type to trigger error message
  - edit document names
- Work Item History ("Timeline")
  - click Event List links (left side)
  - click Timeline Tabs
- Data Entry Page (existing work item)
  - edit, add and remove items from rosters
  - check Work Item Menu: History, Upload, Corrections and My Defaults
- My Accounts
  - test card and list views
  - create a new account (personal and commercial)
  - view/create/edit work, locations and people
- My Work
  - test card and list views
  - double click on a selected work item
  - check hidden action buttons
  - check bottom pagination widget
  - move a work item to an account
- My Reports
  - check the Report Console (data buttons under each chart)
  - click on components (pie slices, chart handles, etc)
- Get Started
  - make and process and upload
  - check uploaded work item

# Style Guide

This topic describes the out of the box styles used in AgencyPortal 5.2.

## Overall Color Palette

AgencyPortal has a monochromatic color scheme. This was intentional since colors are used selectively in the application for specific meanings, such as work item statuses.



Indigo-blue (#416daf) is the prominent color for links, primary buttons and highlighting specific information. This color complements the status colors while still being more prominent.



## Example

The default dominant blue color is used to highlight which page the user is on in the data entry process. Pages not filled out yet display in gray.

## Navigation Bar

The main navigation bar is global on all pages. The font, color, etc are completely customizable. Out of the box, the navigation bar is dark gray so that it is always the most dominant element on a page.



Icons display next to each page name.

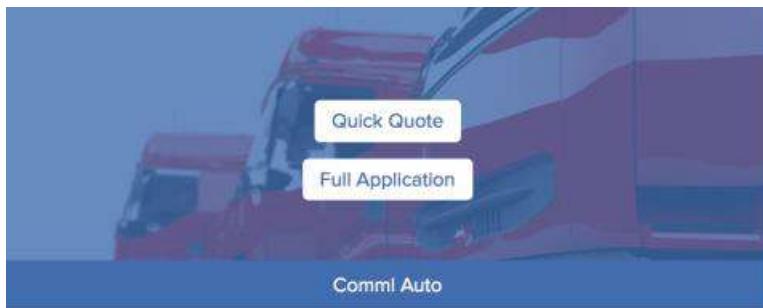
## Note

The Home page is assigned the dashboard icon since a house icon is already in use by the homeowners LOB.

## Home Page

### LOB Thumbnails

Each of the LOB thumbnails on the home page is customizable to include images.



## Work Items

Work items display, by default, in card format.

### Status Colors

The following are assigned to each status type to distinguish the status of the work item:

- Referred - Yellow (#f0ad4e)
- Declined - Red (#c0392b)
- In-Progress - Teal (#16a085)
- Approved - Green (#27ae60)
- Bound - Dark-Blue (#2c3e50)

Certain statuses, such as Declined, inherit an associated color, such as red. This eliminates the use of red for any other status. This allows for each status to be visually different.

### Example

Approved is green since it is the opposite of red on the color wheel.

Other statuses that may be harder to define, such as bound, fall in place after the others are identified. This helps to create a spectrum of the statuses and allows you to decide which colors to associate with each status.

The use of color helps give the work items their own identities, which allows the user to easily decipher when there are multiple items to choose from. Status colors are also helpful to refer back to work items in certain dashboard items, such as quote boxes.

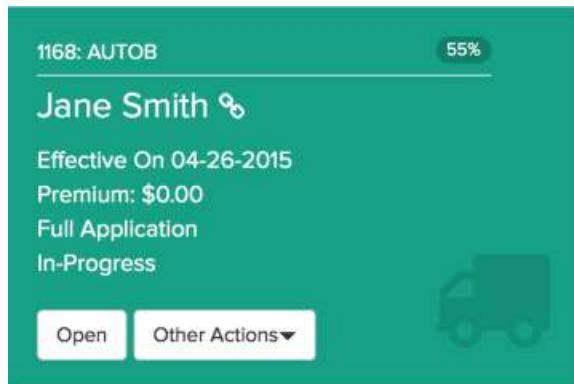


## LOB Icons

The icon that displays on a work item denotes the LOB. We chose from the Font Awesome gallery of icons to describe the LOB most accurately.

- Commercial Auto (truck icon)
- Farmowners (leaf icon)
- Homeowners (house icon)
- Personal auto (road icon)
- Worker's comp (wrench icon)

Hovering over a work item fills the card with the status color.



## Note

More colors and icons may be needed depending on whether there are more LOBs/statuses.

## Data Entry

The following field elements are available to add to a page as you see fit:

- Basic input fields (required and non-required fields)
  - Text areas for descriptions
  - Drop-down menus for questions with multiple option

* Account Type	<input checked="" type="radio"/> Commercial <input type="radio"/> Personal
* First Name	<input type="text"/>
* Last Name	<input type="text"/>
SSN	<input type="text"/>
Home Phone Number	<input type="text"/>
Work Phone Number	<input type="text"/>
Mobile Phone Number	<input type="text"/>
Primary Email Address	<input type="text"/>

- Check boxes or radio buttons (e.g., certain LOBs have a questionnaire with a list of yes/no questions)

### Eligibility Questionnaire

	Yes	No
* 1. Does applicant own, operate or lease aircraft/watercraft?	<input checked="" type="radio"/>	<input type="radio"/>
* 2. Do/have past, present, or discontinued operations involve(d) storing, treating, discharging, applying, disposing, or transporting of hazardous materials? (e.g. landfills, wastes, fuel tanks, etc.)	<input checked="" type="radio"/>	<input type="radio"/>
* 3. Any work performed underground or above 15 feet?	<input checked="" type="radio"/>	<input type="radio"/>
* 4. Any work performed on barges, vessels, docks, bridge over water?	<input checked="" type="radio"/>	<input type="radio"/>
* 5. Do you own or operate any other business?	<input checked="" type="radio"/>	<input type="radio"/>
* 6. Are sub-contractors used? (If Yes, give % of work subcontracted)	<input checked="" type="radio"/>	<input type="radio"/>
* 7. Any work sublet without certificates of insurance?	<input checked="" type="radio"/>	<input type="radio"/>

- Dates use the calendar widget. The selected date displays in blue; today's date displays in grey.



- Icons (display with data entry fields)
  - Help (question mark)
  - Replace information with previous info from upload (arrow)
  - Invalid field value (no/stop icon)

Credit Bureau Name	Select One	<input type="button" value=""/>
Credit Bureau ID	111222333	
Tax ID Type	FEIN	
Tax ID	01-1421111	

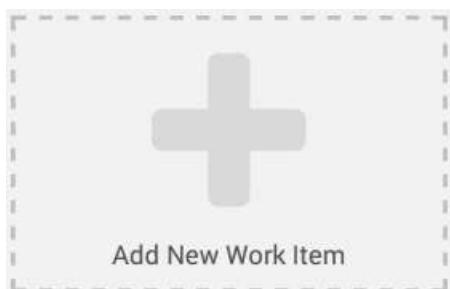
- Rosters
  - Roster uploaded from file/added roster (option to Edit or Delete) - normal text color
  - Deleted roster (option to undo) - grey
  - Edited roster (option to edit or delete) - italicized
  - Unrecoverable roster (no option) - grey

Category	Actions
Roster Uploaded From File	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
Deleted Roster	<input type="button" value="Undo"/>
Edited Roster	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
Roster Needs Editing	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
Unrecoverable Roster	

It's particularly important for rosters to have an established hierarchy so each roster type is clear to the user. A roster that needs editing should be the most dominant in appearance while a deleted or unrecoverable roster is least dominant.

## Add Buttons

On the My Work and My Accounts pages, an add button with a + displays to add a new work item or account. The size and shape of the icon mimics the card size of work item and account cards, but the dotted line and gray color enable them to be less visually important. This allows users to know the option is always there, but does not distract their main focus. The main focus of the page should be the work item or account cards.





## Account Cards

Account cards are similar to work item cards. There are two types of account cards: Personal and Business.

The account cards are much simpler in design, with either the person icon to indicate a personal account or building icon to indicate a business/commercial account.

Two side-by-side account cards. The left card is for "JAN Jeans, Inc" and is labeled "Commercial". It includes the address "1188 64 Altantic Ave Boston MA 02210" and a building icon. The right card is for "Jane Doe" and is labeled "Personal". It includes the address "1186 22 Summer St Boston MA 02210" and a person icon.

Since accounts do not have statuses, the active blue color displays when you click or hover on the card.

A single account card for "Jane Doe". The card has a blue background and contains the following information: "Personal", "1186", "22 Summer St Boston MA 02210", and a person icon. At the bottom are two buttons: "Open" and "Delete".

## Account Information

The account information section displays the basic information for that account. There are also options to edit, merge or delete the account. The style of the Account Information component mimics the style of modals; however, unlike a modal, it is in a fixed position on the page.

An "Account Information" modal window. It contains the following sections: "Shane McCarthy" (Person, Account # 6186), "Contact Information" (Sleepy Street Side Street, Boston, MA 12312-3123, (123) 456-7890 (Work), (123) 456-7890 (Mobile), www.jens.com), and "Created" (2014-07-14 by test test) and "Last Updated" (2014-07-14 by Zemman Hte).

# Button Guide

Buttons are used throughout the AgencyPortal application to create an interactive, user-friendly experience that is consistent from device to device. Buttons help distinguish the product as an application that offers more functionality compared to a website.

## Important

To be consistent throughout the application, create a standardized set of rules for all of the various uses of buttons within the application.

## Button Types

<b>Primary</b>	<p>Primary function buttons are used to highlight a specific action a user will choose more than 50% of the time. It is also a useful tool for guiding a user to select a specific action within a scenario. The primary action buttons are highlighted in a dominant color. Out of the box, these buttons are in the same blue used actively within the application.</p> <p><b>Example</b></p> <p>Yes is a primary action since users usually click Yes to confirm an action that initiates a process.</p>
<b>Secondary</b>	<p>Secondary function buttons are used to create a distinction from the primary action. Secondary action buttons display in the default Bootstrap gray style. This allows the right contrast compared to the primary function buttons, which need to be visually more significant.</p> <p><b>Example</b></p> <p>No or Cancel is used for an action initiated by a user. After the action is initiated by a user, it's unlikely they would change their mind; therefore, No or Cancel are visually less important.</p> <p>Other examples are Undo, Delete, etc.</p>

## Buttons and User Permissions

When a user does not have permissions for certain actions, only secondary button actions display. Primary button actions that are off limits to a user are simply not shown.

If custom actions are wanted (e.g., Save and add another), a decision must be made whether a button type is appropriate over the other.

## Special Cases (Secondary Buttons Only)

In specific cases, there may only be secondary buttons.

## Example

A page where there are many sets of similar buttons. It can be visually overwhelming for there to be more than a few primary buttons, so only secondary buttons display.

Out of the box, this scenario occurs on the My Work card, which displays in groups.

## Single Click vs Double Click

All buttons should be activated on with a single click within the application. Double clicking is typically reserved for opening applications/folders on a desktop and not in a web interface.

## Alignment

Buttons always align on the left side of a page, modal or card. This helps maintain consistency in a range of components that vary in size. Keeping all buttons on the left helps ensure the application is intuitive and easy to use. It also allows users to expect where to find buttons.

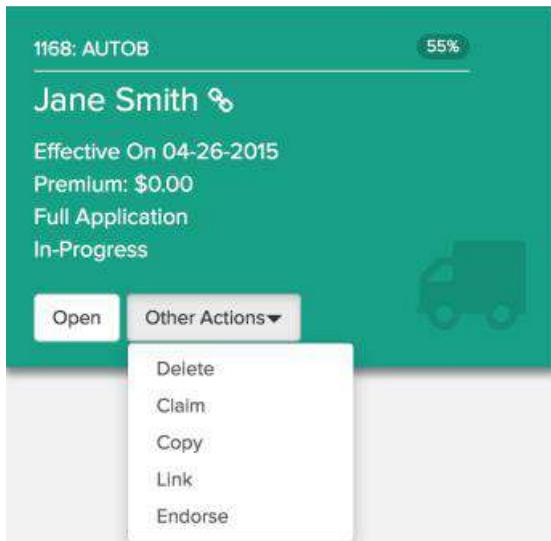
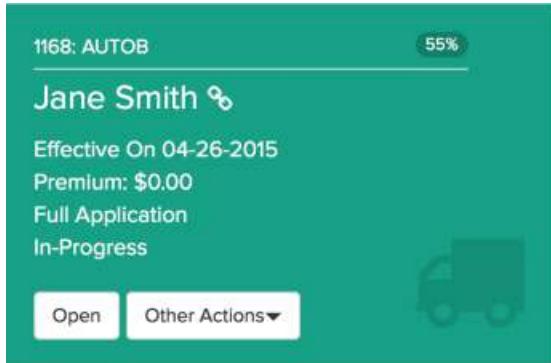
## Drop-down Buttons

Drop-down buttons are a useful tool when more than two actions are needed in a limited space.

### Example

On the work item card, there is already text, iconography and color in one area, and there is only so much room to have buttons represented individually before the space becomes too crowded.

Drop-down buttons also make it easier for responsive view. As the screen gets smaller, there is less space to display buttons. Consolidating buttons into a single drop-down button makes it easier to display valuable information within the limited space.



## Button Grouping

You can group buttons in groups of two or three (any more than three small actions are better represented in a drop-down button).

Location Information

Location

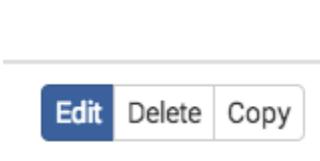
Address City, MA 11111

Continue Add More Actions ▾

Button groups can include drop-down buttons or other traditional buttons. A way to differentiate between buttons in a group is to use the primary/secondary button language.

### Example

In this example of a three button group, Edit is the primary function:



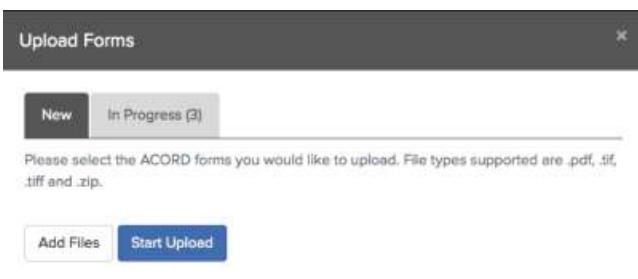
### Using Icon vs. Button

Certain actions are better described using pictures or icons instead of represented with a button. Icons provide an easy way to an action's meaning across to the user while saving time and space.

Icons should not be placed within a button.

### Button Sequencing

The sequencing of the upload buttons is different. The primary action, Start Upload, is on the right because you need to add files before you can upload. For a first-time user, Add Files is the most important of the two; however, for any other upload, Start Upload is the most important action, which is why it displays as the primary action.



### Buttons vs. Links

There are specific advantages to using buttons and links. In certain cases, it makes sense to use links rather than buttons. For example, the toolbar within a work item.

In the above sample, it makes more sense to use links since buttons take up too much space and links are traditionally used more.

### Important

Keep in mind what a user is already familiar with in terms of a user interface. It is better to use buttons to call out specific attention to a link, such as the Get Started button in the main navigation menu. You want the user to know, at any time within the application, they can start an upload. The prominence of a button in a menu of links helps highlight this specific, very important action.

### Buttons with Input Fields

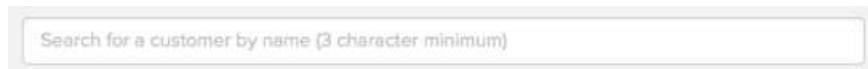
Certain buttons are grouped with input fields.

#### Example

The search button. The input is where the user types in their search and the button initiates the action.



An exception to the rule is the search on the My Work page. No button is needed because the search is initialized once the user starts typing in the search bar.



Another type of button is the calendar button on data entry pages. When you click the Calendar icon, an option to select the date displays. The date the user selected displays in the input field.

Grouping a button with an input field allows the user to easily view the information without having to click for a pop-up or other type of module. This functionality limits the number of clicks the user has to make, as well as helps reinforce the action taking place, such as the search example above.

# Modal Guide

Modals allow us to display information and actions in a neatly, organized way and allows the user to click off at any time. Modals are less invalid than normal pop-ups and can include various element types, such as search bars and tables. Modals are used throughout AgencyPortal in many different ways, such as informational or to initiate a workflow.

## Important

It's important to create a standard for all modals, regardless of the type, to ensure design consistency.

## General Modal Standards

Every modal should have:

- a header (header text size should be the largest of all text in the modal)
- an X to close the modal
- buttons with left alignment

## Action Modals

Action modals are used to allow the user to conveniently perform certain tasks, pertaining to accounts or work items, on the same page.



### Action-Specific Criteria

The following is action-specific criteria related to action modals:

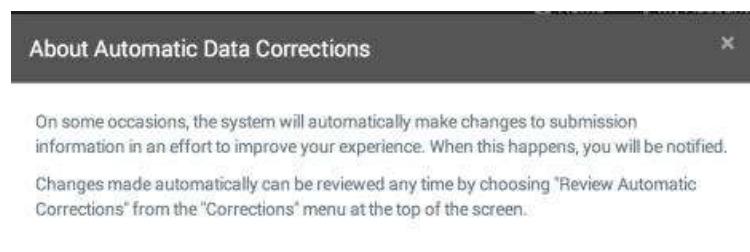
- **Assign** - No buttons on initial page. User searches for username, selects the username, then confirms with Yes or No. Confirmation displays, indicating work item is assigned.
- **Move** - No buttons on initial page. User searches for account, then confirms with Yes or No. Confirmation message displays, indicating work item is moved.
- **Copy** - Confirm with Yes or No. Next page displays message indicating new work item created.
- **Delete** - Confirm with Yes or No for both accounts and work items.
- **Merge** - No buttons on initial page. User searches for account, selects the account, then confirms with Yes or No.

## Other Modals

Modals are also used to display information, especially information that doesn't necessarily need to be visible at all times, such as "help" information.

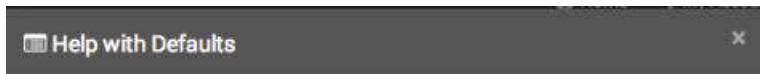
### Example

Help with specific aspects of work items, such as defaults or corrections. The user can access this helpful information at any time within the work item.



## What not to do

The following is an example of what not to do with modals:



If you frequently write similar types of policies, using defaults can save you considerable data entry time. Once you have got the submission where you want it (i.e. you've added some coverages, answered underwriting questions, etc) use the "save to defaults" action in the drop down at the bottom of the page to save a copy (and some keystrokes!) for later.

Once you've created a set of defaults, next time you start a new submission from scratch, choose "Apply to this Submission" from the "My Defaults" menu at the top of the screen. Blank fields in the new submission will be filled in with your chosen defaults.

When applying default values to submissions in which fields have already been filled out, not all fields will be replaced.

The defaults you setup belong only to you, other users can not see or change them.

---

[Close](#)

The X and Close button are redundant. Both options to close the modal are not needed. Use the X to simplify the design.

# Developer Guide

This section includes additional reference information and examples for configuring and maintaining the components of your AgencyPortal project. Browse through the following topics for detailed information on each topic:

- [Data](#) - The Data topic includes data related information needed to configure your project. This includes Agencyport XML Engine (AXE), data model, data schema, faster XML parsing, XML schema, advanced data management (ACORD simplified programming) and change management.
- [Product Definition](#) - The Product Definition topic includes detailed information related to the components used to bring your project to life. This includes how to define transaction definition files (TDF)/page libraries, option lists, dynamic transaction rendering (DTR)/behaviors, field validations, transaction validations, connectors, XARC rules, workflow management, views, product data base and product definition memory conservation.
- [Work Items](#) - The Work Items topic provides detailed information on the various components that are included in work items, such as work item management, account management, Solr configuration, Work Item Assistant, Timeline and work list.
- [Security](#) - The Security topic includes a variety of detailed information pertaining to security within the AgencyPortal application. This includes sub-topics on effective security programming, ACSI, application layer, security adaptor layer, security provider, security reference implementation, implementing single sign on and additional security considerations.
- [Request Processing](#) - The Request Processing topic includes information on request processing, which provides the ability to support a standard sequence of controlling tasks to display and process different types of pages to make up transactions and customize those tasks that are required to display and process pages.
- [Client-Side Development](#) - The Client-Side Development topic contains information on client-side development, which includes altering HTML markup, styling the user interface and client-side programming using JavaScript.
- [PDF Generation](#) - The PDF Generation topic provides information on the PDF generation feature within AgencyPortal.
- [Integration](#) - The Integration topic lists products integrated with AgencyPortal 5.2, such as Connect5, Turnstile and policy integration kits, and information pertaining to the features and components of those products.
- [Application Properties](#) - The Application Properties topic includes information about application properties, which includes a list of available application properties and data schemas and data schema resources.
- [Upload Writer Programming](#) - The Under Writer Programming topic contains information on features available for the upload writer infrastructure and how to use these features.
- [Logging Configuration](#) - The Logging Configuration topic includes information on the various logging configuration features available within AgencyPortal, including available logging packages and classes and configuration files.
- [Debug Console](#) - The Debug Console topic includes detailed information on the AgencyPortal debug console components and how to create custom debug panels.
- [Bootservice Provider](#) - The Bootservice Provider topic provides information on bootservice provider, the components that make up the subsystem and how to configure it.
- [Event Audit](#) - The Event Audit topic information on event logging within AgencyPortal. This includes the event audit data model, event display and event auditing API.
- [Localization](#) - The Localization topic provides details on how to change the application's language and how to control display text.
- [Performance Logging](#) - The Performance Logging topic includes information on performance testing within the AgencyPortal application, as well as information on how to install, set up and use JMeter for performance testing.
- [Reports](#) - The Reports topic provides information on the different types of reports available to display within the application and how to configure them for your project.

The targeted audience for this section is a software developer assigned with the task of customizing AgencyPortal. The order of the topics that display in this section are arranged by importance.

# Data

The data access management component is the layer of software in the SDK that is responsible for the following:

- On the display side, it transfers data stored as an XML document image (APDataCollection instance) to HTML page entities as dictated by the TDF.
  - Applies default values, as configured in the TDF, to the fields on pages the user has not yet visited.
  - Reads those data values from the XML document image using APDataCollection.getFieldValue() for regular fields and transforms the data value over to an HTML field for pages the user has already visited.
  - Engages the roster reader whose responsibility it is to paint the list portion on an SDK roster page.
  - Interacts with the change management subsystem to support the UIR data collection interface associated with upload or policy change oriented work items.
- On the process side, it manages the update to the XML document based on the action posted to the server. The actions typically are one of the following:
  - Insert or update the values posted to the server over HTTP to the XML document. This is in reaction to the user committing a data entry or roster add/edit page.
    - For each regular field, APDataCollection.setFieldValue() is called to transfer the field's value into the XML document.
    - For view-based fields, the process side processing associated with those views are engaged and their purpose is to transform and transfer data values streamed in from the browser into the XML document.
    - Engages the empty aggregate removal feature to eliminate empty elements.
    - Referential integrity processors can be registered with each LOB whose job is to retain the data integrity of the work item's XML document.
  - Delete XML element(s) as they apply to roster entries. This engages the delete logic associated with special field helpers and issues an APDataCollection.deleteElement() method call to delete the main XML aggregate in context for that roster.
  - Recover XML element(s) as it applies to roster entries. This supports the ability to recover data for work items from the original document. This also engages special recover logic on any special field helpers registered with that page.
- Negotiates with the database persistence layer to carry out read, insert, update and delete database operations to manage the XML document associated with the current work item. Each XML document has an associated work item identifier (aka, work item ID), which uniquely identifies each XML document, including a host of other supporting data elements.

The main SDK Java class that controls most of the functionality described above is the `com.agencyport.data.DataManager` Java class.

# Agencyport XML Engine (AXE)

The Agencyport XML Engine (AXE) is a term used to describe the components of the framework developed to manage the complexities of working with ACORD XML as a data format. The Java class that supports AXE is the APDataCollection class and, in many discussion circles, APDataCollection and AXE are used synonymously.

The term APDataCollection is used in two contexts:

- The application data, which refers to a JDOM based XML object (`org.jdom2.Document`) that is wrapped by the APDataCollection class.
- The set of mechanisms associated with creating and managing the XML state of the document using its API.

APDataCollection can be used to manage any XML document with a predefined XML schema available at runtime.

The purpose of APDataCollection is to:

- provide a comprehensive component that supports the insertion, update, deletion and retrieval of data stored in XML format.
- simplify the creation of XML instances by bundling the creation of element nodes (XML hierarchy) and the setting of the element's value in one API call.
- automatically add required child elements as optional parent elements.
- establish the correct internal sequencing of sub-elements, as defined by the XSD schema, independent and irrespective, from the order the application actually creates the XML hierarchy.

APDataCollection is the component underlying the mapping from the Transaction Definition File (TDF) to the underlying XML data store. The syntax available in the TDF to map fields from/to the data store is that of the APDataCollection. This syntax is a modified XPath syntax similar to W3 XPath.

## APDataCollection Properties

The following are the APDataCollection property in the framework.properties file.

Property	Description	Default Value	Recommended Setting (if any)
default_xml_pretty_format		false	
system_generated_id_attribute_type	<p>This property dictates the use of globally or document unique IDs.</p> <ul style="list-style-type: none"><li>• If this property is missing or is equal to the value 'guid,' then a 36 byte globally unique ID generator engages.</li><li>• If the value is 'docuid,' then a document unique sequence number generates.</li></ul> <p>The value generates in the following format: Nnnnnnnn (where N is the character N and nnnnnnnn is a sequence number unique within the scope of both the current and original documents maintained by APDataCollection).</p>	guid	
autoMaintainIdAttributes	Engages the automatic ID attribute maintenance at application scope.	true	

## Data Schemas and Data Schema Resources

The DATA\_SCHEMA and DATA\_SCHEMA\_RESOURCES properties are the way in which an application can make the XML schemas known to Agencyport XML Engine (AXE). Both properties support the notion of multiple schemas via a semi-colon delimited list.

- DATA\_SCHEMA assumes that the XML schema files are physically located on the file system and is the traditional way of telling AXE about your schema.

- DATA\_SCHEMA\_RESOURCES assumes that the schemas are either available to the class loader or is a URL. An application can use this technique if they choose to bundle resources, such as schemas in a JAR file.

These entries list the data schemas known to the system. The simple syntax for "naming" an individual schema is:

```
xsd_filename[Element:MyRootElementName]
(where MyRootElementName becomes the value of the target attribute in the transaction file)
#####
DATA_SCHEMAS
#####
DATA_SCHEMAS+=${shared_resources_root}/schemas/ap_templates_extension.xsd[Element:BOPPolicy QuoteInqRq,
Element:BOPPolicyModRq, Element:BOPPolicyRenewRq]
```

## Characteristics of AXE's XPath Syntax

- Access to XML data via AXE XPath supports the following:
  - o "XPath-like" syntax used for both retrieval of XML data, as well as the update (including delete and insert) of that data.
  - o A subset carved from full W3C XPath support to support the majority of needs.
  - o **Enables the same expression to support both retrieval and update/delete sides of XML access.**
- W3C XPath and AXE "XPath-like" comparison are:
  - o AXE XPath used for both update and retrieval of XML data values, W3C XPath supports retrieval only.
  - o AXE XPath uses '.' token as element separation token versus '/'. For example:

```
InsuredOrPrincipal[InsuredOrPrincipalInfo.InsuredOrPrincipalRoleCd="Insured"].GeneralPartyInfo.NameInfo.CommlName.
CommercialName
o AXE XPath always assumes the root element as the current context and is never explicitly designated.
o AXE XPath supports only predicates containing equality condition(s).
o AXE XPath supports compound predicate only under the condition that all of the predicates are logical ANDed (logical OR not supported).
o Other W3C functionality not supported in AXE Path:
 ▪ No wild cards (someParent/*/someElementName)
 ▪ position()=last
o Accessing repeating elements
 ▪ An AP XPath expression is typically combined with Java int [] arrays for specifying the repeating levels in a given hierarchy.
 ▪ AP XPath position for repeating elements is zero based rather than one based.
```

## ID Attribute Generation

Unique IDs are generated and applied to elements automatically by the AXE framework.

The RandomGUID class generates 36 byte globally unique IDs. For upload and policy change work items, this can increase the XML payload size by a factor of 2 -> 3 since ID attribute values are applied to all of the elements.

The system\_generated\_id\_attribute\_type property supports the creation of shorter values for the ID attribute.

- If this property is missing or equal to the value of guid, then the historical usage of the RandomGUID class is engaged, producing the generation of the 36 byte globally unique ID values.
- If the value is equal to docuid, then a unique document sequence number generates. The value generated follows the following format: Nnnnnnnn (where the N is the character N and nnnnnnnn is a sequence number unique within the scope of both the current and original documents maintained by APDataCollection).

If you want to dynamically generate an ID value from custom code, conforming to the specified format in the property file, API APDataCollection.generateUID() exposes this same functionality to Java application programmers. The following XML stream includes the docuid property setting:

```
<PersAutoPolicyQuoteInqRq> <RqUID>E86640F5-50EC-4142-C53A-FD0F8C225D56</RqUID> <TransactionRequestDt id="N1">2004-07-27</TransactionRequestDt> <CurCd id="N2">USD</CurCd> <Producer id="N3"> <GeneralPartyInfo id="N4"> <NameInfo id="N5">
<CommlName id="N6"> <CommercialName id="N7">Barefoot Bay / Towerhill Partners</CommercialName> </CommlName> </NameInfo>
```

# XML Schema

The APDataCollection is based on a valid XSD schema. The schema supplies the following important pieces of information to the AXE:

- defines the XML hierarchy that the application will use
- defines which elements are required and optional
- defines which elements repeat
- defines the sequencing, or the order of sub elements, in relation to the parent element

An XSD schema contains support for many complex features, such as data types for element values, enumerations for lists, etc. The APDataCollection supports only a core set of features required by the framework.

The following table defines the behavior of AXE in conjunction with the relevant XSD expressions:

AXE characteristic	XSD Syntax	AXE default if not explicitly specified in XSD
Element is required	minOccurs="n" (where n is greater than zero)	Element is required
Element is optional	minOccurs="0"	Element is required
Element is repeating	maxOccurs="n" or "unbounded" (where n is greater than 1)	Element is not repeating
Element is non repeating	maxOccurs="1"	Element is not repeating
Element is mutually exclusive with other sibling elements defined within group	<xsd:choice>	N/A

## Data Schemas and Data Schema Resources

The DATA\_SCHEMAS and DATA\_SCHEMA\_RESOURCES properties are the way in which an application can make the XML schemas known to AXE. Both properties support the notion of multiple schemas via a semi-colon delimited list.

The first property, DATA\_SCHEMAS, assumes that the XML schema files are physically located on the file system and is the traditional way of telling AXE about your schema.

The second property, DATA\_SCHEMA\_RESOURCES, assumes that the schemas are either available to the class loader or is a URL.

An application can use the second technique if they choose to bundle resources, such as schemas, in a JAR file.

These entries list the data schemas known to the system. The simple syntax for "naming" an individual schema is:

```
xsd_filename[Element:MyRootElementName]
```

In the transaction file, MyRootElementName becomes the value of the target attribute:

```
#####DATA_SCHEMAS # Supports multiple schema files separated by semi-colons. #####ACORD_FILE_NAME_1_7_0=1_7_0/ap_acord-pc-v1_7_0-nodoc-nocodes.xsd PRECONDITIONS_SCHEMA_FILE_NAME=preconditions/preconditions.xsd TRANSACTION_DEFINITION_RULE_BASE_SCHEMA_FILE_NAME=behavior/transactionDefinitionBehavior.xsd DATA_SCHEMAS=${my_context_path}WEB-INF/schemas/${ACORD_FILE_NAME_1_7_0}[Element:PersAutoPolicyQuoteInqRq,Element:PersAutoPolicyModRq,Element:WorkCompPolicyQuoteInqRqElement:WorkCompPolicyModRq];\\ ${my_context_path}WEB-INF/schemas/${PRECONDITIONS_SCHEMA_FILE_NAME}[Element:PersonalAutoPreConditions];\\ ${my_context_path}WEB-INF/schemas/${PRECONDITIONS_SCHEMA_FILE_NAME}[Element:WorkersCompPreConditions];\\ ${my_context_path}WEB-INF/schemas/${TRANSACTION_DEFINITION_RULE_BASE_SCHEMA_FILE_NAME}[Element:transactionDefinitionBehavior] DATASERVICES_SCHEMA_DIR=schemas/ CHOICEPOINT_SCHEMA_FILE=${DATASERVICES_SCHEMA_DIR}choicepoint.xsd DATA_SCHEMA_RESOURCES=http://wiki.keyonce.com/documents/AgencyPortal/schemas/transactionDefinitionBehavior.xsd[Element:transactionDefinitionBehavior];\\ ${CHOICEPOINT_SCHEMA_FILE}[Element:NCFRequest,Element:NCFResponse]
```

## Creating a XSD Schema

Creating XSD files from scratch is difficult with a plain text editor. It is well worth investing in a tool, such as Altova's XML Spy™, for creating XSDs from scratch or for altering existing XSD files.

If the application data model is based on ACORD, you can use existing ACORD XSD unchanged or alter it to fit the requirements of the application.

Another way to create an XSD is to generate an XML instance based on the desired XML structure, open it in a tool, such as Altova's XML Spy™, and then have the tool generate the schema. Refer to the [XML Schema Part 0](#) documentation section of the W3C website for more information on the source specification for XSD.

## Referencing an XML Schema from the TDF

The data schema file describes the data store. The data elements transferred to/from the physical data store and the HTML pages are, in a sense, managed by the XML schema associated with the transaction. The default schemas used in most implementations are defined by ACORD. In general, each line of business has a specific schema (entities common across lines of business are, of course, shared). The fieldIds found in the TDF "map" directly to the element names found in the XML schema and are, thus, ACORD standard data field names. To create a custom XML schema and associate it to a transaction, refer to the following example:

### Example

The following is a small sample of a hypothetical schema. The "top level" element is `DirectorsAndOfficersPolicy` and it contains a number of "sub elements." One of these "sub elements," `Comm1PolicyType`, is illustrated. `Comm1PolicyType` in turn "explodes" into further elements.

```
<xs:element name="DirectorsAndOfficersPolicy"> <xs:complexType> <xs:sequence> <xs:element name="Producer" type="ProducerType"/>
<xs:element name="Insured" type="Insured"/> <xs:element name="Comm1Policy" type="Comm1Policy"/> <xs:element name="Location"
type="Location"/> <xs:element name="DOLineBusiness" type="DOLineBusinessType"/> <xs:element name="Messages"
type="MessagesType" minOccurs="0"/> <xs:element name="FileAttachment" type="FileAttachmentType"/> </xs:sequence>
</xs:complexType> </xs:element> <xs:complexType name="Comm1Policy"> <xs:sequence> <xs:element ref="PolicyNumber"/> <xs:element
ref="CompanyProductCd"/> <xs:element name="ContractTerm" type="ContractTermType" minOccurs="0"/> <xs:element
ref="RateEffectiveDt"/> <xs:element name="QuoteInfo" type="QuoteInfoType" minOccurs="0"/> <xs:element name="Remarks"
type="RemarksType" /> <xs:element name="MiscParty" type="MiscPartyType"/> <xs:element name="Comm1PolicySupplement"
type="Comm1PolicySupplement"/> </xs:sequence> </xs:complexType>
```

The transaction definition file "root" for the above sample would look something like the following:

```
<transaction id="DO" title="Directors and Officers" target="DirectorsAndOfficersPolicy" ...>
```

And a field entry would look something like the following:

```
<fieldElement type="text" id="Comm1Policy.PolicyNumber" label="Policy Num" .. />
```

The AXE engine constructs the target field name using the bold fields and comes up with `DirectorsAndOfficersPolicy.Comm1Policy.PolicyNumber`. It is then able to navigate to the proper place in the physical data store and update or retrieve the data value coming in from, or going out to, the browser.

# Faster XML Parsing

Under heavy load, the demand for JAXP SAX parsers is very high. Prior to AgencyPortal 5, the standard practice for parsing an XML stream into a DOM instance used the following JDOM code:

```
SAXBuilder saxBuilder = new SAXBuilder();
try {
 return saxBuilder.build(inputStream);
} catch (IOException exception) {
 throw new APException(exception);
}
```

This code seems harmless in and of itself; however, when there is a high demand for many `SAXBuilder` instances, they are created (new `SAXBuilder`) across many threads at the same time a concurrency locking situation arises down in the core of the JAXP implementation, causing a bottleneck. This situation was discovered in our own performance testing at Agencyport, but was also brought to our attention by one of our customers. To address this situation, a `SAXBuilder` object pool was created. Instead of creating a new instance, the following logic is used everywhere in the framework where a `SAXBuilder` is needed:

```
import com.agencyport.xml.JDOMFactory; SAXBuilder saxBuilder = JDOMFactory.get().reserveSAXBuilder(); try { return
saxBuilder.build(inputStream); } catch (IOException exception) { throw new APException(exception); } finally {
JDOMFactory.get().unreserveSAXBuilder(saxBuilder); }
```

Custom application logic should be enhanced to use this technique as well. The importance of the final block approach cannot be overstated. If you need a SAXParser to validate, there is a different set of APIs on the `JDOMFactory` class that must be used for reserving and then unreserving the SAX builder instances.

```
reserveValidatingSAXBuilder() unreserveValidatingSAXBuilder()
```

The default pool size is 10 instances. If demand goes beyond that, then new instances are created on the fly and returned; they will not be pooled. If you need to alter the default pool size of 10, then the following application property can be configured:

```
SAXBuilder.Pool.max_pool_size=nnn Where nnn is a positive integer
```

Testing has verified that under heavy demand for `SAXBuilder` instances, the object pool approach does pay off. Product development has carried out JUnit testing that compares the use of `JDOMFactory` with the traditional new `SAXBuilder()` approach. Each test spins up 25 concurrent threads, each thread parsing the same XML stream 100 iterations (total 2500 parsed documents resulting) using the default object pool size of 10. The following information is notable:

```
@ Test public void testJDOMFactoryEfficiency() { try { //Load classes on both sides for a fairer test runThreads(false, 1, 1); runThreads(true,
1, 1); long timeForSAXBuilders = runThreads(false, NUM_THREADS, NUM_ITERATIONS); long timeForJDOMFactory = runThreads(true,
NUM_THREADS, NUM_ITERATIONS); Assert.assertTrue("Time for JDOM Factory based parsing was more inefficient than using new SAX
Builders", timeForJDOMFactory < timeForSAXBuilders); System.out.println("Time in ms using new SAX Builders = " +
timeForSAXBuilders); System.out.println("Time in ms using Pooled SAX Builders = " + timeForJDOMFactory); } catch (Exception exception)
{ exception.printStackTrace(); } }
```

Time in ms using new SAX Builders = 1816

Time in ms using Pooled SAX Builders = 498

```
Apr 28, 2014 10:55:13 AM com.agencyport.shutdown.ApplicationShutdownOfficer shutdown
```

```
INFO: Releasing resources associated with resource destroyer: 'com.agencyport.xml.JDOMFactory'
```

```
Apr 28, 2014 10:55:13 AM com.agencyport.pool.ObjectPool release
```

```
INFO: Statistics for object pool: hash(228893339), Pool key: 1, Max pool size: 10, Current pool
size: 10, Number of reserve requests: 2501, Number of unreserve requests: 2501, Number of pooled
objects created: 10, Number of transient objects created: 255, Number of times pooled objects
reused: 2236, Cache Hit Ratio: 89.40423830467813%
```

# Advanced Data Management

Simple data management can be described as the simple movement of data between the value as it is stored in the XML document and how it is rendered on the HTML page. This is often a one-to-one relationship between the XML data value and the HTML field. It can also describe a situation where all of the fields collected on a roster page fit neatly within the roster source as ACORD defines that particular roster source aggregate. When either one, or both, of these patterns are violated, some course of action needs to take place.

Advanced data management is a general term used to identify one or more techniques for handling complex data management situations like these and others. Another notable challenge faced by developers is the cleanup of XML content on page revisits for optional field values when they are removed or cleared by the user.

The framework provides several ways for handling these situations. The following is an inventory of the more predominant techniques you can use to solve some of the challenges that developers have historically encountered:

- **Views** - Views can be used to handle many situations where the XPath "homes" for the XML values are dynamically governed by the selected list option that drives said field itself or by the selected list option of a dependent field. As of AgencyPortal 5, views have addressability to the HTML data container. Custom views can also be used to compute a second field from a primary field.
- **Group IDs and referential integrity processors** - Both of these techniques are typically used together to clean up XML content when the user has removed one or more entities from the work item XML. As of AgencyPortal 5, views can contribute to a common delete point via the `groupId` attribute.
- **Empty aggregate removal transformer** - During a series of XML transformations, this transfer can be added as one of the last steps to ensure that empty XML aggregates are properly removed.
- **ACORD simplification** - This is the main technique used to solve the situation when data values collected on a roster page don't fall cleanly under the roster source aggregate, as defined by ACORD. It can also be used to handle other corner case data update scenarios. An additional responsibility accompanies the technique of creating two sets of transformations to support both Agencyport's ACORD simplified and the ACORD standard data models.

## Note

The ACORD Help File, which describes the ACORD XML standard, is a highly beneficial tool to use. This file is available to ACORD members. For more information, and to obtain a copy of the help file, [contact ACORD](#) or refer to the [ACORD website](#).

- **Configurable index management** - Many situations arise, especially in commercial lines, where the roster source defines an XML path with more than one repeating element in it. Prior to AgencyPortal 4.1, applications had to write custom code to properly initialize a index management state during IPDTR and when processing the submission for a roster add or edit operation itself. Now, an index element type is available to the roster page definition, which provides a syntax that the framework would interpret for deriving the index management state.

## Views

This section includes a few techniques in which views can be used for in the area of advanced data management. Refer to the [Views](#) section for a complete review of this topic.

The following are a few examples illustrating these techniques:

### Example 1

In this example, there is a need for a custom view that will adjust the XPath for a field that is based on the select list option from another field.

The `AlternateSearchIdView` actual view in the AgencyPortal 5.2 template core illustrates this scenario. This view takes advantage of a feature in AgencyPortal 5.2 that allows access to the HTML data container, which is something that was not available in previous versions of the framework.

The business scenario under consideration is the limit amount for hired, non-owned coverage on the BOP LOB. The coverage code that qualifies this limit is HRDBD, NOWND or HNA, depending on the coverage type code field that the user selects on the preceding field.

You will notice this looks like your typical mutually exclusive fieldset based view except for the `searchId` custom argument, which identifies the ACORD ID of the field that should be used to select the correct field set in the view's metadata.

## Hired and Non-Owned Auto Liability

Coverage Type

Hired/Borrowed

Coverage Not Requested

**Hired/Borrowed**

Non Owned

Hired/Nonowned

Coverage Type

Hired/Borrowed

Limit

\$300,000

**\$300,000**

\$500,000

\$1,000,000

\$2,000,000

**This coverage adds an exclusion for the activity described below.**

The following is the field element and select list configuration for the coverage type code field that this field depends on:

```
<fieldElement type="selectlist" id="HiredNonOwnedAutoLiability.CoverageCd" label="Coverage Type"
viewId="bop Views.xml:BOPLineBusiness.LiabilityInfo.CommlCoverage.HIRED_NON_OWNED.CoverageCode"
uniqueId="hiredNonOwnedAutoLiabilityCoverageCd"> <optionList reader="xmlreader" source="codeListRef.xml"
target="coverageNotRequested" /><optionList reader="xmlreader" source="bopCodeListRef.xml"
target="hiredAndNonOwnedAutoLiabilityCoverageType" /></fieldElement>

<optionList id="hiredAndNonOwnedAutoLiabilityCoverageType"><option value="HRDBD">Hired/Borrowed</option> <option
value="NOWND">Non Owned</option> <option value="HNA">Hired/Nonowned</option> </optionList>

The following is the field element and view metadata for the coverage limit itself:
```

```
<fieldElement type="selectlist" id="HiredNonOwnedAutoLiability.Limit" label="Limit"
viewId="bop Views.xml:BOPLineBusiness.LiabilityInfo.CommlCoverage.HIRED_NON_OWNED.Limit"
uniqueId="hiredNonOwnedAutoLiabilityLimit"> <optionList reader="xmlreader" source="codeListRef.xml" target="blank" /><optionList
reader="xmlreader" source="bopCodeListRef.xml" target="hiredAndNonOwnedAutoLiabilityLimit" /></fieldElement>

<view id="BOPLineBusiness.LiabilityInfo.CommlCoverage.HIRED_NON_OWNED.Limit" title="View for handling hired/non owned
coverage limit" type="mutuallyExclusiveFieldSets" deleteMode="onEmptyValue"> <customArgument name="searchId"
value="HiredNonOwnedAutoLiability.CoverageCd" /><fieldSet searchId="HRDBD"> <field
type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='HRDBD'].Limit.FormatInteger</field> <deleteField
type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='HRDBD']</deleteField> </fieldSet> <fieldSet searchId="NOWND">
<field type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='NOW ND'].Limit.FormatInteger</field> <deleteField
type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='NOWND']</deleteField> </fieldSet> <fieldSet searchId="HNA">
<field type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='HNA'].Limit.FormatInteger</field> <deleteField
type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='HNA']</deleteField> </fieldSet>
<viewClassName>com.agencyport.shared.custom.AlternateSearchIdView</viewClassName> </view>
```

The following is the custom view source. Note how the custom argument search ID is used:

```
/* * Created on Aug 13, 2014 by nbaker AgencyPort Insurance Services, Inc. */ package com.agencyport.shared.custom; import java.util.Map;
import org.jdom2.Element; import com.agencyport.domXML.APDataCollection; import com.agencyport.domXML.view.View; import
com.agencyport.shared.APException; import com.agencyport.webshared.HTMLDataContainer; import
com.agencyport.workitem.context.WorkItemContextStore; /* * The AlternateSearchIdView class supports a search id for mutually exclusive
field sets that is different from the data value itself. * @since 5.0 */ public class AlternateSearchIdView extends View { /* * The
<code>serialVersionUID</code> */ private static final long serialVersionUID = 1377654133640169919L; /* * The
<code>SEARCH_ID</code> is a constant for the value 'searchId'. */ private static final String SEARCH_ID = "searchId"; /* * The
<code>searchId</code> is the search id value. This is used as a key into the HTML data container. */ private String searchId; /* * Constructs an
instance. */ public AlternateSearchIdView() { } /* * {@inheritDoc} */ @Override public void init(Element viewElement) throws APException
{ super.init(viewElement); // initialize superMap<String, String> customArguments = CustomViewParser.parseView(viewElement, false,
SEARCH_ID); searchId = customArguments.get(SEARCH_ID); } /* * {@inheritDoc} */ @Override public void update(APDataCollection
```

```
apData, int[] idArray, String value, String associatedElementPath) { HTMLDataContainer htmlDC =
WorkItemContextStore.get().getHTMLDataContainer(); String searchIdValue = htmlDC.getStringValue(searchId, null); update(apData, idArray,
value, searchIdValue, associatedElementPath); } }
```

## Example 2

In this example, there is a need for a custom view that derives the value of one field based on the value of another field multiplied by some factor.

Formula: final XML value = (*dependent field value \* a multiplier factor from a selection option*).

The business scenario: The Prds/Cmp Ops liability coverage limit in BOP is computed by taking the liability occurrence limit and multiplying it by a factor (typically either 2x or 3x). In the case that follows, if the user selects the 1,000,000/2,000,000 option for the each occurrence liability limit and select 2x in the Prds/Cmp Ops multiplier, then a value of 2,000,000 (2x1,000,000) should be stored in the Prds/Cmp Ops liability limit field in the resultant XML.



The following is the field, its select list options, view definition, as well as the Java source supporting this advanced data management challenge:

```
<fieldElement type="selectlist" id="PRDCO.Multiplier" label="Prds/Cmp Ops Agg Mult" required="true" size="10" defaultValue="2" viewId="bopViews.xml:BOPLineBusiness.LiabilityInfo.CommlCoverage.PRDCO"> <optionList reader="xmlreader" source="codeListRef.xml" target="selectOne" /><optionList reader="xmlreader" source="bopCodeListRef.xml" target="prdCOMultiplier" /></fieldElement>

<optionList id="prdCOMultiplier"> <option value="2">2x</option> <option value="3">3x</option> </optionList>

<view id="BOPLineBusiness.LiabilityInfo.CommlCoverage.PRDCO" title="Multiplier View for PRDCO coverage" type="standard" deleteMode="onEmptyValue"> <customArgument name="source" value="BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='EAOC']&.Limit.FormatInteger"/> <fieldSet> <field type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='PRDCO']&.Limit.FormatInteger</field> <deleteField>BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='PIADV']</deleteField> </fieldSet> <viewClassName>com.agencyport.bop.custom.MultiplierView</viewClassName> </view>

/**Created on Aug 13, 2014 by nbaker@AgencyPortInsuranceServices, Inc.*/
package com.agencyport.bop.custom; import java.util.Map; import org.jdom2.Element; import com.agencyport.domXML.APDataCollection; import com.agencyport.domXML.view.Field; import com.agencyport.domXML.view.View; import com.agencyport.fieldvalidation.validators.builtin.NumericValidator; import com.agencyport.shared.APEException; import com.agencyport.shared.custom.CustomViewParser; /**The MultiplierView class manages the update and display of fields whose values are determined by a multiplier factor upon another data entity in the workitem. On update, result=multiplicator*source. On display, *multiplicatorfactor=result/source. The multiplicator factor is the value which is passed to this class's update method by the framework and the return value of this class's read method. */ @since 5.0 replacement for old SFHs OccurrenceMultiplierHelper and Pers Advertising Injury Helper*/ public class MultiplierView extends View {
 /**The serialVersionUID */
 private static final long serialVersionUID=8299687731656501042L; /**The SOURCE</code> is the name of the custom attribute that links to the related source. */
 private static final String SOURCE="source"; /**The ZERO</code> is a constant or a string version of the value 0. */
 private static final String ZERO="0"; /**The sourceId</code> is the id for the source data entity upon which this and the factor is multiplied to arrive at the result. */
 private String sourceId; /**The viewElement</code> is a reference to the XML view metadata element. */
 private Element viewElement; /**NON_NUMERIC_ERROR_MSG_TEMPLATE</code> is an error message template used when non numeric operands are encountered. */
 private static final String NON_NUMERIC_ERROR_MSG_TEMPLATE="The view requires numeric operands. One or both of the operands were determined not to be numeric: %s %s"; /**Constructs an instance. */
 public MultiplierView() { }
 /**@inheritDoc */
 @Override public void init(Element viewElement) throws APEException {
 super.init(viewElement); //initializes super this. viewElement=viewElement; Map<String, String> customArguments=CustomViewParser.parseView(viewElement, false, SOURCE); sourceId=customArguments.get(SOURCE); }
 /**We consider a value of zero as "empty" */
 @Override public boolean isEmptyValue(String value) {
 return Field.isEmptyValue(value)||ZERO.equals(value); }
 /**@inheritDoc */
 @Override public void update(APDataCollection apData, int[] idArray, String value, String associatedElementPath) {
 String multiplicatorFactorValue=value; if(!Field.isEmptyValue(multiplicatorFactorValue)) {
 String sourceValue=apData.getFieldValue(sourceId, idArray, ZERO); if(!Field.isEmptyValue(sourceValue)) {
 if(!NumericValidator.isInteger(sourceValue)||!NumericValidator.isInteger(multiplicatorFactorValue)) { try {
 CustomViewParser.raiseException(String.format(NON_NUMERIC_ERROR_MSG_TEMPLATE, sourceValue, multiplicatorFactorValue),); }
 } catch (APEException e) { throw new RuntimeException(e); }
 }
 }
 }
}
```

```

viewElement); }catch(APExceptionapException){ thrownewIllegalArgumentException(apException); }
}intmultiplierFactor=Integer.parseInt(multiplierFactorValue);intsource=Integer.parseInt(sourceValue);inttarget=source*multiplierFactor;super.update(apData, idArray, Integer.toString(target), associatedElementPath);return; } }super.update(apData, idArray, value, associatedElementPath);//ThiswilldeletetheentityduetoouroverloadofthemethodisEmptyValue() }/**{* @inheritDoc
}*/@OverridepublicStringread(APDataCollectionapData, int[]idArray, StringdefaultValue, StringassociatedElementPath)throwsAPException{
StringtargetValue=super.read(apData, idArray, defaultValue, associatedElementPath);if(!isEmptyValue(targetValue)){
StringsourceValue=apData.getFieldValue(sourceId, idArray, ZERO);if(!isEmptyValue(sourceValue)){
intsource=Integer.parseInt(sourceValue);inttarget=Integer.parseInt(targetValue);intmultiplier=target/source;returnInteger.toString(multiplier); }
}returnZERO; } }

```

## Group ID Processing

Group IDs have been available for some time. They are needed for optional fields only and provide an alternate delete point (usually higher up in the XML hierarchy) for one or more related optional fields to support the situation of a user revisiting a page and clearing out or emptying the field(s).

The `groupId` attribute contains the delete point to apply if all of the fields sharing the same `groupId` value are both optional and empty. Multiple optional field elements that relate a common parent should contain a reference to the same `groupId` value. Group ID processing takes place after all field level elements have been rendered within the scope of `executeDataStagingForModification()` before process side connectors are called.

The following is an example of two fields that are related, a deductible type code and the deductible amount. When both are emptied, you don't want the parent coverage aggregate to remain in the XML document:

### Example

```

<fieldElement type="selectlist" id="BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='PD'].Deductible.DeductibleTypeCd"
label="PD Deductible Type" groupId="BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='PD']"> </fieldElement> <fieldElement
type="selectlist" id="BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='PD'].Deductible.FormatInteger" label="PD Liability
Deductible" groupId="BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='PD']"> </fieldElement>

```

## Referential Integrity

Prior to the 4.1 release of AgencyPortal, referential integrity was handled entirely with application custom code. A referential integrity architecture was introduced in 4.1. The built-in referential integrity operations supported by the product are:

1. Maintaining the positional integrity of aggregates whose parentage may need to change due to a key data value change of a related foreign aggregate. An example of this is a workers comp\_location info aggregate that is nested under a given work comp rate state. If the state province code on the location aggregate to which it points changes, then that workers comp location info aggregate needs to move under a work comp rate state whose state province code matches the updated state province code on the location aggregate.
2. Maintaining the integrity of aggregates with invalid references to foreign aggregates. If an aggregate is deleted via a roster delete operation, then any outstanding references to that deleted aggregate need some kind of attention. The framework provides the ability to perform one of the following:
  - Remove the lingering invalid ID references only, leaving the rest of the aggregate intact.
  - Delete all aggregates in their entirety, possessing outstanding invalid ID references.
  - Custom handling on an aggregate-by-aggregate basis with custom code.
3. Removing empty aggregates at a document level.
4. Customizing through extension of the base referential integrity processor class.

AgencyPortal does not engage referential integrity operations unless explicitly configured. The configuration model is application property oriented and supports the following basic properties:

Base Application Property Name	Data Type	Description	Default Value	Sample Value
referential_integrity_manager_class_name	String	Specifies the Java class name that implements the <code>com.agencyport.domXML.refintegrity.IReferentialIntegrityManager</code> implementation	N/A	com.agencyport.data.ComprehensiveReferentialIntegrityManager
remove_empty_aggregates	Boolean	Specifies whether to engage empty aggregate removal	Depends on whether a work item has an original	true

<b>Base Application Property Name</b>	<b>Data Type</b>	<b>Description</b>	<b>Default Value</b>	<b>Sample Value</b>
			document. If it does, then the default is false; otherwise, it is true.	
remove_aggregates_in_valid_idrefs	Boolean	Specifies whether to remove aggregates in their entirety with invalid lingering ID references	false	true
remove_invalid_idrefs	Boolean	Specifies whether to remove invalid ID references, leaving their parent aggregates intact	true	false
element_paths_to_ignore_values	String list (semicolon delimited)	Specifies a list of AXE element paths whose values are to be ignored when determining the candidacy of sibling and child empty aggregate removal.	N/A	WorkCompLineBusiness.WorkCompRateState.StateProvCd

An application property name prefixing facility supports the ability to configure those properties described in the previous section independently by the following aspects:

- LOB code
- Whether policy change management is active for a work item (i.e., applications may want to vary one or more of the above properties, depending on whether an original document is available. Classically, this would apply to work items whose data is sourced from an external system)
  - Uploaded work items
  - Endorsement based work items
  - Renewal based work items

If policy management is in effect for a given work item, the prefix precedence follows this order:

1. LOB code.policychange.baseproperty name
2. LOB code.base property name
3. policychange.base property name
4. base property name alone
5. default

If policy management is not in effect, the prefix precedence follows this order:

1. LOB code.base property name
2. base property name alone
3. default

The literal policy change allows properties to be independently configured based on whether policy change is engaged for a given work item within the same LOB family.

The following are a few examples:

### Example

```
#####
Referential Integrity management configuration
AUTOB.referential_integrity_manager_class_name=com.agencyport.commAuto.custom.CommlAutoReferentialIntegrityManager
AUTOB.element_paths_to_ignore_values=CommILineBusiness.CommlRateState.StateProvCd AUTOB.maintain_dependent_indices=true
AUTOB.policychange.maintain_dependent_indices=false
#####
Referential Integrity management configuration
BOP.referential_integrity_manager_class_name=com.agencyport.domXML.refintegrity.BasicReferentialIntegrityManager
BOP.remove_aggregates_with_invalid_idrefs=false BOP.maintain_dependent_indices=true
BOP.policychange.maintain_dependent_indices=false BOP.remove_invalid_idrefs=true
#####
Referential Integrity management configuration

```

```
configuration #####
WORK.referential_integrity_manager_class_name=com.agencyport.workerscomp.custom.WorkersCompReferentialIntegrityManager
WORK.element_paths_to_ignore_values=WorkCompLineBusiness.WorkCompRateState.StateProvCd
WORK.remove_aggregates_with_invalid_idrefs=true WORK.maintain_dependent_indices=true
WORK.policychange.maintain_dependent_indices=false
```

# ACORD Simplified Programming

Agencyport introduced the special field helper (SFH) widget in 3.x to support the retrieval, update, deletion and recovery of data values stored outside of the roster aggregate data source configured in the TDF. The insurance line of business that originally drove its creation was Business Owners Policy (BOP). However, as subsequent commercial lines of business were implemented, specifically those using the ACORD aggregates (CommISubLocation, CommIPropertyInfo, GeneralLiabilityClassification), the incidence of special field handling increased to preserve referential integrity of the ACORD data model as the ACORD XML document is processed.

The SFH widget provided the necessary customlogic to maintain the "foreign" aggregate data relations that the standard AgencyPortal data engine did not natively support. Each discreet user action, such as add, update, delete and recover, was accompanied by its own set of unique challenges, requiring its own brand of custom code.

Over time, ensuing products, such as change management and DTR, added additional responsibilities onto the SFH interface, specifically in the areas of data recovery and data cleanup respectively. Still, with the maturation of the DTR engine, it became possible to configure DTR behaviors without the need to add customcode to the application's precondition Java class implementation.

The toolkit component attempts to shift more and more of the responsibility of configuration from developers onto analysts. With this tool, it became imperative to simplify the TDF authoring process even more. However, one of the biggest obstacles that still remains to achieve this goal are those data elements managed by special field helpers.

# Description of Existing Special Field Helper Functionality

As Agencyport software practices evolved, SFHs came to support functionality beyond its original designated purpose of foreign aggregate management. The following are various SFH capabilities that have evolved over time:

- Foreign aggregate management (initial purpose)
- Dynamic XPath derivation of one or more fields, based on the value of another field.

## Example

BOP hired / non-owned liability coverage.

- Dynamic XPath mapping of a single input control value to/from potentially more than one ACORD data value.

## Example

BOP blanket information (BLDG/BPP/BOTH).

- Derivation of the ACORD data value from the screen value [ACORD value = f(screen value)].

## Example

BOP Prds/Cmp Ops Agg Mult coverage.

- Index management initialization specifically by way of the `SpecialFieldHelper.prepareForUpdate(int)` method for roster sources that have more than one level of repetition.

# Basic Goal and Premise of ACORD Simplified

The main objective of embracing an ACORD simplified XML model is to reduce the amount of custom code needed to maintain the entity relationships between various XML aggregates during normal page processing. This approach permits the ACORD standard schema to be "simplified," using the technique of schema extensions to more effectively utilize the native capabilities of the AgencyPortal TDF driven data engine.

## Example

Standard ACORD models the aggregate `Comm1PropertyInfo` under the `PropertyInfo` parent and ties that information to the `Comm1SubLocation` using reference ID attribution. This simplified technique "allows" an aggregate, such as `Comm1PropertyInfo` to become a child of the `Comm1SubLocation` aggregate.

This approach allows you to change a field, such as the following BOP equipment break down coverage limit, without the need for a special field helper:

```
<fieldElement type="text"
id="SP.BOPLineBusiness.PropertyInfo.Comm1PropertyInfo[SubjectInsuranceCd='BMEBU'].Comm1Coverage[CoverageCd='BMEBU'].Limit.F
ormatInteger" label="Equipt Brkdwn Sub-limit" size="10" maxLength="14" ><validation type="numeric" id="N206" /></fieldElement>
```

To something like this:

```
<fieldElement type="text" id="Comm1SubLocation.Comm1PropertyInfo[SubjectInsuranceCd='BMEBU'].Comm1Coverage[CoverageCd='
BMEBU'].Limit.FormatInteger" label="Equipt Brkdwn Sub-limit" size="10" maxLength="14" ><validation type="numeric" id="N206" />
</fieldElement>
```

This technique simplifies the page-to-page data mapping operations greatly since the data engine native to the product can achieve the read/write data values from/to the XML document without the aid of custom code.

Embracing the notion of ACORD simplification means that the XML documents persisted to the `xmlstore` table would not conform to the standard ACORD data model. The ACORD simplified model is positioned as private to the application and not something you would force your trading partners to comply with; therefore, transformations will be needed to support the ACORD standard definition in your public interfaces.

Although the adoption of the ACORD simplified data model adds the additional task of writing transformations, this task is a finite, more rudimentary task from an application development viewpoint when compared to the daunting responsibility of maintaining the referential integrity of the ACORD standard data model on a page-to-page basis.

# Step-by-Step to ACORD Simplified

This section provides instructions for introducing the ACORD simplified data model for an AgencyPortal based web application. This includes the following:

- [Defining the ACORD Simplified Data Model](#)
- [Registering the ACORD Simplified Model with AXE](#)
- [Adjusting the TDF to Use the ACORD Simplified Model](#)
- [Sample Simplifications](#)

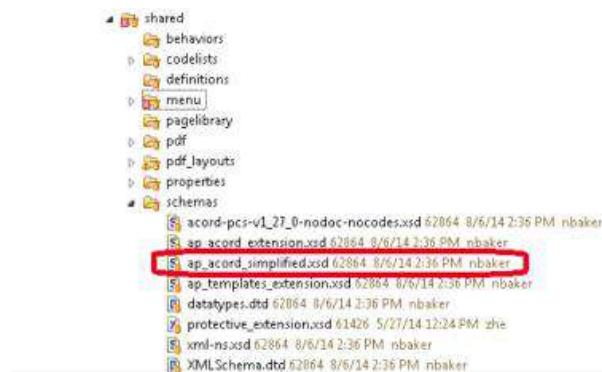
## Defining the ACORD Simplified Data Model

The main technique in the ACORD simplified data model involves extending the ACORD schema for those ACORD complex types that are the pain points in your application or, in other words, involving those aggregates containing fields that would require special field handling if you were still using the standard ACORD data model.

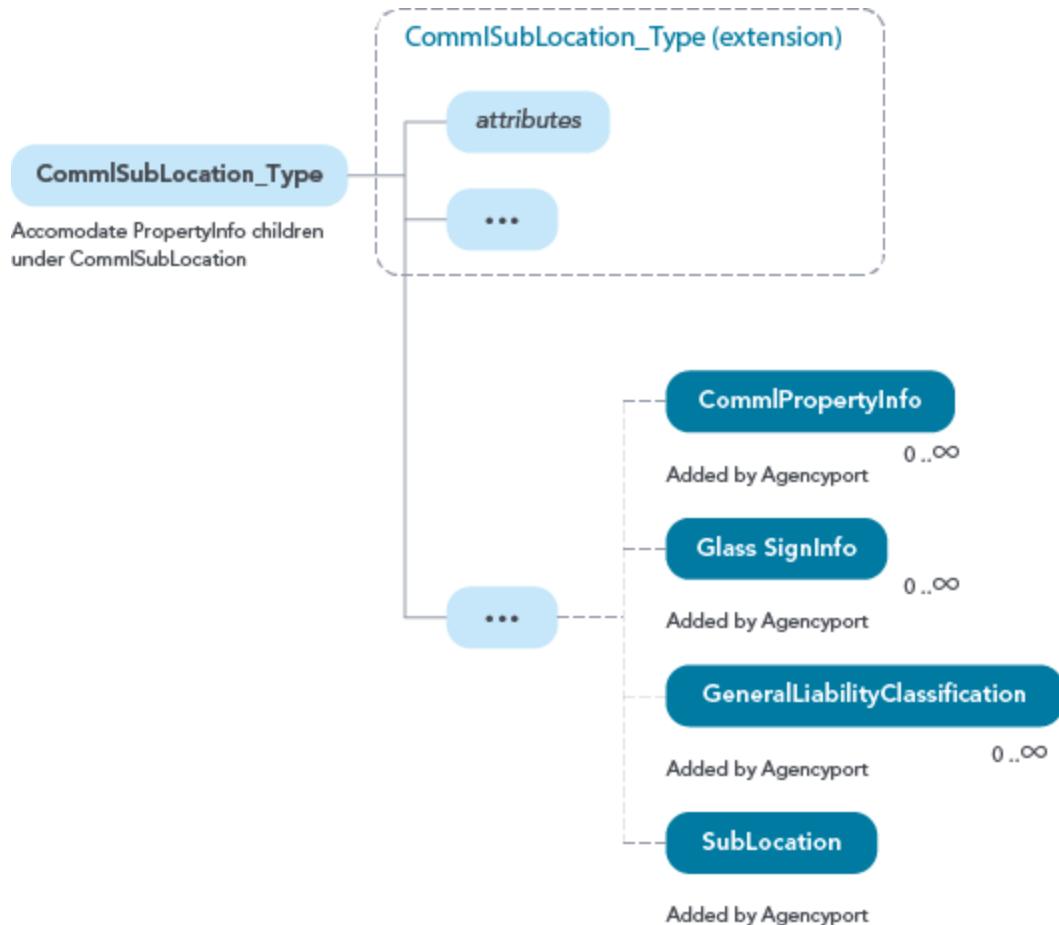
Refer to the following guiding principles:

- First, concentrate on those ACORD aggregates that are foreign to the aggregate context found in your roster source configurations.
- Although foreign aggregate "elimination" will dominate this task, don't limit yourself to just this aspect of the problem domain. With exception of index management responsibilities, which is not relevant in this task, also consider eliminating SFHs for fields with peculiarities of their own. Refer to the [Description of Existing Special Field Helper Functionality](#) section for more information.
- Most of the scenarios detailed in the [Description of Existing Field Helper Functionality](#) section deal with fields whose XPath qualifying predicates are dynamically related to some other changeable field value on the page. An example of this is the `BlanketInfo` aggregate. Refer to the [BlanketInfo](#) section for a detailed explanation of this type of challenge. In some of these situations, you can choose to use an existing or new customview to handle the I/O. In othersituations, you may want to defer those to your ACORD simplified and standard transformations.
- The original standard ACORD homes for these aggregates must remain intact so that the same schema can support both simplified and standard ACORD representations at the same time.
- If you find that you still need a number of SFHs, even after your remodeling effort, then take that model into question.
- The expression of your application's ACORD simplified data model should be accomplished using schema extensions [in your own application schema](#). Making direct updates to either the ACORD or the AgencyPortal template schema files (or `ap_acord_simplified.xsd`) is not an approved technique for modifying any of the ACORD entities.
- Product development offers their expertise to review your application's simplified data model as part of the vetting process.
- DO NOT advance to any further steps in this portion of the documentation until you are totally satisfied with the simplified model you have crafted. Remodeling iterations are fine as long as TDF path changes are not involved. Vetting out the design of the simplified model should be taken seriously. You want your ACORD simplified finalized before applying those new XPaths as part of your application's TDF.

The ACORD simplification modeling work referred to in subsequent sections is the template core bundle in the WEB-INF/shared/schemas folder.



The modeling in this schema relates to the `CommSubLocation` aggregate.



The modeling effort illustrated in the above diagram permits the **CommIPropertyInfo**, **GlassSignInfo**, **GeneralLiabilityClassification** and **SubLocation** aggregates to be valid children of the **CommISubLocation** parent aggregate. This eliminates the need for any special field helpers for TDF fields that would historically be required to maintain the foreign aggregate relationships under the ACORD standard model. Moreover, this technique will permit "standard" AgencyPortal declarative field ID notation to be adopted for many of the fields contained in these aggregates.

Consider the above model an illustration of the technique; the modeling discussions you make can vary.

## Registering the ACORD Simplified Model with AXE

The following is a sample of registering the ACORD simplified model with AXE:

- Assume that you have placed the schema extensions to simplify ACORD for your application in an XSD file whose file name is **abc\_simplified\_acord.xsd** and is deployed in a directory under **WEB-INF** called **shared/schemas**.
- Assume that the LOB for your application is **BOP**.

The following AXE configuration follows:

```
DATA_SCHEMAS+=${my_context_path}WEB-INF/shared/schemas/ap_simplified_acord.xsd[Element:BOPPolicyQuoteInqRq]
```

The next time your server boots, your ACORD simplified data model is active.

## Adjusting the TDF to Use the ACORD Simplified Model

Adjusting the TDF to use the ACORD simplified model will be fairly time consuming, depending on the number of fields that have been impacted. That is why it is important to ensure that the simplified ACORD model has been finalized and thoroughly vetted before starting this step.

If the starting point of your application is one that uses special field helpers (SFHs) heavily, then you will have an extensive job on your hands. You will need to change and verify the XPath home for each field impacted by the simplified remodeling effort.

### Example

You will need to change XPaths from something like this:

```
<fieldElement type="text" id="SP.BOPLineBusiness.PropertyInfo.CommlPropertyInfo[SubjectInsuranceCd='BMEBU'].CommlCoverage[CoverageCd='BMEBU'].Limit.FormatInteger" label="Equipt Brkdwn Sub-limit" size="10" maxLength="14"> <validation type="numeric" id="N206"/> </fieldElement>
```

to something like this:

```
<fieldElement type="text" id="CommlSub Location.CommlPropertyInfo[SubjectInsuranceCd='BMEBU'].CommlCoverage[CoverageCd='BMEBU'].Limit.FormatInteger" label="Equipt Brkdwn Sub-limit" size="10" maxLength="14"> <validation type="numeric" id="N206"/> </fieldElement>
```

### Removing Old SFHs from Your Application

Remove any old SFHs that are no longer needed from the project source folder. You also need to remove the references, such as the following, from the product database XML file:

```
<artifact type="sfh" resourceId="com.agencyport.bop.custom.BOPBuildingHelper" title="Building Helper SFH" id="N109" />
```

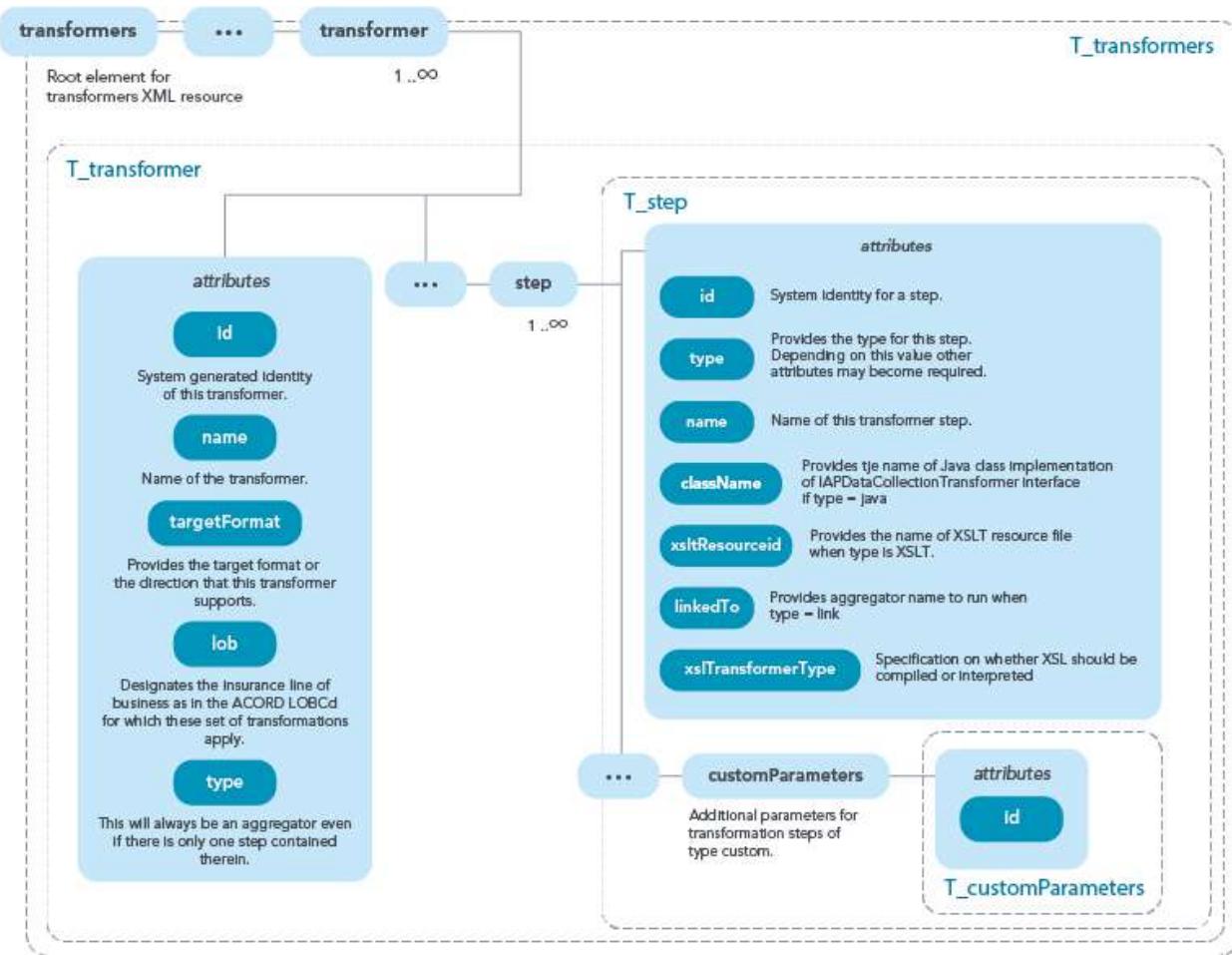
### Creating Necessary Transformers to Support ACORD Simplified To/From ACORD Standard Transformations

Product definitions were enhanced in AgencyPortal 4.1 to support a new built-in resource type (transformer type) with the requisite resource provider implementation.

### Important

It is important to note that these transfer resource types are not visible in the toolkit product menu.

Refer to the [XML schema definition](#) for information on the transfers product type.



The task of building out the transformations can begin once your application's ACORD simplified data model is finalized and made to manifest in your application's extension schema file. Before performing this task, we recommend you first review additional information on transformations.

The product only supports two distinct sets of transformation steps. However, for sake of this discussion only, two are directly relevant to the ACORD simplified data model. The application needs to accommodate a bi-directional transformation paradigm.

- From an ACORD standard representation to your own application's ACORD simplified representation. The most common use of this will be during upload writer processing.
- From your own application's ACORD simplified representation to an ACORD standard representation.

This process is limited to the following directions, as described in this product enumeration:

```

/* * Created on Dec 4, 2009 by nbaker AgencyPort Insurance Services, Inc. */
package com.agencyport.domXML.transform; import
com.agencyport.type.ExtensibleEnum; /**
 * The TargetFormat enumeration delineates the supported target transformation formats. This was
previously a Java enumeration that was converted to a class in version 5.0 to allow applications to invent their own target types.
 */
public class TargetFormat extends ExtensibleEnum<TargetFormat> { /**
 * The <code>ACORDStandard</code> identifies the transformation target format
for the XML format according to the ACORD 1.x schema published by www.acord.org.
 */
public static final TargetFormat ACORDStandard = new TargetFormat("ACORDStandard"); /**
 * The <code>ACORDSimplified</code> identifies the transformation target format for
Agencyport's simplification of the ACORD 1.x format published by Agencyport's schema extensions of ACORD.
 */
public static final TargetFormat ACORDSimplified = new TargetFormat("ACORDSimplified"); /**
 * The <code>ACORDWorkItemDownFill</code> identifies the transformation target format for
WorkItem XML
*/
public static final TargetFormat ACORDWorkItemDownFill = new TargetFormat("ACORDWorkItemDownFill"); /**
 * The <code>ACORDWorkItemUpFill</code> identifies the transformation target format for
WorkItem XML
*/
public static final TargetFormat ACORDWorkItemUpFill = new TargetFormat("ACORDWorkItemUpFill"); /**
 * The <code>ANY</code> denotes a wild card target type that does not have any affinity to any specific target format. * Useful for utility
transformations that are agnostic of the target format. Aggregators of this type can only be accessed by name and not be LOB code and target
format. * @since 5.0 */
public static final TargetFormat ANY = new TargetFormat("ANY"); /**
 * Constructs an instance. * @param
targetFormat is the target format value.
*/
public TargetFormat(String targetFormat){ super(targetFormat, TargetFormat.class); } /**
 * Returns
the target format value.
*/
String getTargetFormat(){ return targetFormat; } /**
 * Sets the target format value.
*/
void setTargetFormat(String targetFormat){ this.targetFormat = targetFormat; } /**
 * Gets the
target format value.
*/
String getName(){ return name; } /**
 * Sets the
target format value.
*/
void setName(String name){ this.name = name; } /**
 * Gets the
target format value.
*/
String getId(){ return id; } /**
 * Sets the
target format value.
*/
void setId(String id){ this.id = id; } /**
 * Gets the
target format value.
*/
String getClassname(){ return className; } /**
 * Sets the
target format value.
*/
void setClassName(String className){ this.className = className; } /**
 * Gets the
target format value.
*/
String getXsitResourceid(){ return xsitResourceId; } /**
 * Sets the
target format value.
*/
void setXsitResourceid(String xsitResourceId){ this.xsitResourceId = xsitResourceId; } /**
 * Gets the
target format value.
*/
String getLinkedTo(){ return linkedTo; } /**
 * Sets the
target format value.
*/
void setLinkedTo(String linkedTo){ this.linkedTo = linkedTo; } /**
 * Gets the
target format value.
*/
String getXsitTransformerType(){ return xsitTransformerType; } /**
 * Sets the
target format value.
*/
void setXsitTransformerType(String xsitTransformerType){ this.xsitTransformerType = xsitTransformerType; } /**
 * Gets the
target format value.
*/
String getCustomParameters(){ return customParameters; } /**
 * Sets the
target format value.
*/
void setCustomParameters(String customParameters){ this.customParameters = customParameters; } /**
 * Gets the
target format value.
*/
String getId(){ return id; } /**
 * Sets the
target format value.
*/
void setId(String id){ this.id = id; }
}

```

```
the enumeration instance for the given name. * @param name is the value of the enumeration. * @return the enumeration instance for the given name. */public static TargetFormat valueOf(String name){ return valueOf(name, TargetFormat.class); } }
```

The product automatically engages the first of these transformations (`ACORDSimplified`) during an upload writer operation. Refer to the following to see how the `ImportBaseServlet.transform()` method calls `ImportBaseServlet.convertACORDStandardToACORDSimplified()` and how it automatically engages the transfer if one has been registered:

```
protected void convertACORDStandardToACORDSimplified(APDataCollection apData, AgencyConnectConversionStepData stepData) throws APException { apData.setDocument((Document) apData.getCurrentDocument().clone(), IXMLConstants.VIEW_CURRENT_ACORD_STANDARD_DOCUMENT, false); IAPDataCollectionTransformer transformer = apData.getRegisteredTransformer(TargetFormat.ACORDSimplified); if (transformer != null) { // Update the current document in place, leaving the ACORD standard document untouched on its own view transformer.transform(apData, TargetFormat.ACORDSimplified, null); writeAPDataToDebugOutput(apData, "convertACORDStandardToACORDSimplified"); } }
```

#### **Creating Transformation Processing Units**

When looking at the `transformer.xsd` XML schema file, notice that the transformer can comprise of one or more steps. Each step performs a discreet set of transformations and internally invokes an implementation of the `com.agencyport.domXML.transform.IAPDataCollectionTransformer` interface.

You can write transformation steps as XSLT or Java. The product has its own wrappers around the XSLT units, conforming to this interface, so there is no need for you to wrap the XSLT. If you package your transformation logical units as Java, you must implement the aforementioned interface. XSLT can be run as compiled or interpreted processing modes. The default processing mode is compiled. You will need to use the interpreted processing mode if you use functions in your XSLT.

The transformation examples in the Sample Transformers section supports the sample BOP simplified data model previously referred to in this documentation.

#### **Configuring the Transformers Resource**

Configuration is necessary to supply the order and type of the transformation units that you have developed. This configuration falls along LOB boundaries nicely. Each transformer step is executed in the order as it is physically sequenced in the resource.

#### **Important**

It is important to point out that for the registration of these transformations to be automatically made, the LOB attribute value on the transformer element must match the LOB on the associated work item's `APDataCollection`.

The following is a sample of the contents in the `bop_transformers.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?> <transformers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://reference.agencyport.com/schemas/4.2/transformer/transformer.xsd"> <transformer name="A P BOP Simplified ACORD to ACORD standard" type="aggregator" targetFormat="ACORDStandard" lob="BOP"> <step name="BOP Transformation target - ACORD Standard Step 1" type="XSLT" xsltResourceId="bop_standard_stage_1.xls"/> <step name="BOP Transformation target - ACORD Standard Step 2" type="XSLT" xsltResourceId="bop_standard_stage_2.xls"/> <step name="Shared Standard Built-in Transformations" type="link" linkedTo="builtin-standardizations"/> </transformer> <transformer name="ACORD standard to AP BOP Simplified ACORD" type="aggregator" targetFormat="ACORDSimplified" lob="BOP"> <step name="BOP Transformation target - ACORD Simplified Step 1" type="XSLT" xsltResourceId="bop_simplified_stage_1.xls"/> <step name="BOP Transformation target - ACORD Simplified Step 2" type="XSLT" xsltResourceId="bop_simplified_stage_2.xls"/> </transformer> </transformers>
```

#### **Important**

It is important to note the following interesting configuration:

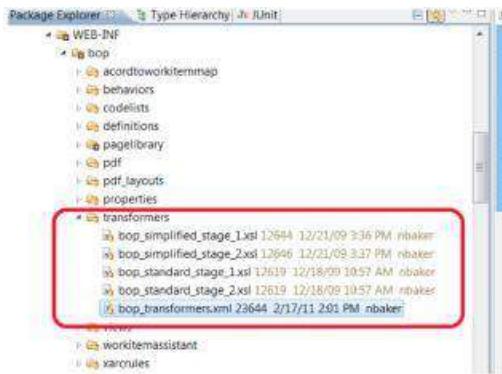
```
<step name="Shared Standard Built-in Transformations" type="link" linkedTo="builtin-standardizations" />
```

This will link to a set of transformations that the product offers out of the box. The functionality that these transformations include is generic and LOB agnostic operations:

- Removal of the `ModInfo.AP_DomainAggregateXPath`
- Removal of the `saved_xxxxxxx` attributes
- Removal of empty aggregates

Changes are your trading partners will not want to receive any of the above content.

The transformation processing units and the transformers XML resource are stored in the system, such as the following:



#### Registering the Transfer Resource in the Product Database

Lastly, you must register the transformer resource(s) in the product database.

#### Example

```
<product type="BOP" version="4.2.0.0" title="Business Owners" path="bop" description="Business Owners Insurance Policy"> <artifact resourceId="bop_transformers.xml" type="transformer" description="" title="BOP Transformations"/> </product> <product type="shared" shared="true" version="4.2.0.0" title="Shared" path="shared" description="Shared Product"> <artifact resourceId="shared_transformers.xml" type="transformer" description="" title="ACORD Standard Transformations"/> </product>
```

#### Converting Application to Use New XPaths

You must make adjustments to any resource that uses an XPath that was changed as a result of the remodeling effort. The following includes some examples of adjustments needed:

- Any APDataCollection or JDOM based Java custom code. This could be in the form of connectors, APCommand extensions, PDF rendering logic, policy summary bean provider logic, etc.
- Any XARC rules with XPaths configured to the ACORD standard model.

#### Other Pertinent Information

##### APDataCollection

APDataCollection includes four additional view types; each view type pertaining to one of the ACORD supported views. Each of these views is available by taking the existing simplified ACORD XML corollary view and applying the appropriate transformation to it. Applications can make use of APDataCollection.changeViewType() accordingly.

The following table lists all existing and new view types, including their interpretations:

APDataCollection View Identifier	Description	Persisted to XMLSTORE Table
IXMLConstants.VIEW_CURRENT_DOCUMENT	Refers to the current Simplified ACORD XML XML document.	Yes
IXMLConstants.VIEW_ORIGINAL_DOCUMENT	Refers to the original Simplified ACORD XML XML document. Available on work items whose starting image came from an external source.	Yes
IXMLConstants.VIEW_MERGED_DOCUMENT	Refers to the previous Simplified ACORD XML XML document or the current document's previous commit point.	No
IXMLConstants.VIEW_MERGED_DOCUMENT	Refers to the Simplified ACORD XML document based on the comparison of two other view types.	No

<b>APDataCollection View Identifier</b>	<b>Description</b>	<b>Persisted to XMLSTORE Table</b>
IXMLConstants.VIEW_CURRENT_ACORD_DOCUMENT	Refers to the current ACORD compliant XML document. Derived by applying transformation against the document @ IXMLConstants.VIEW_CURRENT_DOCUMENT	No
IXMLConstants.VIEW_ORIGINAL_ACORD_DOCUMENT	Refers to the original ACORD compliant XML document. Derived by applying transformation against the document @ IXMLConstants.VIEW_ORIGINAL_DOCUMENT.	No
IXMLConstants.VIEW_PREVIOUS_ACORD_DOCUMENT	Refers to the previous ACORD compliant XML document. Derived by applying transformation against the document @ IXMLContants.VIEW_PREVIOUS_DOCUMENT.	No
IXMLConstants.VIEW_MERGED_ACORD_DOCUMENT	Refers to the merged ACORD compliant XML document. Derived by applying transformation against the document @ IXMLConstants.VIEW_MERGED_DOCUMENT.	No

You may need the ACORD standard version of your work item XML so that you can call a rating engine or pass to some other external service. The following code snippet would be appropriate:

```
private Document getACORDStandard(APDataCollection dataCollection) { // The call to change view type will engage the proper transformer
int restoreView = dataCollection.changeViewType(IXMLConstants.VIEW_CURRENT_ACORD_STANDARD_DOCUMENT); try { return
dataCollection.getDocument(IXMLConstants.VIEW_CURRENT_ACORD_STANDARD_DOCUMENT); } finally {
dataCollection.changeViewType(restoreView); } }
```

Be aware that APDataCollection.changeViewType(int viewType) will invoke the appropriate transformations on any of the IXMLConstants#VIEW\_xxxxx\_ACORD\_STANDARD\_DOCUMENT view types if one is not yet rendered.

The APDataCollection.getDocument(int viewType) method simply returns the standard document instance that is sitting there (which could be null). This allows the application to have full control on how often the transformations are actually engaged. In order to force a transformation on one of the standard documents, which could be out of date or stale, you must call APDataCollection.setDocument(null, IXMLConstatns#VIEW\_xxxxx\_ACORD\_STANDARD\_DOCUMENT, false) so that the next time the changeViewType method is called, it would indeed engage the transformations.

The APDataCollection class does not internally try to infer how often transformations should or should not be run.

Remember to always restore the view type within a finally block whenever you change the view type as illustrated above.

#### Debug Console

The debug console presents all of the various document views available for a work item. When one of the ACORD standard views is selected, the ACORD standard target format transformation is engaged. This provides an easy way to debug that particular set of transformations.

Refer to the [Debug Console](#) topic for more information.

## Sample Simplifications

### BlanketInfo

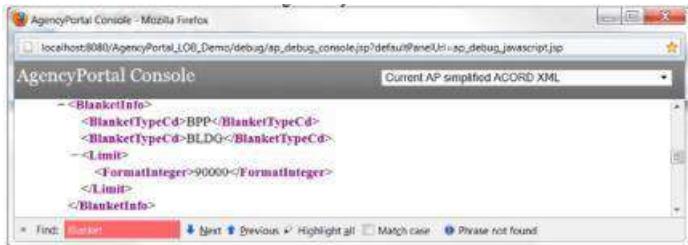
Under ACORD standard, the blanket info illustrates a case where the selected blanket type field not only determines the coverage code on `BlanketInfo`, but also has an impact on the number of underlying `BlanketTypeCd` aggregates you end up with.

Consider the following interface:

Under the ACORD standard, the resultant XML looks like the following when you select Building from the Blanket Type drop-down:

Under the ACORD standard, the resultant XML looks like the following when you select Business Personal Property from the Blanket Type drop-down:

Under ACORD standard, the resultant XML looks like the following when you select Both from the Blanket Type drop-down:



The ACORD standard model requires a special field helper to maintain the `BlanketTypeCd` value and the funky cardinality when BOTH is selected:

```
<pageElement id="blanket" type="fieldset" legend="Blanket"> <fieldElement type="selectlist"
id="SP.BOPLineBusiness.PropertyInfo.BlanketInfo[BlanketTypeCd='SELECTED']" label="Blanket Type" defaultValue="BOTH"> <optionList
reader="xmlreader" source="codeListRef.xml" target="blank"/> <optionList reader="xmlreader" source="bopCodeListRef.xml"
target="umbpropertytype"/></fieldElement> <fieldElement type="text"
id="SP.BOPLineBusiness.PropertyInfo.BlanketInfo[BlanketTypeCd='SELECTED'].Limit.FormatInteger" label="Blanket Limit" size="10"
maxLength="14"> <validation type="numeric" id="N177"/> <fieldHelper type="balloon">This must equal the sum of your blanket types for all
locations</fieldHelper> </fieldElement> </pageElement>
```

ACORD simplified model simplifies things greatly. Under this mode:

- We allow the value from the select list to map directly to the `BlanketTypeCd`, allowing the value of "BOTH" on `BlanketTypeCd` (even though it is not a valid coverage code).
- We get to remove the need for any special field helper under the covers as follows:

```
<pageElement id="blanket" type="fieldset" legend="Blanket"> <fieldElement type="selectlist"
id="BOPLineBusiness.PropertyInfo.BlanketInfo.BlancketTypeCd" label="Blanket Type" defaultValue="BOTH"
groupId="BOPLineBusiness.PropertyInfo.BlanketInfo"> <optionList reader="xmlreader" source="codeListRef.xml" target="blank"/>
<optionList reader="xmlreader" source="bopCodeListRef.xml" target="umbpropertytype"/></fieldElement> <fieldElement type="text"
id="BOPLineBusiness.PropertyInfo.BlanketInfo.Limit.FormatInteger" label="Blanket Limit" size="10" maxLength="14"> <validation
type="numeric" id="N177"/> <fieldHelper type="balloon">This must equal the sum of your blanket types for all locations</fieldHelper>
</fieldElement> </pageElement>
```

Under ACORD simplified, the resultant XML looks like the following when you select Both from the Blanket Type drop-down:



Now, you just need to write the transformations to/from the ACORD standard representation to deal with how BOTH is represented on both sides. The following is a Java based transformer to illustrate this use case:

```
/* *Created on Nov 10, 2011 by nbaker AgencyPort Insurance Services, Inc. */ package com.agencyport.bop.transformers; import
org.jdom.Element; import com.agencyport.domXML.APDataCollection; import
com.agencyport.domXML.transform.APDataCollectionTransformationException; import
com.agencyport.domXML.transform.IAPDataCollectionTransformer; import com.agencyport.domXML.transform.TargetFormat; /** * The
BlanketInfoTransformer class supports both the ACORD simplified and standard transformations of the BOP * BlanketInfo aggregate.* / public
class BlanketInfoTransformer implements IAPDataCollectionTransformer { private static final String BLANKET_TYPE_CD_XPATH =
"BOPLineBusiness.PropertyInfo.BlanketInfo.BlanketTypeCd"; private static final String BOTH = "BOTH"; private static final String
BUILDING = "BLDG"; private static final String BUILDING_PERSONAL_PROPERTY = "PP"; /** * Constructs an instance.* / public
BlanketInfoTransformer() {} /** * @param doc */ public void transform(APDataCollection dataCollection, TargetFormat targetFormat,
Element customParameters) throws APDataCollectionTransformationException { if (targetFormat.equals(TargetFormat.ACORDSimplified)) {
simplify(dataCollection); } else { standardize(dataCollection); } } /** * Simplifies the BlanketInfo aggregate.* @param dataCollection is the
XML to update.* @throws APDataCollectionTransformationException */ private void simplify(APDataCollection dataCollection) throws
APDataCollectionTransformationException { if (dataCollection.getCount(BLANKET_TYPE_CD_XPATH) > 1) {
dataCollection.setFieldValue(BLANKET_TYPE_CD_XPATH, BOTH); dataCollection.deleteElement(BLANKET_TYPE_CD_XPATH, 1); } }
/** * Standardizes the BlanketInfo aggregate.* @param dataCollection is the XML to update.* @throws
APDataCollectionTransformationException */ private void standardize(APDataCollection dataCollection) throws
```

```
APDataCollectionTransformationException { if (BOTH.equals(dataCollection.getFieldValue(BLANKET_TYPE_CD_XPATH, null))) {
 dataCollection.setFieldValue(BLANKET_TYPE_CD_XPATH, 0, BUILDING); dataCollection.setFieldValue(BLA NKET_TYPE_CD_XPATH,
1, BUILDING_PERSONAL_PROPERTY); } } }
```

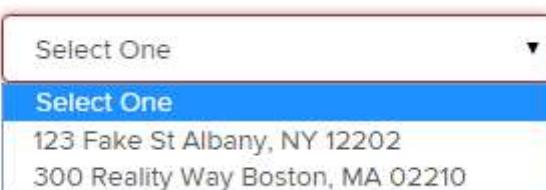
# Index Management

Many situations arise, especially in commercial lines, where the roster source defines an XML path with more than one repeating element in it. An index element type (<pageElement type="index" />) is available to the roster page definition, which provides a syntax that the framework would interpret for deriving the index management state. It can be used to tell the framework which appropriate aggregate to update for a roster entry when the ancestry of the roster source element has other repeaters that need to be predicated.

Consider the rating classifications roster page for a Workers Comp transaction. The following example indicates that the prior Locations page has already gathered the following location aggregates from the user:

```
<Location id="N70" saved="true">
 <ItemIdInfo id="N67"/>
 <Addr id="N68">
 <Addr1 id="N71">123 Fake St</Addr1>
 <City id="N72">Albany</City>
 <StateProvCd id="N69">NY</StateProvCd>
 <PostalCode id="N73">12202</PostalCode>
 <CountryCd id="N74">US</CountryCd>
 </Addr>
</Location>
<Location id="N79" saved="true">
 <ItemIdInfo id="N76"/>
 <Addr id="N77">
 <Addr1 id="N80">300 Reality Way</Addr1>
 <City id="N81">Boston</City>
 <StateProvCd id="N78">MA</StateProvCd>
 <PostalCode id="N82">02210</PostalCode>
 <CountryCd id="N83">US</CountryCd>
 </Addr>
</Location>
```

The ratingClassification page has a select list whose options consist of the locations previously entered by the user. This select list is populated using the dynamicListTemplate infrastructure; refer to the [Option Lists](#) topic for more information.



A screenshot of a user interface showing a dropdown menu. The menu is titled "Select One" and contains two items: "123 Fake St Albany, NY 12202" and "300 Reality Way Boston, MA 02210". The menu is displayed over a form field labeled "Location".

The data values for the options in the select list above are the values of the corresponding location aggregate's ID attributes (i.e., in this example, "N70" or "N79"). There is also a business rule in place that ensures that at least one rating classification roster entry is created for each location.

Let's say the user enters the following rating information values for the first rating classification entry:

\* Denotes Required Fields

Please list all rating classifications for all locations

### Rating Information

For Location	Class Code	Full Time Empl	Part Time Empl	Est. Annual Exposure
No Rating Information found for this work item				

### Rating Information

\* Location 300 Reality Way Boston, MA 02210 ▾

U.S.L & H. No ?

List of Class Codes displayed is based on the state present in the selected location

\* Class Code 0034 Farm: Poultry Or Egg Producer and Drivers ▾

Full Time Employees 2

Part Time Employees 1

Est. Annual Exposure

Voluntary Comp No ▾

**Add**

In this example, the roster source path for this Rating Classification page is set to the following value:

```
WorkCompLineBusiness.WorkCompRateState.WorkCompLocInfo.WorkCompRateClass
```

When the user clicks the Add button, the standard roster process would create a WorkCompRateState aggregate if one didn't already exist, create a WorkCompLocInfo if one didn't already exist and then add this and all subsequent WorkCompRateClass aggregates created by roster data entry to the first WorkCompLocInfo within the first WorkCompRateState.

When you consider the previous statement, you can infer that, without additional configuration, an index of 0 is assumed for any repeaters in the roster source aggregates ancestry. This means that the roster source path looks like the following in terms of how it's evaluated by the system:

```
WorkCompLineBusiness.WorkCompRateState[0].WorkCompLocInfo[0].WorkCompRateClass
```

From a data perspective, this simply would not work. After the user has satisfied the minimum requirement of creating at least one rating classification roster entry for each location, the following characteristics about the business data must be true to maintain compliance with the ACORD standard:

- Based on the two locations aggregates that exist, which are composed of one NY Location and one MA Location, the final XML will need to have two WorkCompRateState aggregates (one with a StateProvCd element of "NY", one with a StateProvCd element of "MA").
- The WorkCompLocInfo aggregates, of which there will be two in total, will need to be created underneath the appropriate WorkCompRateState aggregate.
- Each WorkCompLocInfo aggregate will have an attribute named locationRef, which is set to the ID of the related Location aggregate [i.e., the data value of the roster entry's location select list (in this example, "N70" or "N79")].
- If there are multiple rating classification entries that relate to the same location, then all WorkCompRateClass aggregates for that location should exist under a single WorkCompLocInfo aggregate.

Taken together, the rules above mean that the system needs a way to dynamically predicate the ancestor repeater's roster page's source path, based on the location selected by the user. We need a way to express to the framework that if the first location is selected by the user in the new rating classification, then the source path of the roster should evaluate to this:

```
WorkCompLineBusiness.WorkCompRateState[StateProvCd='NY'].WorkCompLocInfo[@LocationRef='N70'].WorkCompRateClass
```

Similarly, if the second location is selected by the user, the source path of the roster should evaluate to this:

```
WorkCompLineBusiness.WorkCompRateState[StateProvCd='MA'].WorkCompLocInfo[@LocationRef='N79'].WorkCompRateClass
```

Generally, when <LOCATION\_SELECTION> corresponds to the data value of the location select list and <LOCATION\_STATE> is resolved to be the StateProvCd element of the location aggregate with an ID matching the <LOCATION\_SELECTION>, the source path of the roster should evaluate to the following:

```
WorkCompLineBusiness.WorkCompRateState[StateProvCd='<LOCATION_STATE>'].WorkCompLocInfo[@LocationRef='<LOCATION_SELECTION>'].WorkCompRateClass
```

Additional configuration is provided by adding an index pageElement to the roster page. The index pageElement uses two attributes, which allows developers to configure the path substitution that is needed when deriving the predicates illustrated above.

- The index attribute is used to specify the predicates for the parent of the roster source. Substitution variables can be used to dynamically replace parts of the predicate with values from the APDataCollection or the HTMLDataContainer (a special prefix is used to support lookups from the HTMLDataContainer). This is a mandatory attribute when using the index pageElement. The path specified is comprised of the parent of the roster source path with predicates added for any repeaters in the path, which must be qualified.
  - In this example, this will be a path to WorkCompLocInfo with predicates added for WorkCompRateState and WorkCompLocInfo.
- The (optional) relatedIndex attribute is used to draw a relationship with and look up data from a foreign aggregate if such a relationship is needed when deriving the values for the substitute variables specified in the index path. The rule of thumb is that you will need a relatedIndex attribute if your index path contains a substitute value, which is resolved from some repeating element path outside of the current roster source path hierarchy.
  - In this example, the StateProvCd needs to be resolved from the one of the repeating location aggregates, which is outside of the current WorkCompRateClass path hierarchy. Since the index path will have a substitution token pointing to the path Location.Addr.StateProvCd, the relatedIndex is used to tell the framework which location aggregate to use when retrieving the state.

Determining the path for the relatedIndex attribute is trivial. The path to the repeating location aggregate is what we care about and we can target the appropriate location aggregate by predicated the ID attribute with the select value from the Location select list on the UI (this value can be retrieved from the HTMLDataContainer). In this example, the relatedIndex should be set to the following value:

```
Location[@id='${HTMLDC.WorkCompLineBusiness.WorkCompRateState.WorkCompLocInfo.@LocationRef}']
```

The \${} syntax is the standard token substitution notation that is used throughout the framework. HTMLDC is a special prefix that indicates that the rest of the token is a key that the framework should use to look up from the HTMLDataContainer object on request. The key corresponds to the Location select list's fieldElement's ID attribute (the uniqueId attribute must be used if one is present on the fieldElement).

After the relationship with the foreign aggregate has been defined using the relatedIndex attribute, the index attribute can now contain APDataCollection tokens, which refer to paths under the Location of the aggregate. In this example, the index attribute should be set to:

```
WorkCompLineBusiness.WorkCompRateState[StateProvCd='${Location.Addr.StateProvCd}'.WorkCompLocInfo[@LocationRef='${Location.@id}']]
```

In the case of both of the tokens above, the location part of the path will be predicated based on the relatedIndex that is configured. It's work noting that we could have used the HTMLDC to get the @LocationRef, but since Location.@id as illustrated above works just as well since we've already established the relationship to the location aggregate with the relatedIndex attribute.

All together, the index pageElement for this roster page would be configured in the TDF as follows:

```
<pageElement type="index" id="ratingInfoIndexId"
relatedIndex="Location[@id='${HTMLDC.WorkCompLineBusiness.WorkCompRateState.WorkCompLocInfo.@LocationRef}']"
index="WorkCompLineBusiness.WorkCompRateState[StateProvCd='${Location.Addr.StateProvCd}'.WorkCompLocInfo[@LocationRef='${Location.@id}']]"/>
```

With this configuration in place, the three Rating Classification roster entries are submitted as follows:

\* Denotes Required Fields

Please list all rating classifications for all locations

### Rating Information

For Location	Class Code	Full Time Empl	Part Time Empl	Est. Annual Exposure	
300 Reality Way Boston, MA 02210	0034	2	1		<a href="#">Edit</a> <a href="#">Delete</a>
300 Reality Way Boston, MA 02210	0036	3			<a href="#">Edit</a> <a href="#">Delete</a>
123 Fake St Albany, NY 12202	8227		1		<a href="#">Edit</a> <a href="#">Delete</a>

[Continue](#)

[Add New](#)

[More Actions...](#) ▾

The XML will also be generated appropriately as follows:

```

<WorkCompRateState id="N85">
 <StateProvCd id="N84">MA</StateProvCd>
 <SafetyGroupInd id="N91">0</SafetyGroupInd>
 <CreditOrSurcharge id="N87">
 <CreditSurchargeCd id="N86">AREM</CreditSurchargeCd>
 <NumericValue id="N88">
 <FormatModFactor id="N89">1.0</FormatModFactor>
 </NumericValue>
 </CreditOrSurcharge>
 <WorkCompLocInfo LocationRef="N79" id="N100">
 <WorkCompRateClass id="N101" saved="true">
 <NumEmployeesFullTime id="N105">2</NumEmployeesFullTime>
 <NumEmployeesPartTime id="N106">1</NumEmployeesPartTime>
 <RatingClassificationCd id="N103">0034</RatingClassificationCd>
 <RatingClassificationDesc id="N104">Farm: Poultry Or Egg Producer and Drivers</RatingClassificationDesc>
 </WorkCompRateClass>
 <WorkCompRateClass id="N107" saved="true">
 <NumEmployeesFullTime id="N111">3</NumEmployeesFullTime>
 <RatingClassificationCd id="N109">0036</RatingClassificationCd>
 <RatingClassificationDesc id="N110">Farm: Dairy and Drivers</RatingClassificationDesc>
 </WorkCompRateClass>
 </WorkCompLocInfo>
 <RiskId id="N90">2323</RiskId>
</WorkCompRateState>
<WorkCompRateState id="N93">
 <StateProvCd id="N92">NY</StateProvCd>
 <SafetyGroupInd id="N99">0</SafetyGroupInd>
 <CreditOrSurcharge id="N95">
 <CreditSurchargeCd id="N94">AREM</CreditSurchargeCd>
 <NumericValue id="N96">
 <FormatModFactor id="N97">1.0</FormatModFactor>
 </NumericValue>
 </CreditOrSurcharge>
 <WorkCompLocInfo LocationRef="N70" id="N112">
 <WorkCompRateClass id="N113" saved="true">
 <NumEmployeesPartTime id="N117">1</NumEmployeesPartTime>
 <RatingClassificationCd id="N115">8227</RatingClassificationCd>
 <RatingClassificationDesc id="N116">Permanent Yard</RatingClassificationDesc>
 </WorkCompRateClass>
 </WorkCompLocInfo>
 <RiskId id="N98">2323</RiskId>
</WorkCompRateState>

```

## Index Management Configuration

To support index state derivation at runtime, additional metadata is available in the roster page of a TDF. The following table describes the composition of this type index page element:

Entity	Usage/Interpretation	Data type/requiredness	Sample Value
pageElement. @type="index"	Identifies this page element for the purpose of configuring index state management		

Entity	Usage/Interpretation	Data type/requiredness	Sample Value
pageElement. @index	<p>Specifies the qualified XPath for deriving the parent index for this roster source.</p> <p>Substitution variables will be dynamically replaced from data from APDataCollection or HTMLDataContainer (or IntraPageDTRBehaviorManager in IPDTR). Page variables will be prefixed with a HTMLDC. prefix.</p>	<p>String - valid only if type is index.</p> <p>Required if type is index.</p>	WorkCompLineBusiness.WorkCompRateState[StateProvCd='\${Location.Addr.StateProvCd}'].WorkCompLocInfo[@LocationRef='\${Location.@id}']
pageElement. @relatedIndex	<p>Specifies the qualified XPath for deriving the index for the related foreign aggregate.</p> <p>Substitution variables will be dynamically replaced from data from APDataCollection or HTMLDataContainer (or IntraPageDTRBehaviorManager in IPDTR). Page variables will be prefixed with a HTMLDC. prefix.</p>	<p>String - valid only if type is index.</p> <p>Required only if the index attribute contains references to foreign aggregate data values.</p>	Location[@id='\${HTMLDC.WorkCompLineBusiness.WorkCompRateState.WorkCompLocInfo.@LocationRef}']

#### Note

You still need to establish the relatedIndex on the pageElement for foreign assets, even if a joinInfo to the same foreign aggregate is present.

# Change Management

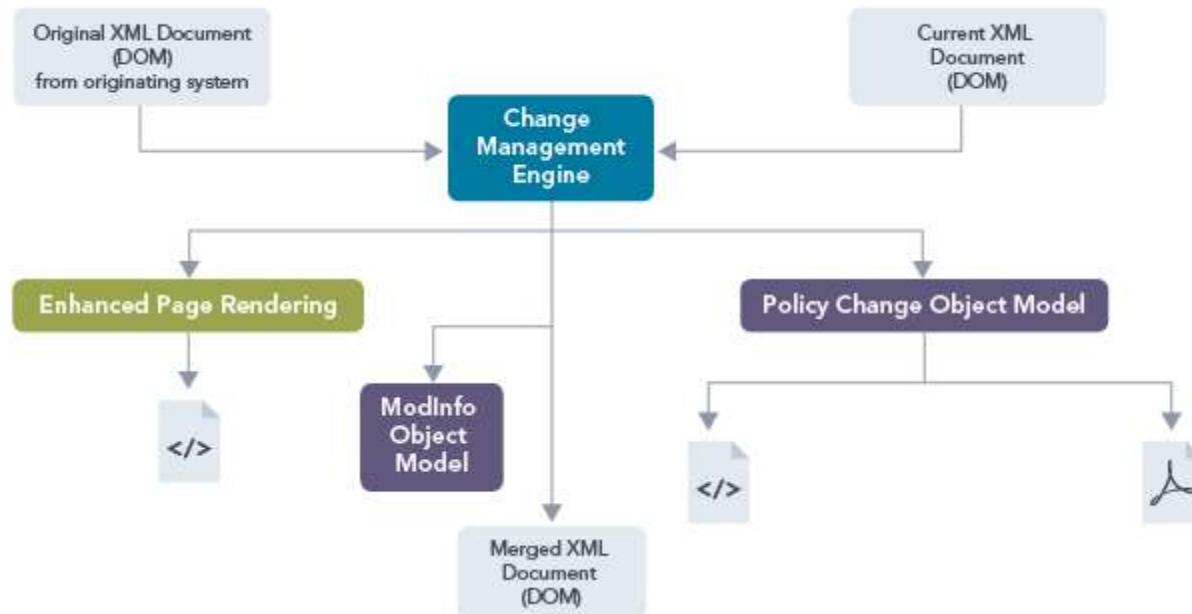
The change management engine is leveraged internally by the SDK in its support of policy change type transactions, such as endorsements or renewals, but also to support an enhanced upload/data bridging UI data collection experience. This topic provides a high level overview of the change management engine features. You may notice in the diagram below that the engine requires two XML documents to be available:

- The XML document represents a fixed point in time, which is typically a snapshot of the data at the point of introduction into the system
- The current XML document, which contains all of the user's updates

The following are the by products that this infrastructure creates:

- An enhanced data collection page rendering subsystem
- Data change reports with a raw XML data orientation spin:
  - A Java based reporting API object model (ModInfo object model as indicated in the diagram)
  - An ACORD based instance with ModInfo status aggregates (merged XML document as indicated in the diagram)
- Policy change summary object model, which is a Java object model that reports the changes using a TDF view (instead of a raw data orientation):
  - Policy change summarizer web page view
  - Policy change summarizer PDF view

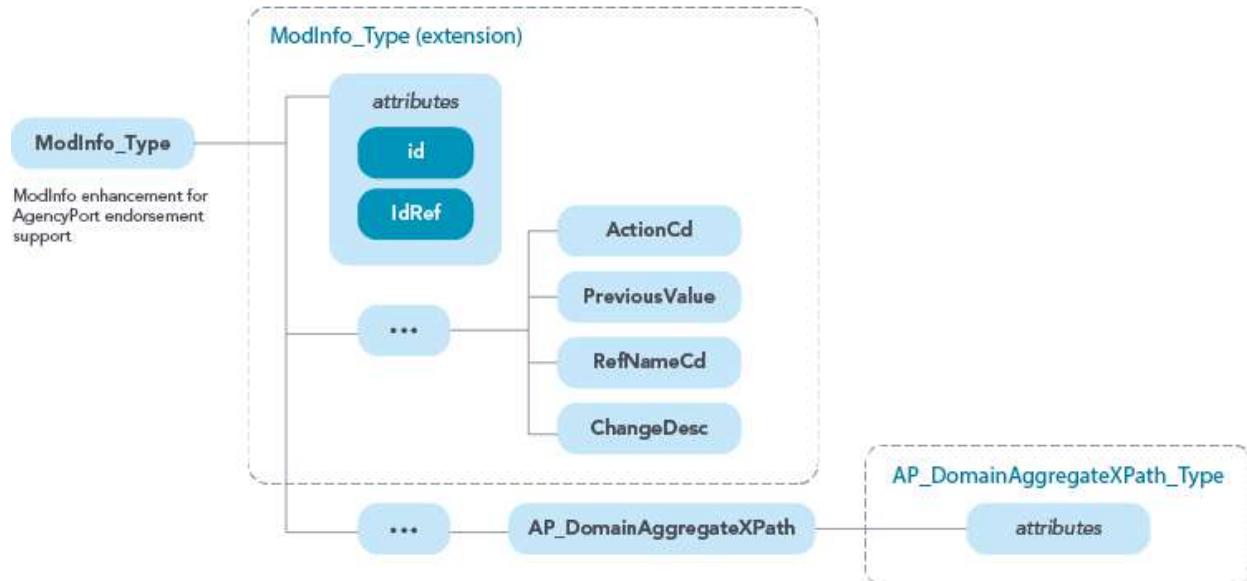
## Fundamentals



# ACORD ModInfo Review

The ModInfo construct is based on an ACORD data-centric artifact that provides a facility for distinguishing aggregate additions/deletions, changes and relationship changes via XML syntax. This aggregate contains what kind of action was applied to the data, what its previous value was (if the action changed the original data value) and a reference to the domain aggregate that underwent the action.

The following depicts how ACORD models the ModInfo aggregate:



# Merged Document (DOM)

The merged document (DOM) is comprised of:

- the current DOM with current values
- the aggregates in the original DOM, which have been deleted from the current document
- ModInfo ACORD aggregates for every addition, deletion, change (with previous value) and relationship change

The merged document is rendered on-demand and is not persisted to the `xmlstore` table.

The following `ModInfo` major actions apply:

- Addition? - XML aggregate appears in the current document, but not in the original document.
- Deletion? - XML aggregate appears in the original document, but not in the current document.
- Data Change? - When the same XML aggregate exists on both the current and original documents, but contains a data value that is different.
- Relationship Change? - When an aggregate is related to a different aggregate via an `xxxxxxxxRef` attribute value in the current document (compared to the original document).

Additions and deletions have a `ModInfo` entry for the highest relevant level in the document associated with the addition or deletion. XML children beneath a deleted or added parent have no `ModInfo` entries. Changes are tracked at the element level.

# ModInfo Action Use Cases

The following are ModInfo action use cases:

1. Add a new driver to a personal auto policy:

```
<ModInfo id="AB4B70D25E6A43F3D1D2AC782A6953B13A" IdRef="A2B89C0630C059B57E4BDDD0317D218FCA">
<ActionCd>A</ActionCd>
<AP_DomainAggregateXPath>PersAutoLineBusiness[@id='ACEEE8BA2988B71AD097CF616FFF472A1A'].PersDriver[@id='A2B8
9C0630C059B57E4BDDD0317D218FCA']</AP_DomainAggregateXPath> </ModInfo> <PersAutoLineBusiness
id="ACEEE8BA2988B71AD097CF616FFF472A1A"> <LOBCd
id="A1D801BE2F9CCEAFE041931616BEFFEE4A">AUTOP</LOBCd> <PersDriver
id="A2B89C0630C059B57E4BDDD0317D218FCA"/> </PersAutoLineBusiness>
```

2. Delete a garaging location on a personal auto policy:

```
<ModInfo id="A4EECFC1E45FF6E06FA2C1392C8A94986A" IdRef="A5291D6FD4E6DA5470861FFD4284BA993A">
<ActionCd>D</ActionCd>
<AP_DomainAggregateXPath>Location[@id='A5291D6FD4E6DA5470861FFD4284BA993A']</AP_DomainAggregateXPath>
</ModInfo> - <Location id="A5291D6FD4E6DA5470861FFD4284BA993A"> - <ItemInfo
id="A5730DA16CDA595EA66F17E33F7BB8223A"> <AgencyId
id="AAAF56C09C8FBA566DD02735349B52DF9A">0001</AgencyId> <ItemInfo> - <Addr
id="AA460AFD8D633033C0713DA43AD70D3A1A"> <AddrTypeCd
id="A7BA04A0C1E89B5D10E97CC2EFA3E8DB6A">GaragingAddress</AddrTypeCd> <Addr1
id="AAE1F5E43575748F616BECC20CFF539C8A">19 Pleasant St</Addr1> <City
id="A582D50240392C55534BEF8004653E86CA">Keene</City> <StateProvCd
id="AD9D20D81EC9DC73EB9BF3814F00207A">ME</StateProvCd> <PostalCode
id="AF1CD8ED18764B0F7BDE992B91629CA7FA">03431</PostalCode> </Addr> </Location>
```

3. Change a driver's middle name from "Will" to "Jeffrey" on a personal auto policy:

```
<ModInfo id="AA19F564B37C399FC0F85C14AAD42CBC3A" IdRef="AA06879DC376CD30DD058988741BC5D33A">
<ActionCd>M</ActionCd> <PreviousValue>Will</PreviousValue> <AP_DomainAggregateXPath>
PersAutoLineBusiness[@id='ACEEE8BA2988B71AD097CF616FFF472A1A'].
PersDriver[@id='AF0157E8B4EBBFED681A2351849EE957CA'].
GeneralPartyInfo[@id='A60C8442BF23072325E2A0BA831BD0197A'].
NameInfo[@id='A96163B622522D41CB51ADA3157F7EF62A'].
PersonName[@id='A8E73EF510B76FC75CD442A77FE1D9BD1A'].
OtherGivenName[@id='AA06879DC376CD30DD058988741BC5D33A'] </AP_DomainAggregateXPath> </ModInfo>
<PersAutoLineBusiness id="ACEEE8BA2988B71AD097CF616FFF472A1A"> <PersDriver
id="AF0157E8B4EBBFED681A2351849EE957CA"> <GeneralPartyInfo id="A60C8442BF23072325E2A0BA831BD0197A">
<NameInfo id="A96163B622522D41CB51ADA3157F7EF62A"> <PersonName
id="A8E73EF510B76FC75CD442A77FE1D9BD1A"> <OtherGivenName id="AA06879DC376CD30DD058988741BC5D33A">
Jeffrey </OtherGivenName> <PersonName> </NameInfo> </GeneralPartyInfo> </PersDriver> </PersAutoLineBusiness>
```

4. Change the garaging location on vehicle #1 on a personal auto policy:

```
<ModInfo id="A453D834A06E64A4CA75094F4B8DEF4F3A" IdRef="AB8BE62852A22AD6AAF546C819F85C3E3A">
<ActionCd>R</ActionCd> <RefNameCd
PreviousRefs="A5291D6FD4E6DA5470861FFD4284BA993A">LocationRef</RefNameCd>
<AP_DomainAggregateXPath>PersAutoLineBusiness[@id='ACEEE8BA2988B71AD097CF616FFF472A1A'].PersVeh[@id='AB8BE
62852A22AD6AAF546C819F85C3E3A']</AP_DomainAggregateXPath> </ModInfo> <Location
id="A5291D6FD4E6DA5470861FFD4284BA993A"> <Location id="A8208A5EA81EB89A07DBEA E4006F6A850A/"> -----
<PersAutoLineBusiness id="ACEEE8BA2988B71AD097CF616FFF472A1A"> <PersVeh
id="AB8BE62852A22AD6AAF546C819F85C3E3A" LocationRef="A8208A5EA81EB89A07DBEA E4006F6A850A"/>
```

# Diff/Merge Algorithm

The following are assumptions of diff algorithm:

- Each XML element will contain a key ID attribute value. Those with no ID attribute value will be ignored by diff.
- Keys are immutable once assigned to an element.
- Keys are conserved across views.
- Only attributes with names ending in 'Ref' are evaluated for relationship changes (e.g., 'LocationRef')

Diff is a two pass approach and generates the `ModInfo` object model as output:

- 1st pass detects additions and changes (including relationship changes) creating a `ModInfo` object instance for each modification.
- 2nd pass detects deletions, creating a `ModInfo` object instance for each deletion.

Action code derivation:

- What constitutes an addition? A key value (ID attribute value) that is present in the current document, but not present in the original.
- What constitutes a deletion? A key value that is present in the original document, but not present in the current.
- What constitutes a change? When the same key value is present on both the original and current elements, but the element contains data values that are different.
- What constitutes a relationship change? When the same key value is present on both the original and current elements, but the `xxxxxRef` attribute values are different.

Generation of merged document:

- Merged document initialized first to a copy of the current document into a new one.
- Splices aggregates that are present in the original document, but not the current document into the merged document (the aggregates that were deleted from the current).
- Adds a `ModInfo` XML aggregate into the merged document for each and every instance in the related `ModInfo` object model.

# Change Management API

The Change Management API provides access to various APDataCollection views via the changeView Type method:

- VIEW\_CURRENT\_DOCUMENT - mutable, persisted
- VIEW\_ORIGINAL\_DOCUMENT - immutable, persisted
- VIEW\_MERGED\_DOCUMENT - rendered on demand, not persisted

Mentionable Java classes:

- MergeManager - responsible for generating ModInfo instances and the merged DOM
- ModInfo - contains core modification information for a data entity
- ModInfoGroup - abstraction layer over the core ModInfo class that is used to select/filter specific types or category groups of ModInfos
- PolicyChangeSummarizer - provides a user-friendly, TDF-based report on what has changed for a given XML instance

AXE API enhances the usability of the ACORD ModInfo data:

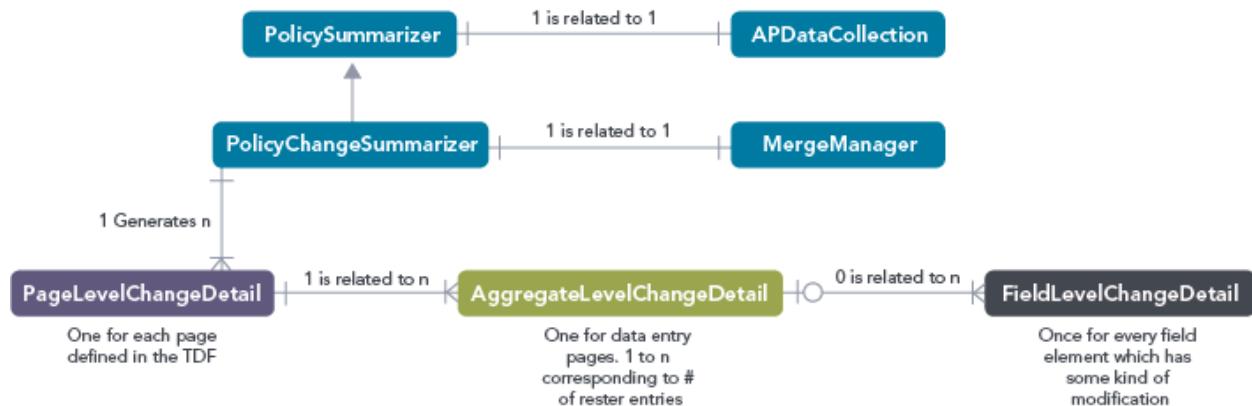
- Access to all ModInfo data via the ModInfo Java class
- Filtering ModInfo data by both action code and/or aggregate name via the MergeManager Java class
- Grouping ModInfo data by aggregate name via the ModInfoGroup Java class

Index management:

- Contains changes to support a more complexity incurred by the deletion of original aggregate possibility.
- ViewTypeResolver class useful for normalizing index arrays across views
- IndexManager class enhanced to accept view type parameters

The following diagram is an object view of the policy change object model. The policy change object model is a TDF-centric view of changes (versus a raw data-centric view reported by the core ModInfo class)

**Change Management API: Policy Change Summary Object Model**



# Agencyport XML Engine (AXE) API

This section provides assistance to an application programmer who wants to render various outputs associated with policy change transactions, as well as information on how to use the Agencyport XML Engine (AXE) API to achieve a desired rendering. Also included is a set of best practices for AXE programming techniques in the context of policy change management.

Although this documentation is specific to personalauto, you can also apply these techniques to any endorsement based PDF or for other specific view types (e.g., policy change summary view).

## Important

An alternate PDF format was adopted for the ACORD template applications team. This format does not conform to the ACORD PDF policy change forms. The format embraced by the ACORD template team is closely symmetric with the output format associated with the SDK's own policy change summarizer object model.

The object model exposed by this infrastructure is closely aligned with a portal application's own transaction definition (TDF) and, therefore, the rendered PDF format conforms closely to the pages that the user is already familiar with. However, even for endorsement application groups that choose to adopt this more straight-forward rendering style, we suggest you read through this section since it provides a conceptual and more detailed understanding of the AXE API that supports all Agencyport endorsement applications, regardless of the PDF rendering technique embraced.

## Note

This information is targeted to readers who have a detailed understanding of Agencyport's endorsement support module and have reviewed and understood the content of such specification. The information in this section is not a detailed specification on the rendering of all the fields in an endorsement, but rather establishes the techniques for using the AXE API to achieve such renderings.

We recommend you review the test JUNIT test cases found in the ap\_sdk\apbasetest\com.agencyport.domXML package, which are the realized implementation of the test plan for the AXE portion of the Agencyport endorsement support module.

The physical order of repeating aggregates in a resultant merged document view for a given APDataCollection instance should not be construed as an agent's or underwriter's understanding of an aggregate's "identity." For instance, the personalauto "dropping one vehicle" and "adding another" scenario.

## Example

You have a personal auto policy currently with three vehicles on it before any changes are applied. The first vehicle is dropped and a new vehicle is added with a net result of three vehicles still on the policy. The following conclusions can be made:

In the original document view, the vehicles are in the order in which they were acquired from the carrier PAS.

Physical Sequence of Vehicles in XML	Implied Vehicle Identity
1	1 (vehicle to be dropped from policy)
2	2
3	3

In the current document view (after all changes are applied), the vehicle aggregates display as follows:

Physical Sequence of Vehicles in XML	Implied Vehicle Identity
1	2
2	3

<b>Physical Sequence of Vehicles in XML</b>	<b>Implied Vehicle Identity</b>
3	4 (vehicle added to policy)

In the merged document view, the vehicles display as follows:

<b>Physical Sequence of Vehicles in XML</b>	<b>Implied Vehicle Identity</b>
1	2
2	3
3	4 (vehicle added to policy)
4	1 (vehicle dropped from policy)

The AXE com.agencyport.domXML.changemanagement.ModInfoGroup class addresses this identity concern and sorts the result sets of ModInfoGroup in implied aggregate identity order rather than their respective order in the merged document. The implied aggregate identity is not inherently related to the ItemIdInfo.AgencyId technique used by some application instances.

# Rendering AgencyPort Policy Change Summary Format

This section provides a use case illustrating the policy change summary infrastructure. It is important to note that some of the basic methods exposed can and should be used by PDF rendering processors as well. Review the Java doc in detail on both the `com.agencyport.policysummary.PolicySummarizer` and `com.agencyport.policysummary.changemanagement.PolicyChangeSummarizer` classes for helper methods around entities, such as insured name, mailing address, etc.

## PersonalAutoPolicyChangeSummarizer

The `PersonalAutoPolicyChangeSummarizer` class derives from `com.agencyport.policysummary.changemanagement.PolicyChangeSummarizer`. It alters the framework's `PolicyChangeSummarizer` base class default functionality as follows:

- Changes the description for both the driver and vehicle aggregates to include dynamic data rather than the default description eye catcher.
- Overrides the action code normalization for the vehicle underinsured coverage to provide the Deleted/Added action code when appropriate.

### Example

```
/* * Created on Feb 17, 2006 by norm AgencyPort Insurance Services, Inc. */ package com.agencyport.servlets.persauto.endorse; import
java.text.DateFormat; import java.util.Iterator; import com.agencyport.domXML.APDataCollection; import
com.agencyport.domXML.ElementPathExpression; import com.agencyport.domXML.IXMLConstants; import
com.agencyport.domXML.changemanagement.MergeManager; import com.agencyport.domXML.changemanagement.ModInfoGroup; import
com.agencyport.policysummary.changemanagement.PolicyChangeSummarizer; public class PersonalAutoPolicyChangeSummarizer extends
PolicyChangeSummarizer { private static final ElementPathExpression driverBaseXPath = new
ElementPathExpression("PersAutoLineBusiness.PersDriver"); static protected final FormatDataBlock[] vehicleElements = { new
FormatDataBlock("ModelYear", " ", null, new FormatDataBlock("Manufacturer", " ", "xmlreader:codeListRef.xml:vehicleMake"), new
FormatDataBlock("Model", " ", null) }; public PersonalAutoPolicyChangeSummarizer(MergeManager mergeManager, APDataCollection
apData, String defaultValueWhenNotPresent, DateFormat defaultDisplayDateFormat, DateFormat defaultXMLStorageDateFormat, String
policyAggregateName) { super(mergeManager, apData, defaultValueWhenNotPresent, defaultDisplayDateFormat,
defaultXMLStorageDateFormat, policyAggregateName); } public ModInfoGroup[] getDriverModInfoGroups() { return
ModInfoGroup.selectModInfoGroups(mergeManager, driverBaseXPath.getExpressionWithOnlyElementHierarchy()); } public String
getDriverName(ModInfoGroup driverGroup) { String xPathForThisDriverName =
ElementPathExpression.buildNextElementPath(driverGroup.getAggregateElementPath()).getExpressionWithAllPredicates(),
"GeneralPartyInfo.NameInfo"); int save = IXMLConstants.VIEW_CURRENT_DOCUMENT; if (driverGroup.getAggregateActionCode() ==
IXMLConstants.DELETE_ACTION) { save = apData.changeViewType(IXMLConstants.VIEW_ORIGINAL_DOCUMENT); } String
driverName = stringArrayAsString(buildDataBlock(xPathForThisDriverName, null, personalNameElements)); apData.changeViewType(save);
return driverName; } public String getVehicleDescription(ModInfoGroup vehGroup) { int save =
IXMLConstants.VIEW_CURRENT_DOCUMENT; if (vehGroup.getAggregateActionCode() == IXMLConstants.DELETE_ACTION) { save =
apData.changeViewType(IXMLConstants.VIEW_ORIGINAL_DOCUMENT); } String vehicleDescription =
stringArrayAsString(buildDataBlock(vehGroup.getAggregateElementPath().getExpressionWithAllPredicates(), null, vehicleElements));
apData.changeViewType(save); return vehicleDescription; } public String getAggregateDescription(ModInfoGroup group) {
ElementPathExpression groupElementPath = group.getAggregateElementPath(); int lastElementIx =
groupElementPath.elementCountInHierarchy(); Iterator it = groupElementPath.getElementHierarchyIterator(lastElementIx - 1); String
lastElementName = (String) it.next(); if (lastElementName.equals("PersDriver")) { return getDriverName(group); } else if
(lastElementName.equals("Pers Veh")) { return getVehicleDescription(group); } return super.getAggregateDescription(group); } /** This
accommodates the potential delete / add scenario for the optional underinsured * motorist coverage. Since this coverage is optional and coverves
repeat when a user deletes the coverage (by selecting * 'No Coverage Selected'), commits page, revisits and readds then this situation occurs.*
This delegates to the super class for 99% of functionality. * @see
com.agencyport.policysummary.changemanagement.PolicyChangeSummarizer#normalizeModInfoActionCode(boolean,
com.agencyport.html.elements.BaseElement, int[], int[], int) */ protected int normalizeModInfoActionCode(boolean
isContainedInARepeatingAggregate, BaseElement baseElement, int[] originalIdArray, int[] currentIdArray, int actionDerivedFromValues) {
String fieldId = baseElement.getId(); if (fieldId.equals("PersAutoLineBusiness.PersVeh.Coverage.Underinsured")) { int normalizedActionCode
= normalizeModInfoActionCode(isContainedInARepeatingAggregate, "PersAutoLineBusiness.PersVeh.Coverage[CoverageCd='UNDSP']",
originalIdArray, currentIdArray, actionDerivedFromValues); return normalizedActionCode; } return
super.normalizeModInfoActionCode(isContainedInARepeatingAggregate, baseElement, originalIdArray, currentIdArray,
actionDerivedFromValues); } }
```

(missing or bad snippet)

## CMDDisplaypolicyChangeSummary

The CMDDisplaypolicyChangeSummary class is the APCommand override for the policy change summary page. Its job is to create the policy change summarizer it's derived from. Put the following reference of this into a place where the custom JSP page can get to and be done:

```
package com.agencyport.servlets.persauto.endorse; import com.agencyport.data.DataManager; import
com.agencyport.domXML.APDataCollection; import com.agencyport.policysummary.changemanagement.PolicyChangeSummarizer; import
com.agencyport.shared.APException; /* * The CMDDisplaypolicyChangeSummary class is an example on how to prepare a * policy change
summary object. */ public class CMDDisplaypolicyChangeSummary extends
com.agencyport.servlets.shared.endorse.CMDDisplaypolicyChangeSummary { /* (non-Javadoc) * @see
com.agencyport.servlets.shared.endorse.CMDDisplaypolicyChangeSummary#
getSummarizer(com.agencyport.domXML.APDataCollection,
com.agencyport.data.DataManager) */ protected PolicyChangeSummarizer getSummarizer(APDataCollection apData, DataManager
dataManager) throws APException { return new PersonalAutoPolicyChangeSummarizer(apData.getMergeManager(), dataManager, null, null,
null, "PersPolicy"); } }
```

## persautoEndorse/policyChangeSummary.jsp

The persautoEndorse/policyChangeSummary.jsp uses the PersonalAutoPolicyChangeSummarizer and inherited behavior to render the policy change summary page, including the use of the change detail object model. The following JSP can be found in the ACORD template application repository:

### Note

This particular rendition is really not specific to personal auto endorsements except for the Java packaging and the casting of the PolicyChangeSummarizer instance. It is anticipated that this code can and will be generalized to serve up multiple LOB policy changes.

```
<!--begin endorse/policyChangeSummary --> <!--BEGIN STANDARD FRAMEWORK --> <%@include file="..//policychange/pageSetup.jsp"
%> <%@include file="..//policychange/style.jsp" %> <div id="pageBody" class="pageBodyWithSubmissionNavigation"><jsp:include
page="..//shared/messageTemplate.jsp" flush="true" /><!-- Note that htmlPage, hasChanges, and processMethod values are rendered by the
pageSetup.jsp include above --> <form action="FrontServlet" id="<% =htmlPage.getId()%>" name="<% =htmlPage.getId()%>" method="post">
<input type="hidden" name="NEXT" id="NEXT" value="" /> <input type="HIDDEN" name="PAGE_NAME"
value="<% =htmlPage.getId()%>" /> <input type="HIDDEN" name="METHOD" value="<% =processMethod%>" /><!-- Propagate the page title
from the TDF --> <h3 class="pageTitle"><% =htmlPage.getTitle()%></h3> <!-- Include the standard tips --> <jsp:include
page="..//policychange/standardTips.jsp" flush="true" /> <h3>Policy Information</h3> <table summary="Submission Summary"
class="summary"> <jsp:include page="..//policychange/polInfo.jsp" flush="true" /><!-- If you have a commercial lob then you will need to pass a
true value for the following jsp param. --> <jsp:param name="IS_COMMERCIAL" value="false" /> </jsp:include> <% if(hasChanges) {>%>
<tr> <td>Policy Change Form</td> <td>=<%=su
mmary.getItemId()%>&rnd=<%=Math.random()%>" target="new_window" title="Change Request PDF"
style="display: block; background:white url(themes/agencyportal/adobe_pdf.gif) no-repeat top left; height: 35px; width: 35px;" /></td> </tr> <%>
<!-- If you want to add more table rows just add as many tr entries as you want here. --> </table> <jsp:include
page="..//policychange/changeRequests.jsp" flush="true" /><!-- You can alter the title of this section with the following jsp:param. If you need to
change the style then change style.jsp. The following value is the default which is used if not passed. --> <jsp:param
name="CHANGE_REQUEST_TITLE" value="Change Requests Summarized by Screen" /> </jsp:include> <% if(hasChanges) {>%> <!-- This
depends on a Java script function 'processChangeRequestAction()' which is declared in pageSetup.jsp --> <h3>Actions</h3> <p style="font-
weight: bold">This endorsement request has not yet been submitted to the carrier. Click the "submit" button below to do so now.</p> <div
class="buttons"> <input type="button" name="submitButton" value="Submit" onclick="processChangeRequestAction(this.value);"/> <select
class="secondaryActions" onchange="processChangeRequestAction(this.value);> <option selected="" value="">More Actions...</option>
<option value="Save and Exit">Save and Exit</option> <option value="Cancel">Cancel</option> <option value="Undo All Changes">Undo
All Changes</option> </select> </div> <% } %> <jsp:include page="..//shared/control_data.jsp" flush="true" /> </form> </div>
```

## Policy Change PDF Manager

The policy change PDF manager uses the PolicyChangeSummarizer for all 100% of all data retrieval, including the TDF centric change management object model (PageLevelChangeDetail, AggregateLevelChangeDetail and FieldLevelChangeDetail classes). Since this rendered format is so close to the web page format rendered by the JSP in the previous section, a huge reuse opportunity is taken advantage of by the fact that both the web page "view" and PDF "view" both share the same PolicyChangeSummarizer component.

### Note

This particular rendition really is not specific to personal auto endorsements except for the Java packaging and the casting of the PolicyChangeSummarizer instance. It is anticipated that this code can and will be generalized to serve up multiple LOB policy changes.



```

1)); if (fieldChanges[j].getOriginalDataValue() != null) printValue(fieldChanges[j].getOriginalDisplayValue(), "Value.Original." + (pdfIndex + 1));
printValue(fieldChanges[j].getCurrentDisplayValue(), "Value.Current." + (pdfIndex + 1));
pdfDocument.printField(fieldChanges[j].getModInfoActionCodeString(), "Value.Action." + (pdfIndex + 1); pdfIndex++; } } } return
pageIndex; } private String getLimitedLabelText(String Value, String field) { FieldDefinition fieldDef = pdfDocument.getFieldDefinition(field);
int maxHeight = fieldDef.getMaxHeight(); int maxWidth = fieldDef.getMaxWidth(); int maxLength = maxHeight * maxWidth; if
(Value.length() > maxLength) return Value.substring(0, maxLength - 15) + "..."; return Value; } private void printValue(String value, String
field) { PdfBox box = pdfDocument.getBox(field); box.addText(value); box.print(); } private void writeHeaderFooter(int pageIndex, String
sectionName, boolean pageContinuation) { String value = summary.getWorkItemId(); pdfDocument.printField(value, "ReferenceNumber");
value = summary.getPolicyNumber(); if (value != null) pdfDocument.printField(value, "PolicyNumber"); value =
summary.getTransactionEffectiveDate(); if (value != null) pdfDocument.printField(value, "DateOfChange"); value =
summary.getPolicyEffectiveDate(); if (value != null) pdfDocument.printField(value, "EffectiveDate"); value =
summary.getPolicyExpirationDate(); if (value != null) pdfDocument.printField(value, "ExpirationDate"); // Insured writeInsured(); // Producer
writeProducer(); if (pageContinuation) sectionName += " (Continued)"; pdfDocument.printField(sectionName, "Section.Heading");
pdfDocument.printField(new Integer(pageIndex).toString(), "PageNumber"); } private void writeInsured() { String name =
summary.getInsuredName(XMLConstants.VIEW_CURRENT_DOCUMENT, false); if (name != null) pdfDocument.printField(name,
"Insured.Info1"); String[] address = summary.getInsuredMailingAddress(XMLConstants.VIEW_CURRENT_DOCUMENT); for (int i = 0; i <
address.length; i++) { pdfDocument.printField(address[i], "Insured.Info" + (i + 2)); } } private void writeProducer() { String name =
summary.getProducerName(XMLConstants.VIEW_CURRENT_DOCUMENT); if (name != null) pdfDocument.printField(name,
"Producer.Info1"); String[] address = summary.getProducerAddress(XMLConstants.VIEW_CURRENT_DOCUMENT); for (int i = 0; i <
address.length; i++) { pdfDocument.printField(address[i], "Producer.Info" + (i + 2)); } }

```

# AXE Best Practices

This section indicates the AXE best practices around programming policy change applications. It highlights a number of concerns and techniques that portal Java developers should be cognizant of when coding a portal application that supports policy change.

We recommend you read this entire section even if your application is not required to render ACORD policy change PDF forms.

## Delete/Add versus Change

The following scenario pertains typically to fields marked in the TDF as required = "false" or to roster items with the ACORD data element marked as repeating (`maxOcc="unbounded"`).

The `ModInfo` change management diffing algorithm internal to AXE assumes a conservation of the ID attribute values spanning the before and after image XML documents for a change action to be reported. This has an implication on whether delete/add or change (or none at all) `ModInfos` are created. This has an impact on the resultant merged document and potentially on downstream systems that would consume such merged document. The Java developer needs to understand this and how an application XML update algorithm can ultimately affect the resultant `ModInfos` that are generated.

There may be custom-coded algorithms that adopt a simple delete all and then re-add approach.

### Note

Such techniques can create a plethora of delete and add `ModInfos` for all of the aggregates deleted and re-added. This may or may not be the desired result.

There are times when a delete/add approach reflects a "logical change" in the user's mind.

### Example

Coverages - If a coverage was originally on a personal auto policy and, as part of the endorsement data collection request, the agent deleted the coverage and then re-added it with a different limit (or even the same limit), the `ModInfos` generated will reflect the delete and add actions; however, in the user's mind, they either changed it (or restored it).

In the SDK, the view and empty aggregate features pertaining to optimal repeating data may result in a delete/add `ModInfo` report when, in the mind of the user, a change was intended.

When a user adds or deletes an entry on a portal roster page, the notion of add and delete are fairly straight forward. However, there are many data elements that can be deleted implicitly by the framework via views, the empty aggregate feature or custom code.

### Example

If a user deleted an optimal coverage with a \$1000.00 deductible, committed the request, reopened the request, navigated to the pertinent data collection page and then re-added that coverage with a deductible of \$2500.00 all within the same endorsement request, the user thinks that they basically increased the coverage's deductible from \$1000.00 to \$2500.00. These scenarios are reported, by default, as "Deleted/Added" by the policy change summarizer infrastructure.

The following `ModInfo` report from the axe-31-pos JUNIT test case provides a contrived use case around the deleted/added versus change scenario. In this case, the BI coverage was originally on the PAS image (ID attribute value of '`A0621C3E549DEF44B2F4691012F37AE36A`'). The user visited the vehicle page and de-selected this BI coverage, committed the change and then revisited the page and re-added the BI coverage (this time with an attribute value of '`A0621C3E549DEF44B2F4691012F37AE36A`'). Assume that the BI coverage is optimal in this scenario.

```
- <ModInfo id="AC6101246F8E8AF0A1D926F602A88BE6EA" IdRef="A31B15022E8AA432AA924517D06936B8CA"> <ActionCd id="A5A95A45FAA307CE60BC612F21715CC0A">A</ActionCd> <AP_DomainAggregateXPath id="A44B7627195316A48065C6F9FB900A9E6A">PersAutoLineBusiness[@id='AD7ADCEF4154551ADAEE0034369FCC6E9A'].PersVeh[@id='IDA64776E4D499C657E0025B24B333DC4BCA'].Coverage[@id='A31B15022E8AA432AA924517D06936B8CA']</AP_DomainAggregateXPath></ModInfo> - <ModInfo id="A21BBCF9930968C1E195A125B71236F90A" IdRef="A0621C3E549DEF44B2F4691012F37AE36A"> <ActionCd id="A2E4B47A77A22E4C5EC9CD7195795E7E0A">D</ActionCd> <AP_DomainAggregateXPath id="A18CA17B5E07D529E735FF5D9FE53357DA">PersAutoLineBusiness[@id='AD7ADCEF4154551ADAEE0034369FCC6E9A'].PersVeh[@id='IDA64776E4D499C657E0025B24B333DC4BCA'].Coverage[@id='A0621C3E549DEF44B2F4691012F37AE36A']</AP_DomainAggregateXPath></ModInfo> - <Coverage id="A31B15022E8AA432AA924517D06936B8CA"> <CoverageCd id="A8CFBE129DC453F1708E4D978DABB87D4A">BI</CoverageCd> - <Limit id="AFE88E4D9B2538BA2EB36697460D39047A"> - <FormatCurrencyAmt id="A36D47ACD7E1CAED3CBFC6BBB95934770A"> <Amt id="A46FAC3276095EAEF138DAF2198245165A">200000</Amt> </FormatCurrencyAmt> </Limit> - <Limit
```

```

id="A0F6C31D646C377199BB31BB9456FA273A"> - <FormatCurrencyAmt id="ABD33882AC34049959F8A40538B9A14A2A"> <Amt
id="A65A31C8A61FC4BC177DCFCBAE4030791A">400000</Amt> </FormatCurrencyAmt> </Limit> </Coverage> - <Coverage
id="A0621C3E549DEF44B2F4691012F37AE36A"> <CoverageCd id="A026D1CB226DFE08BBFCC8D8E81DF4552A">BI</CoverageCd>
- <Limit id="AF4B4F1E68922899D3AE63F0DDE1A9DB0A"> - <FormatCurrencyAmt id="AA92F1EEE8C6D01E4488BF1361C6C63A3A">
<Amt id="A90FFD630BDC64075711D887EE1FAC3DEA">100000</Amt> </FormatCurrencyAmt> </Limit> - <Limit
id="A210715F1EB625EAAB359E1E820C1A58A"> - <FormatCurrencyAmt id="A03960E1ABFD5C8F7B1256FD1759C6238A"> <Amt
id="AAE2240F96284388B7E4D11B2E94E1858A">300000</Amt> </FormatCurrencyAmt> </Limit> </Coverage>
```

## Note

The default action rendered by the `PolicyChangeSummarizer` for this scenario is Delete/Added and not Changed. Furthermore, it is important to note that even if the user "restored" the values back to the original, the Deleted/Added action is still rendered by default.

## Deleted/Added Guiding Principles

Regardless of the cause, the notion of the "delete/add" scenario is a reality. Trying to minimize it should be a goal; trying to eradicate it totally is fruitless and not cost effective.

### Conserve ID Attribute Values

Review the design of the `APDataCollection` update procedures carried in your special field helpers or other custom handling code. Techniques that delete all related aggregates and then re-add the relevant ones will typically not result in the conservation of aggregates. After an aggregate's ID attribute is thrown away [via an `APDataCollection.deleteElement()`], the only way to get it back is to refer to the original document and get the ID attribute back via custom code and overwrite the ID attribute value.

### Example

An example of custom code that preserves ID attributes (rather than adopting the simple delete all and then re-add algorithm) is the `com.agencyport.custom.persauto.OtherStateLicenseFieldHelper` special field helper. Review this code to get some insight on how to conserve ID attributes.

## Note

The AXE framework automatically conserves ID attribute values for scalar aggregates (non-repeating aggregates) only. It is the fields that are optional and that repeat (e.g., optional coverages), which can cause this delete/add scenario.

### Action Code Rendering by the Policy Change Summarizer Infrastructure

By default, this infrastructure reports delete/add scenarios as "Deleted/Added" actions instead of a "Changed" action. There are several contributing factors that are weighted to make this determination. The framework uses both the comparison of the original and current values, as well as the generated `ModInfo` aggregates together, to determine the rendered action code. In most cases, these match; however, in some rare cases, they do not.

The one scenario that most often doesn't match is the classic Deleted/Added scenario. This rendered action code can be overridden via the derived form by extending the `PolicyChangeSummarizer.normalizeModInfoActionCode()` method. Refer to the `PersonalAutoPolicyChangeSummarizer` example in the [Rendering Agencyport Policy Change Summary Format](#) section. In this example, the TDF field ID is not a real XPath value, so the framework cannot determine the related `ModInfo` instances associated with this field since a view hides the underlying field IDs. Refer to the following guiding principles:

- Use real XPaths for field IDs where possible on repeating, optional fields with associated views. For fields where SDK views are applied, the page display and data processing portions of the SDK engine allow the TDF author to use any unique tag string for the field ID. In the case where applications use the policy change summarizer infrastructure, the use of a real XPath expression enables the framework to navigate to the applicable `ModInfo` aggregates without the aid of custom code. If the field ID cannot be the real XPath expression, or the view is handling some split type of logic across multiple XPaths, then an override of the `PolicyChangeSummarizer.normalizeModInfoActionCode()` method will be required to render the correct action code in some situations.
- For fields where custom special field helpers are applied, the page display and data processing portions of the SDK engine require the TDF author to use a unique tag (preferably prefixed with 'SP') for the field ID. The framework cannot use this tag directly as an XPath expression to fully determine the action code since it cannot navigate to the `ModInfo`s with this tag. For the policy change summarizer to ascertain the action codes relating to associated `ModInfo` aggregates, the application will be required to override `PolicyChangeSummarizer.normalizeModInfoActionCode()` to render the correct action code in some situations.

## Support for Complex Roster Processing

For LOBs with repeaters within repeaters, there are situations when the action of adding to a roster does not always result in the addition of a physical new aggregate in the XML, as specified by the roster source in the TDF. Furthermore, a delete action in those cases does not always physically delete an aggregate. To further complicate this, a recovery does not necessarily recover the roster source aggregate back from the original document. An example of this occurs on the Additional Policy Coverages roster page in the CommAuto template app.

Although the `CommLineBusiness.CommlRateState` aggregate specification is declared in the roster source of the TDF, this is very deceiving. When the user clicks Add or Delete on this page, the default SDK roster processing implementation is "overridden" by application custom code. What truly composes the notion of an additional policy coverage entry on this roster is not governed by the presence of XML aggregates matching the roster source TDF declaration, but through the examination of one or more sub fields within the `CommlRateState` hierarchy.

The following are steps application groups can follow to engage custom aggregate level action derivation for these roster situations, which are not so straight forward:

- Extend the SDK's `PolicyChangeSummarizer` class and override the `normalizeAggregateLevelChangeDetail()` method as appropriate. This same policy change summarizer derived class needs to be engaged by both the roster display function and the policy change summary function.
- Extend the SDK's `RosterReader` class and override the `createPolicyChangeSummarizer` method to create your own policy change summarizer class and make it known to the roster reader.
- Make sure that this same policy change summarizer class is engaged by the policy change summary display servlet.

### Example

The following is an example of a commercial auto policy change summarizer implementation that addresses the additional policy coverage roster complexity:

```
/* * Created on Feb 17, 2006 by norm AgencyPort Insurance Services, Inc. */ package com.agencyport.servlets.commlAuto.endorse; import com.agencyport.custom.commlAuto.AdditionalPolicyCoverageHelper; import com.agencyport.data.DataManager; import com.agencyport.data.IndexManager; import com.agencyport.domXML.ElementPathExpression; import com.agencyport.domXML.IXMLConstants; import com.agencyport.domXML.changemanagement.MergeManager; import com.agencyport.domXML.changemanagement.ModInfoGroup; import com.agencyport.pagebuilder.Page; import com.agencyport.policysummary.changemanagement.AggregateLevelChangeDetail; import com.agencyport.policysummary.changemanagement.PolicyChangeSummarizer; import com.agencyport.shared.APEException; import com.agencyport.utils.ArrayHelper; /** * The ComercialAutoPolicyChangeSummarizer class provides a few special handlers * for the aggregate description portion for a couple of the aggregate eye catchers.
```

```
/* public class CommAutoPolicyChangeSummarizer extends PolicyChangeSummarizer { /** * Constructs an instance. You cannot and should not share an instance across threads. * @see com.agencyport.policysummary.changemanagement.PolicyChangeSummarizer */ public CommAutoPolicyChangeSummarizer(MergeManager mergeManager, DataManager dataManager, String defaultValueWhenNotPresent, String defaultDisplayDateFormat, String defaultXMLStorageDateFormat, String policyAggregateName) throws APEException { super(mergeManager, dataManager, defaultValueWhenNotPresent, defaultDisplayDateFormat, defaultXMLStorageDateFormat, policyAggregateName); } /* (non-Javadoc) * @see com.agencyport.policysummary.changemanagement.PolicyChangeSummarizer#normalizeAggregateLevelChangeDetail(com.agencyport.policysummary.changemanagement.AggregateLevelChangeDetail) */ protected void normalizeAggregateLevelChangeDetail(AggregateLevelChangeDetail aggregateLevelChangeDetail, Page page){ if (page.getTitle().equals("Additional Policy Coverages")) normalizeAggregateLevelActionCodeForAdditionalCov(aggregateLevelChangeDetail); } /** * Since the roster processing around additional policy coverages involves a complex set of rules, this method * provides the fuzzy logic necessary so that the roster and the policy change summary experiences are in synch * with related associated special field handling. * @param aggregateLevelChangeDetail is the aggregate level detail instance. */ private void normalizeAggregateLevelActionCodeForAdditionalCovs(AggregateLevelChangeDetail aggregateLevelChangeDetail){ int productsInterpretationOfActionCode = aggregateLevelChangeDetail.getActionCode(); switch (productsInterpretationOfActionCode){ case IXMLConstants.ADD_ACTION: case IXMLConstants.DELETE_ACTION: return; // We trust this interpretation } int save = apData.changeViewType(IXMLConstants.VIEW_ORIGINAL_DOCUMENT); IndexManager indexManager = dataManager.getIndexManager(); try { int[] idArray = indexManager.getCurrentIdArray("CommAutoLineBusiness.CommlRateState", apData, IXMLConstants.VIEW_ORIGNAL_DOCUMENT);
```

```
boolean hasCovsOnOriginal = false; boolean hasCovsOnCurrent = false; if (idArray != null) hasCovsOnOriginal = AdditionalPolicyCoverageHelper.IsAdditionalCovDataAvailable(apData, idArray); apData.changeViewType(IXMLConstants.VIEW_CURRENT_DOCUMENT); idArray = indexManager.getCurrentIdArray("CommAutoLineBusiness.CommlRateState", apData, IXMLConstants.VIEW_CURRENT_DOCUMENT); if (idArray != null) hasCovsOnCurrent = AdditionalPolicyCoverageHelper.IsAdditionalCovDataAvailable(apData, idArray); if (hasCovsOnOriginal && !hasCovsOnCurrent) aggregateLevelChangeDetail.setForcedActionCode(IXMLConstants.DELETE_ACTION); else if (!hasCovsOnOriginal && hasCovsOnCurrent) aggregateLevelChangeDetail.setForcedActionCode(IXMLConstants.ADD_ACTION); } finally { apData.changeViewType(save); } }
```

The following is an example of a roster reader overriding the policy change summarizer factory method to engage its own policy change summarizer:

```
/** * Mar 8, 2006 @author Agencyport Insurance Services, Inc. */ package com.agencyport.custom.commAuto; import com.agencyport.data.DataManager; import com.agencyport.data.apdata.RosterReader; import com.agencyport.domXML.APDataCollection; import com.agencyport.domXML.changemanagement.MergeManager; import com.agencyport.policy.summary.changemanagement.PolicyChangeSummarizer; import com.agencyport.servlets.commAuto.endorse.CommAutoPolicyChangeSummarizer; import com.agencyport.shared.APEException; public class AutoAdditionalPolicyCovsRosterReader extends RosterReader { /* (non-Javadoc) * @see com.agencyport.data.apdata.RosterReader#passesFilterCriterion(int[]) */ protected boolean passesFilterCriterion(int[] idArray){ boolean passesBaseClassFilter= super.passesFilterCriterion(idArray); if (!passesBaseClassFilter)
```

return passesBaseClassFilter; // Display the roster entry only if the additional coverage data exist  
if(AdditionalPolicyCoverageHelper.IsAdditionalCovDataAvailable(data, idArray)) return true; return false; } protected String getControlledState(APDataCollection data){ return data.getFieldValue("CommPolicy.ControllingStateProvCd", ""); } /\* (non-Javadoc) \* @see com.agencyport.data.apdata.RosterReader#createPolicyChangeSummaryizer(com.agencyport.domXML.changemanagement.MergeManager, com.agencyport.data.DataManager) \*/ protected PolicyChangeSummarizer createPolicyChangeSummaryizer(MergeManager mergeManager, DataManager dataManager) throws APEException { return new CommAutoPolicyChangeSummarizer(mergeManager, dataManager, null, null, null, "CommPolicy"); } } Example of a using same policy change summarizer factory in policy change summarizer contexts: package com.agencyport.servlets.commAuto.endorse; import com.agencyport.data.DataManager; import com.agencyport.domXML.APDataCollection; import com.agencyport.policy.summary.changemanagement.PolicyChangeSummarizer; import com.agencyport.shared.APEException; /\*\* \* The CMDDisplaypolicyChangeSummary class is an example on how to prepare a \* policy change summary object. \*/ public class CMDDisplaypolicyChangeSummary extends com.agencyport.servlets.shared.endorse.CMDDisplaypolicyChangeSummary{ protected PolicyChangeSummarizer getSummarizer(APDataCollection apData, DataManager dataManager) throws APEException { return new CommAutoPolicyChangeSummarizer(apData.getMergeManager(), dataManager, null, null, null, "CommPolicy"); } }

## Managing Multiple Document View Types on APDataCollection

For policy change based applications, the responsibility for managing APDataCollection's document view type currently in context is raised from new business applications. The following guiding principles should be observed:

- Never update the original view in APDataCollection. There is nothing built-in APDataCollection to prevent this, so the developer needs to be aware of this.
  - Never assume the document view is set to what you want it to be with exception to the display side of special field helpers.
  - It is the responsibility of application custom code to restore the document view back to what it was before returning to the framework.
- The following Java code should be used:

```
int saveDocumentViewType = apData.changeViewType(XMLConstants.VIEW_xxxxxxx_DOCUMENT); try { // do stuff } finally {
apData.changeViewType(saveDocumentViewType); }
```

## Index Management

Index management becomes more complex in policy change since two documents are now relevant to the programmer: the current and the original. Here are some general guiding principles to observe:

- On recover actions, the index set on the IndexManager class refers to the aggregate on the ORIGINAL document (versus the current) that is being recovered.
- Never assume that the index position for an aggregate common to both the original and the current documents are the same. The IndexManager class now has an extended getCurrentIdArray method that accepts a view type parameter. The following code is a bit of a custom code taken from the ACORD template MA personal auto endorsement application. The executeCustomPostDataStaging() method is being called after the SDK data engine has recovered a DriverVeh aggregate from the original document. This code is trying to verify that the related driver and vehicle aggregates are still present on the current document. If either the driver or vehicle aggregate is not present, then the ref attribute to that aggregate is removed on the driver veh aggregate that has just been recovered.

```
/* * Created on Jun 14, 2006 by Administrator for Agencyport Insurance Services */ package com.agencyport.servlets.persauto.endorse; import com.agencyport.data.IndexManager; import com.agencyport.domXML.APDataCollection; import com.agencyport.domXML.XMLConstants; import com.agencyport.servlets.base.CMDBaseProcessRosterRecover; import com.agencyport.shared.APEException; import com.agencyport.webshared.IWebsharedConstants; /** * The CMDProcessAssignmentRecover class illustrates a custom recover implementation. */ public class CMDProcessAssignmentRecover extends CMDBaseProcessRosterRecover { private static final String ROOT_XPATH = "PersPolicy.DriverVeh"; private static final String DRIVER_REF = "DriverRef"; private static final String VEHICLE_REF = "VehRef"; private static final String LINE_BUSINESS = "PersAutoLineBusiness"; private static final String PERS_DRIVER = LINE_BUSINESS + ".PersDriver"; private static final String PERS_VEH = LINE_BUSINESS + ".PersVeh"; /** * This is a good illustration of overriding a recovery action as an APCCommand. * @see com.agencyport.servlets.base.APCommand#executeCustomPostDataStaging(int) */ protected void executeCustomPostDataStaging(int dataAccessType) throws APEException { if (IWebsharedConstants.DATA_ACCESS_TYPE_RECOVER != dataAccessType) return; IndexManager indexMgr = dataManager.getIndexManager(); APDataCollection apData = dataManager.getAPDataCollection(); // On recovery actions, the index manager current indices are in the context of the original document and
```

```

not// the current. Here we switch the APDataCollection back to the original document so that we can get the 'current' indices // in terms of the
current document. Custom code should never assume which document view // is in context. int save =
apData.changeViewType(XMLConstants.VIEW_ORIGINAL_DOCUMENT); // This will get the id array of the driver vehicle aggregate which
was just recovered in terms of its index // in the current document int[] idArray = indexMgr.getCurrentIdArray("PersPolicy.DriverVeh", apData,
XMLConstants.VIEW_CURRENT_DOCUMENT); // Now switch back over to the current document since the framework has just // recovered
the driver vehicle aggregate from the original. apData.changeViewType(XMLConstants.VIEW_CURRENT_DOCUMENT); try { // Check the
DriverRef to make sure the driver still exists. If the driver no // longer exists, remove the attribute. processRecover(apData,idArray,
DRIVER_REF, PERS_DRIVER); // Check the VehRef to make sure the vehicle still exists. If the vehicle no // longer exists, remove the
attribute. processRecover(apData,idArray, VEHICLE_REF, PERS_VEH); } finally { apData.changeViewType(save); } } private void
processRecover(APDataCollection apData, int[] idArray, String refAttribute, String relationshipRootXPath) { String idRef =
apData.getFieldValue(ROOT_XPATH + ".@" + refAttribute, idArray, null); if (idRef != null) { String relationshipXPath =
relationshipRootXPath + "[@id=\"" + idRef + "\"]"; int count = apData.getCount(relationshipXPath); // Relationship no longer exists so remove
attribute if (count == 0) { apData.deleteAttribute(ROOT_XPATH, idArray, refAttribute); } } }

```

## Direct Use of JDOM is Discouraged

Any custom JDOM direct update code that circumvents the AXE APDataCollection class should be refactored to use APDataCollection or must propagate ID attribute values accordingly on aggregate additions using an approach that matches the Agencyport AXE interpretation.

# ACORD Policy Change Rendering Technique

This section describes how to use the AXE API, specifically the `ModInfoGroup`, `ModInfo` and `PolicyChangeSummarizer` classes, to create an ACORD-based policy change PDF form (ACORD 71 form).

Review and debug the following Java program as you review this section:

```
ap_sdk\apbasetest\com.agencyport.domXML.changemanagement.PAEndorsementAxeTestCase.java
```

## com.agencyport.domXML.changemanagement.ModInfoGroup

The `com.agencyport.domXML.changemanagement.ModInfoGroup` class, in conjunction with the `ModInfo` class internally, are the "work horse" classes for policy change PDFs and change summary page rendering. Without the `ModInfoGroup`, the application programmer is left with a linear list of `ModInfo` instances attached to the entire policy. The need for grouping `ModInfo` instances by aggregate occurrence is delivered by this class. As the name of this class implies, the main purpose of this class is to group `ModInfos` by aggregate type and occurrence and provide a way to filter `ModInfos` specific to sub aggregates.

```
Static ModInfoGroup[] selectModInfoGroups(MergeManager mergeManager, String aggregateElementPath)
```

This method is how to get a result set of `ModInfoGroup` instances for a given aggregate name for an entire `APDataCollection` instance.

### Example

```
ModInfoGroup[] vehicleModInfoGroups = ModInfoGroup.selectModInfoGroups(mergeManager, "PersAutoLineBusiness.PersVeh");
```

The following are some observations to make when reviewing the returned data:

- There will be one `ModInfoGroup` for each vehicle occurrence present in the merged document, regardless of whether the vehicle experienced changes, additions, deletions or NO changes. Those `ModInfoGroup` instances with no actions will have an empty `ModInfo` array attached (an array with a length of zero) and will have a nominal presence in the result set with an action code of `NO_ACTION`.
- The order of the `ModInfoGroups` in the result set is thought as "correct" as far as the default notion of aggregate identity. This order will NOT match the physical XML order of the aggregates in the merged document in the case where any aggregate deletes have occurred; it adopts what has been earlier introduced as an implied aggregate identity order. The sequence ID available on each `ModInfoGroup` occurrence is a flat linear 1 based sequence numbering algorithm directly relating to its implied "identity." Any reordering of this result set can be achieved by using the `sort` method on the `ModInfoGroup` class.
- It is important to keep in mind that the `ModInfoGroup` "knows" nothing about roster filtering. If you want to filter `ModInfoGroups` via a roster filter, you must filter the `ModInfoGroup` result set with custom code or use a method in the framework, such as `PolicyChangeSummarizer.filterModInfoGroupsPerRosterReaderCriterion()`. This method applies the same filtering logic to the `ModInfoGroup` result set, which it "knows" how to apply to the aggregates.

```
int getAggregateActionCode()
```

`int getAggregateActionCode()` is the derived action code that is based on the underlying action codes of the included mod info instances. The derivation works as follows: If there is only one add mod info and it applies directly to the aggregate under consideration, the n action code returned will be an add. The same consideration applies to delete. Anything else is reported as a change. Relationship changes are reported as a change as well. An aggregate that has undergone no modification will have a `ModInfoGroup` with a value of `NO_ACTION`.

```
int getCombinedDetailedMOdInfoActionCodeValues()
```

`int getCombinedDetailedMOdInfoActionCodeValues()` is equivalent to the following:

```
return ModInfo.determineCombinedActionCodes(this.getModInfos());
```

This provides the action code bit mask for all of the unique action codes distributed across all of the `ModInfo` instances contained within this group.

```
ElementPathExpression getAggregateElementPath()
```

`ElementPathExpression getAggregateElementPath()` provides the XPath expression replete with ID predicates for the aggregate occurrence associated with this group. This can be used to build other Agencyport XPath expressions to gain access to child element data for this aggregate.

### **String getIdRef()**

`String getIdRef()` provides the ID ref value for the domain aggregate for which this `ModInfoGroup` applies. This will match the ID predicate value on the right most element on the aggregate element path for this group.

### **String getModInfos()**

`String getModInfos()` returns the entire `ModInfo` result set associated with this group.

### **MergeManager getMergeManager()**

`MergeManager getMergeManager()` returns the merge manager instance associated with this instance.

### **String getSequenceId()**

`String getSequenceId()` returns the aggregate identity for this group. This is useful for identifying aggregates on various outputs, such as PDFs and change summary pages.

### **Example**

Vehicle n, where n is the return value of this method.

### **void setSequenceId(String sequenceId)**

`void setSequenceId(String sequenceId)` gives the application the ability to override the `ModInfoGroup`'s default definition of aggregate identity. If you intend on using this method, then you will have to refresh the aggregate identity for any new result sets returned by either `selectModInfoGroup` method variation.

### **ModInfoGroups[] selectModInfoGroups(String subAggregatePathFragment)**

Provides a way to gather a subset of the `ModInfoGroup` instances within a given group. An example of this would be to get the `ModInfoGroup` result set for a set of additional interests within a given vehicle.

Assume a `ModInfoGroup aVehicleModInfoGroup` acquired from a previous API call.

To get all of the additional interest `ModInfoGroups` for a given vehicle, call this method as follows:

```
ModInfoGroup[] additionalInterestModInfoGroups = aVehicleModInfoGroup.selectModInfoGroups("AdditionalInterest");
```

### **Note**

You pass in the element path relative to the group's own aggregate path and not the usual entire element path.

### **ModInfo[] selectModInfos(String subAggregatePathFragment, int actionCodeMask)**

This method filters out the mod info instances associated with this mod info group that applies to a child aggregate or sub aggregate physically nested within the aggregate under consideration, as well as the action code bit mask. If there are no `ModInfos` for the sub aggregate, then zero length arrays are returned.

The following illustrates how to get the bodily injury coverage `ModInfo` for a given vehicle:

```
ModInfo [] bodilyInjuryModInfos = aVehicleModInfoGroup.selectModInfos("Coverage[Coverage['BI']]");
```

### **Note**

You pass in the sub element path relative to the group's own aggregate path and not the usual entire element path.

### **int howAffected(String subAggregatePathFragment)**

This is a wrapper method that first invokes the `selectModInfos` API and then passes that to `ModInfo.determineCombinedActionCodes` and returns that result. This provides an easy way to see if a subaggregate was affected and how it has been affected.

The following illustrates how you figure out what happened to a particular coverage from an action code standpoint.

```

int coverageActionCode = aVehicleModInfoGroup.howAffected("Coverage[Coverage['BI']]"); switch (coverageActionCode) { case
IXMLConstants.NO_ACTION: return "nothing changed"; case IXMLConstants.ADD_ACTION: return "coverage was added"; case
IXMLConstants.DELETE_ACTION: return "coverage was dropped"; case (IXMLConstants.ADD_ACTION |
IXMLConstants.DELETE_ACTION): return "coverage was dropped, then readded"; default: return "coverage was changed"; }

```

## ACORD Form 71 (Personal Auto Policy Change PDF) Rendering Specifics

### Identification Section

This section indicates how to render the PDF section in regards to change management.

#### Note

Carefully evaluate the `com.agencyport.policysummary.PolicySummarizer` and `com.agencyport.policysummary.changemanagement.PolicyChangeSummarizer` classes to see how much of the existing functionality can be leveraged to render this section.

Review all Java doc in the `apwebappcom.agencyport.policysummary` and `com.agencyport.policysummary.changemanagement` packages before embarking on any PDF rendering custom code (for APDataCollection data access).

### Printing of Name Insured

The insured name should come from the original document view before any changes. The following code then follows:

```

// Switch to the original document view int save = apData.changeViewType(IXMLConstants.VIEW_ORIGINAL_DOCUMENT); String
lastName =
apdata.getFieldValue("InsuredOrPrincipal[InsuredOrPrincipalInfo.InsuredOrPrincipalRoleCd='Insured'].GeneralPartyInfo.NameInfo.PersonNam
e.Surname"); // get other fields // Restore view type apData.changeViewType(save);

```

Consider using the `com.agencyport.policysummary.PolicySummarizer.getInsuredName()` method, which supports both commercial and personal type names, for both the original and/or current views.

### Determine whether Name Insured or Mailing Address has Changed

To determine whether a set of aggregates related to the insured's personal information has changed, use the `ModInfoGroup` API, specifically the `howAffected` API.

First get the default merge manager from `APDataCollection`  
`MergeManager mergeManager = apData.getMergeManager();`

There is one mod info group assumed for the insured. Assume you will only get one `ModInfoGroup` returned by the select API.

```

ModInfoGroup[] insuredModInfoGroups = ModInfoGroup.selectModInfoGroups(mergeManager,
"InsuredOrPrincipal[InsuredOrPrincipalInfo.InsuredOrPrincipalRoleCd='Insured']");
ModInfoGroup insuredModInfoGroup = insuredModInfoGroup[0]; // assume just one

```

To see if the insured's name has changed, the following conditional logic can be used:

```

if(insuredModInfoGroup.howAffected("GeneralPartyInfo.NameInfo") != IXMLConstants.NO_ACTION) // then you can assume that the insured
name has changed

```

Check the usability of the `PolicyChangeSummarizer hasInsuredNameChanged` method.

Likewise, to determine whether the mailing address has changed, use the following:

```

if(insuredModInfoGroup.howAffected("GeneralPartyInfo.Addr[AddrTypeCd='MailingAddress']") != IXMLConstants.NO_ACTION) // then you
can assume that the insured's mailing address has changed

```

Check the usability of the `PolicyChangeSummarizer hasInsuredMailingAddressChanged` method.

### Determine whether Billing Plan has Changed

This is a very perfunctory task since the `BillingMethod` code is a single occurring aggregate.

```
ModInfo[] billingPlanModInfo = mergeManager.selectModInfos("PersPolicy.BillingMethodCd", IXMLConstants.CHANGE_ACTION); If
(billingPlanModInfo.length > 0) { // the billing method code has changed and should be rendered by using the current view }
```

Check the usability of the `PolicyChangeSummarizer hasBillingMethodCodeChanged` method.

## Vehicle Description/Use Section

The ACORD 71 form accommodates a three vehicle list in this section. The coverages section that follows this section accommodates only two vehicles. The following general presentation rule applies:

- To keep the vehicle coverages and their respective descriptions/use sections on the same page, the maximum number of vehicles that can be reported on a single page that were changed and/or added as part of the endorsement is two. The additional row in the descriptions/use could theoretically be used to report a dropped vehicle. It is up to you whether to position vehicles that don't fit on the first page on a duplicate of that page type rather than using some overflow technique.

It is also important to describe how to determine the Type of Change column for this section and how to render it. The following are some operational assumptions of `ModInfo` instances rendered for each vehicle `ModInfoGroup` instance:

- The sequence ID on the vehicle mod info group should be used for the Veh#, regardless of whether this is generated by the framework or overridden by custom code that uses the `setSequenceId` method on the `ModInfoGroup` class.
- If a vehicle were added, then one (and only one) mod info instance will be generated for that `PersVeh` aggregate alone. All child elements/aggregates nested within the vehicle are assumed to be added and will not have explicit mod info instances associated. This applies to `AdditionalInterest` aggregates or any other aggregate nested within the `PersVeh` aggregate.
- If a vehicle were "dropped" or deleted, then one (and only one) mod info instance will be generated for that entire `PersVeh` aggregate alone. All child elements/aggregates nested within that dropped vehicle are assumed to be deleted and will not have explicit mod info instances generated.
- Changes for vehicles are not so straight forward. A vehicle that was changed could have a variety of different changes.

## Example

A relationship change to a different garaging location, some limits on existing coverages changed, some additional optional coverages added, some existing optional coverages dropped and additional interest modifications.

The following code will gather all of the `ModInfoGroups` for all of the vehicles associated with this policy, regardless of whether they were added, changed, deleted or had no changes at all. Remember, there is a one-to-one relationship between a `ModInfoGroup` and `PersVeh` instance.

```
ModInfoGroup[] vehicleModInfoGroups = ModInfoGroup.selectModInfoGroups(mergeManager, "PersAutoLineBusiness.PersVeh");
```

Refer to the retrieving all of the vehicle mod info groups in this policy and Vehicle mod info group process loop Java samples in the [AXE Java Source Code Samples](#) section. Think about the following:

- If a vehicle is added, then all of the descriptive fields should be displayed.
- If a vehicle is dropped, then the fields that are only necessary to identify that vehicle should print. The code sample provides a plausible interpretation of what fields constitute the "identifying fields" for a personal vehicle.
- If a vehicle has changed, then the "identifying fields" should all print including all accompanying fields that have changed. Technically, some detail fields could have been added or changed or even deleted.

The definition of vehicle change is embodied in the method listed in the `hasVehicleChangedInSomeWay` section:

- For detecting how and if driver percentages have changed in regards to a specific vehicle, refer to the determining whether driver percentages have changed in the [AXE Java Source Code Samples](#) section.

## Important

The driver vehicle aggregates are not physically nested within either the vehicle or driver aggregates and so the corresponding `ModInfoGroup` instances need to be retrieved using, in this case, the vehicle aggregate ID.

- To determine whether to print changes for garaging locations, the first condition that must pass is whether the garaging location is different from the mailing address. Since ACORD 1.x, these are physically separate aggregates (`Location` and `InsuredOrPrincipal[InsuredOrPrincipalInfo.InsuredOrPrincipalRoleCd='Insured'].GeneralPartyInfo.Addr`). There is no other alternative other than to simply compare the two address blocks element by element between the garaging location with the insured's mailing address. Consider using some of the foundational constructs in the `PolicySummarizer` class, namely the `getStandardAddressBlock()` to first get the address data into `String[]` arrays and then using the `areEqual()` method to compare the two address blocks. If it is different, then you will need to see if the garaging location was affected in some way. Refer to the determining whether the garaging location relationship has changed in the [AXE Java Source Code Samples](#) section or the data associated with the garaging location has changed for how to ascertain this.

## Vehicle Coverage/Premium Section

The coverage section only needs to be rendered for vehicles where there is one or more changes to the coverages for an existing vehicle or if you are dealing with a newly added vehicle.

For newly added vehicles, all of the selected coverages should be rendered.

For existing vehicles, a coverage needs to be rendered if it were added, changed or deleted.

Use the sequence ID on the vehicle mod info group for the Veh#.

Use an algorithm similar to the logic present in the `determineVehicleCoverageAction` section for changed vehicles to determine the action on each coverage line item.

### Note

Notice the code that detects the delete/add sensitivity. Remember the user could have first deleted the coverage, committed the data and then revisited the page and re-added the same coverage back. Since the PDF does not support the notion of deleting a coverage and then reading the same coverage, you should treat this scenario as a change.

## General Information Section

To determine whether a vehicle or driver was added, loop through the vehicle or driver mod info groups and call the `howAffected` method.

### Driver Information

This section applies to any drivers that were added, dropped or some other change occurred. Again, the same notion applies, regardless of the use of "identity fields" for drivers that are dropped or changed. An approach similar to the vehicle identity fields is needed. The acquisition of the driver mod info groups is consistent with previous examples. Refer to the Retrieve all of the driver mod info groups for this policy and Driver mod info process loop samples in the [AXE Java Source Code Samples](#) section.

### Accidents/Convictions

Loop through all of the accident violation aggregates for a specific driver. The `determineAccidentsViolationsToPrintForDriver` section illustrates how to ascertain whether there were any new accident or violation aggregates that apply to this driver.

### Additional Interest

The notion of "identity fields" applies to this, as well as for deleted or changed aggregates.

# AXE Java Source Code Samples

This section includes AXE Java source code samples.

## PAEndorsementAxeTestCase.java

```
/* * Created on Jan 24, 2006 by norm AgencyPort Insurance Services, Inc. * * PAEndorsementAxeTestCase.java */ package
com.agencyport.domXML; import com.agencyport.domXML.changemanagement.MergeManager; import
com.agencyport.domXML.changemanagement.ModInfo; import com.agencyport.domXML.changemanagement.ModInfoGroup; /* * The
PAEndorsementAxeTestCase class illustrates the proper use of the AXE change management * API, specifically the ModInfoGroup API, for
rendering PDFs or other general purpose * change rendering requirements. */ public class PAEndorsementAxeTestCase extends
AxeBaseTestCase { // Some canonical base XPaths for commonly used personal auto aggregates private static final ElementPathExpression
insuredBaseXPath = new ElementPathExpression("InsuredOrPrincipal[!InsuredOrPrincipalInfo.InsuredOrPrincipalRoleCd='Insured']"); private
static final ElementPathExpression insuredMailingAddressBaseXPath = new
ElementPathExpression("InsuredOrPrincipal[InsuredOrPrincipalRoleCd='Insured'].GeneralPartyInfo.Addr[AddrTypeCd
='MailingAddress']"); private static final ElementPathExpression vehicleBaseXPath = new
ElementPathExpression("PersAutoLineBusiness.PersVeh"); private static final ElementPathExpression driverBaseXPath = new
ElementPathExpression("PersAutoLineBusiness.PersDriver"); private static final ElementPathExpression garagingLocationBaseXPath = new
ElementPathExpression("Location.Addr[AddrTypeCd='GaragingAddress']"); /* * Change some data fields on the insured's aggregate */ private
void changeInsuredData() { // Change marital status to single String maritalStatusXPath =
ElementPathExpression.buildNextElementPath(insuredBaseXPath.getExpressionWithAllPredicates(),
"InsuredOrPrincipalInfo.PersonInfo.MaritalStatusCd"); apData.setFieldValue(maritalStatusXPath, "S");

// change last name String lastNameXPath =
ElementPathExpression.buildNextElementPath(insuredBaseXPath.getExpressionWithAllPredicates(),
"GeneralPartyInfo.NameInfo.PersonName.Surname"); apData.setFieldValue(lastNameXPath, "Baker"); // change mailing address
apData.setFieldValue(ElementPathExpression.buildNextElementPath(insuredMailingAddressBaseXPath.getExpressionWithAllPredicates(),
"Addr1"), "248 King Street");
apData.setFieldValue(ElementPathExpression.buildNextElementPath(insuredMailingAddressBaseXPath.getExpressionWithAllPredicates(),
"City"), "Groveland");
apData.setFieldValue(ElementPathExpression.buildNextElementPath(insuredMailingAddressBaseXPath.getExpressionWithAllPredicates(),
"StateProvCd"), "MA");
apData.setFieldValue(ElementPathExpression.buildNextElementPath(insuredMailingAddressBaseXPath.getExpressionWithAllPredicates(),
"PostalCode"), "01834"); } /* * Change the address of the first garaging location. Delete the 2nd location. */ private void fixGaragingData() { //
change the first garaging location
apData.setFieldValue(ElementPathExpression.buildNextElementPath(garagingLocationBaseXPath.getExpressionWithAllPredicates(), "Addr1"),
0, "248 King Street");
apData.setFieldValue(ElementPathExpression.buildNextElementPath(garagingLocationBaseXPath.getExpressionWithAllPredicates(), "City"), 0,
"Groveland");
apData.setFieldValue(ElementPathExpression.buildNextElementPath(garagingLocationBaseXPath.getExpressionWithAllPredicates(),
"StateProvCd"), 0, "MA");
apData.setFieldValue(ElementPathExpression.buildNextElementPath(garagingLocationBaseXPath.getExpressionWithAllPredicates(),
"PostalCode"), 0, "01834"); // delete second location that is no longer used since we will // wire to the same location that the first vehicle uses.
apData.deleteElement("Location", 1);

} /* * Change a bunch of stuff on a vehicle * @param vehicleIx is the vehicle index * @throws Exception */ private void
changeVehicleData(int vehicleIx) throws Exception { // Increase the annual mileage estimate
apData.setFieldValue(ElementPathExpression.buildNextElementPath(vehicleBaseXPath.getExpressionWithOnlyElementHierarchy(),
"EstimatedAnnualDistance.NumUnits"), vehicleIx, "30000"); // Increase some of the coverage limits
apData.setFieldValue(ElementPathExpression.buildNextElementPath(vehicleBaseXPath.getExpressionWithOnlyElementHierarchy(),
"Coverage[CoverageCd='BL'].Limit[0].FormatCurrencyAmt.Amt"), vehicleIx, "200000");
apData.setFieldValue(ElementPathExpression.buildNextElementPath(vehicleBaseXPath.getExpressionWithOnlyElementHierarchy(),
"Coverage[CoverageCd='BL'].Limit[1].FormatCurrencyAmt.Amt"), vehicleIx, "600000");
apData.setFieldValue(ElementPathExpression.buildNextElementPath(vehicleBaseXPath.getExpressionWithOnlyElementHierarchy(),
"Coverage[CoverageCd='MEDPM'].Limit[0].FormatCurrencyAmt.Amt"), vehicleIx, "10000");
apData.setFieldValue(ElementPathExpression.buildNextElementPath(vehicleBaseXPath.getExpressionWithOnlyElementHierarchy(),
"Coverage[CoverageCd='NO-FAULT'].Limit[0].FormatCurrencyAmt.Amt"), vehicleIx, "2000"); // Delete a coverage on vehicle #1
apData.deleteElement(ElementPathExpression.buildNextElementPath(vehicleBaseXPath.getExpressionWithOnlyElementHierarchy(),
"Coverage[CoverageCd='APC']"), 0); // change garaging location on this vehicle to the first one String newLocation =
apData.getFieldValue("Location.@id", 0, null);
apData.setFieldValue(ElementPathExpression.buildNextElementPath(vehicleBaseXPath.getExpressionWithOnlyElementHierarchy(),
"@LocationRef"), vehicleIx, newLocation); // change the address of the first garaging location
apData.setFieldValue(ElementPathExpression.buildNextElementPath(vehicleBaseXPath.getExpressionWithOnlyElementHierarchy(),
"AdditionalInterest.GeneralPartyInfo.Addr.Addr1"), 0, 0, "456 Garey North"); // delete the 3rd additional interest on vehicle #1
apData.deleteElement(ElementPathExpression.buildNextElementPath(vehicleBaseXPath.getExpressionWithOnlyElementHierarchy(),
```

```

"AdditionalInterest"), 0, 2); } /** * Add a new driver */ private void addDriver() { int driverIx =
apData.addElement(driverBaseXPath.getExpressionWithOnlyElementHierarchy(), IXMLConstants.FORCE_NEW_OCCURENCE);
apData.setFieldValue(ElementPathExpression.buildNextElementPath(driverBaseXPath.getExpressionWithOnlyElementHierarchy(),
"GeneralPartyInfo.NameInfo.PersonName.Surname"), driverIx, "Baker");
apData.setFieldValue(ElementPathExpression.buildNextElementPath(driverBaseXPath.getExpressionWithOnlyElementHierarchy(),
"GeneralPartyInfo.NameInfo.PersonName.GivenName"), driverIx, "Jonah");
apData.setFieldValue(ElementPathExpression.buildNextElementPath(driverBaseXPath.getExpressionWithOnlyElementHierarchy(),
"GeneralPartyInfo.NameInfo.PersonName.OtherGivenName"), driverIx, "H");
apData.setFieldValue(ElementPathExpression.buildNextElementPath(driverBaseXPath.getExpressionWithOnlyElementHierarchy(),
"DriverInfo.PersonInfo.GenderCd"), driverIx, "M");
apData.setFieldValue(ElementPathExpression.buildNextElementPath(driverBaseXPath.getExpressionWithOnlyElementHierarchy(),
"DriverInfo.PersonInfo.BirthDt"), driverIx, "1987-02-01");
apData.setFieldValue(ElementPathExpression.buildNextElementPath(driverBaseXPath.getExpressionWithOnlyElementHierarchy(),
"DriverInfo.PersonInfo.MaritalStatusCd"), driverIx, "S"); } /** * Deletes all DriverVeh aggregates relating to a specific vehicle */
@param vehRef is the vehicle id for which to apply the delete */ private void deleteAllPreviousDriverUsage(String vehRef) { String xPath =
"PersPolicy.DriverVeh[@VehRef=" + vehRef + "]"; int count = apData.getCount(xPath); while (count > 0){ apData.deleteElement(xPath);
count--; } }

```

### changeDriverPctUsage

```

/** * This applies the use percentage for a given vehicle for the 3 drivers */
@param vehicleIx is the index for the vehicle to apply the use
percents */
@param percentDriver1 is the use percent for this vehicle that driver 1 shares
@param percentDriver2 is the use percent for this
vehicle that driver 2 shares
@param percentDriver3 is the use percent for this vehicle that driver 3 shares */
private void
changeDriverPctUsage(int vehicleIx, int percentDriver1, int percentDriver2, int percentDriver3) { String vehRef =
apData.getFieldValue(ElementPathExpression.buildNextElementPath(vehicleBaseXPath.getExpressionWithOnlyElementHierarchy(), "@id"),
vehicleIx, null); deleteAllPreviousDriverUsage(vehRef); String driver1Ref =
apData.getFieldValue(ElementPathExpression.buildNextElementPath(driverBaseXPath.getExpressionWithOnlyElementHierarchy(), "@id"), 0,
null); String driver2Ref =
apData.getFieldValue(ElementPathExpression.buildNextElementPath(driverBaseXPath.getExpressionWithOnlyElementHierarchy(), "@id"), 1,
null); String driver3Ref =
apData.getFieldValue(ElementPathExpression.buildNextElementPath(driverBaseXPath.getExpressionWithOnlyElementHierarchy(), "@id"), 2,
null); if (percentDriver1 > 0) apData.setFieldValue("PersPolicy.DriverVeh[@DriverRef=" + driver1Ref + " && @" +
"VehRef=" + vehRef + "].UsePct", Integer.toString(percentDriver1)); if (percentDriver2 > 0) apData.setFieldValue("PersPolicy.DriverVeh[@DriverRef=" +
driver2Ref + " && @" + "VehRef=" + vehRef + "].UsePct", Integer.toString(percentDriver2)); if (percentDriver3 > 0)
apData.setFieldValue("PersPolicy.DriverVeh[@DriverRef=" + driver3Ref + " && @" + "VehRef=" + vehRef + "].UsePct",
Integer.toString(percentDriver3)); }

```

### printInsuredMailingAddress

```

/** * This prints the insured's mailing address */
@param viewType is the document view to use.
*/
private void printInsuredMailingAddress(int viewType) { int save = apData.changeViewType(viewType); System.out.println("Printing from " + apData.getViewType());
System.out.println(apData.getFieldValue(ElementPathExpression.buildNextElementPath(insuredMailingAddressBaseXPath.getExpressionWithAllPredicates(),
"Addr1"), ""));
String addr2 =
apData.getFieldValue(ElementPathExpression.buildNextElementPath(insuredMailingAddressBaseXPath.getExpressionWithAllPredicates(),
"Addr2"), ""); if (addr2.length() > 0) System.out.println(addr2);
System.out.print(apData.getFieldValue(ElementPathExpression.buildNextElementPath(insuredMailingAddressBaseXPath.getExpressionWithAllPredicates(),
"City"), ""));
System.out.print(",");
System.out.print(apData.getFieldValue(ElementPathExpression.buildNextElementPath(insuredMailingAddressBaseXPath.getExpressionWithAllPredicates(),
"StateProvCd"), ""));
System.out.print(" ");
System.out.print(apData.getFieldValue(ElementPathExpression.buildNextElementPath(insuredMailingAddressBaseXPath.getExpressionWithAllPredicates(),
"PostalCode"), ""));
apData.changeViewType(save); }

```

### hasAggregateChanged

```

/** * A very simple helper class that returns a false if the subaggregate for the current mod info group has no changes otherwise true */
@param modInfoGroup is the mod info group for the aggregate in consideration to evaluate
@param subAggregatePath is the sub aggregate path identifier.
*/
private boolean hasAggregateChanged(ModInfoGroup modInfoGroup, String subAggregatePath) { int howAffected =
modInfoGroup.howAffected(subAggregatePath); if (howAffected != IXMLConstants.NO_ACTION){
System.out.println(subAggregatePath + " changed."); return true; } else { System.out.println(subAggregatePath + " did not change."); return
false; } }

```

### shouldTryToPrintVehicleDescriptionItem

```

/** * This delves into the notion of which vehicle description fields to render. Here
* are the understood rules:
* 1) If the overall action of the group is add then we need to print everything in the vehicle
* description area
* 2) If the overall action of the group is delete then we only need

```

to print those fields \* that are needed to identify the vehicle itself\* 3) If the overall action of the group is a change then we should always print the id fields

```
* but only print the remainder fields that have been added, changed or deleted * @param vehModInfoGroup is the vehicle mod info group in consideration * @param subAggregatePath is the sub aggregate to evaluate within that vehicle mod info group * @return true if vehicle has any changes at all (add, change or delete) */ private boolean shouldTryToPrintVehicleDescriptionItem(ModInfoGroup vehModInfoGroup, String subAggregatePath){ // These fields should always be printed if the vehicle regardless of // whether the vehicle was added, deleted or changed final String[] vehicleIdFields = { "Manufacturer", "Model", "ModelYear", "VehIdentificationNumber", "RegistrationStateProvCd" }; int aggregateActionCode = vehModInfoGroup.getAggregateActionCode(); boolean shouldPrint = false; if (aggregateActionCode == IXMLConstants.NO_ACTION) { shouldPrint = false; } else if (aggregateActionCode == IXMLConstants.ADD_ACTION) { shouldPrint = true; } else { boolean isAnIdField = false; for (int ix = 0; ix < vehicleIdFields.length && !isAnIdField; ix++){ if (subAggregatePath.indexOf(vehicleIdFields[ix]) > -1){ isAnIdField = true; } } if (isAnIdField){ shouldPrint = true; } else if (aggregateActionCode != IXMLConstants.DELETE_ACTION) { shouldPrint = hasAggregateChanged(vehModInfoGroup, subAggregatePath); } } return shouldPrint; }
```

## howGaragingLocaleChanged

```
/** This method illustrates how to get a mod info group related via a foreign id ref * relationship. This function determines if the vehicle under consideration has been * related to a different location than was in the original snapshot. It also figures
```

```
* out if the garaging location currently related to the vehicle has any data changed. * @param vehModInfoGroup is the vehicle mod info group under consideration * @return true if the overall resultant action code bit mask */ private int howGaragingLocaleChanged(ModInfoGroup vehModInfoGroup){ // Determine if the garaging location relationship has changed ModInfo[] relChangeModInfos = vehModInfoGroup.selectModInfos(null, IXMLConstants.RELATIONSHIP_CHANGE_ACTION); ModInfo garagingRelationChangeModInfo = ModInfo.findRelationshipChangeWithRefNameCd(relChangeModInfos, "LocationRef"); // Get the current Location associated with this vehicle String locationRef = apData.getFieldValue(vehModInfoGroup.getAggregateElementPath().getExpressionWithAllPredicates() + ".@LocationRef", null); int resultantOverallActionCode = IXMLConstants.NO_ACTION; if (garagingRelationChangeModInfo != null) { System.out.println("Vehicle #" + vehModInfoGroup.getSequenceId() + " has been related to a different location " + locationRef); System.out.println("Vehicle #" + vehModInfoGroup.getSequenceId() + " was previously related to location " + garagingRelationChangeModInfo.getPreviousRefsAsDelimitedString(" ")); resultantOverallActionCode = IXMLConstants.RELATIONSHIP_CHANGE_ACTION; } // See if the garaging location itself has changed String locationXPath = "Location[@id=" + locationRef + "]"; MergeManager mergeManager = vehModInfoGroup.getMergeManager(); ModInfoGroup[] locationModInfoGroups = ModInfoGroup.selectModInfoGroups(mergeManager, locationXPath); // There should only be one location aggregate associated with this vehicle ModInfoGroup locationModInfoGroup = locationModInfoGroups[0]; int locationDataActionCode = locationModInfoGroup.howAffected(null); resultantOverallActionCode |= locationDataActionCode; if (locationDataActionCode != IXMLConstants.NO_ACTION) { System.out.println("The garaging location to which vehicle #" + vehModInfoGroup.getSequenceId() + " is now related has changed"); } else { System.out.println("The data associated with the garaging location for vehicle #" + vehModInfoGroup.getSequenceId() + " has not changed"); } return resultantOverallActionCode; }
```

## determineVehicleDescriptionsToPrint

```
/** This loops through a bunch of fields on the vehicle to see if they should * be printed or not * @param vehModInfoGroup is the vehicle mod info group under consideration */ private void determineVehicleDescriptionsToPrint(ModInfoGroup vehModInfoGroup){ final String[] vehicleDescriptionFields = { "Manufacturer", "Model", "ModelYear", "VehIdentificationNumber", "RegistrationStateProvCd", "Displacement", "LeasedDt", "PurchaseDt", "NewVehInd", "VehSymbolCd", "EstimatedAnnualDistance" // etc, etc, etc }; for (int ix = 0; ix < vehicleDescriptionFields.length; ix++){ boolean shouldPrint = shouldTryToPrintVehicleDescriptionItem(vehModInfoGroup, vehicleDescriptionFields[ix]); if (shouldPrint){ System.out.println("The data associated with the aggregate: " + vehicleDescriptionFields[ix] + " for vehicle #" + vehModInfoGroup.getSequenceId() + " should be rendered"); } } }
```

## determineVehicleCoverageAction

```
/** This determines the action code relating to a specific coverage for a vehicle * @param vehModInfoGroup is the vehicle mod info group under consideration * @param coverageCd is the coverage code value * @return the combined resultant action of the specific coverage for this vehicle */ private int determineVehicleCoverageAction(ModInfoGroup vehModInfoGroup, String coverageCd){
```

```
// Get the mod infos for this particular coverage ModInfo[] modInfos = vehModInfoGroup.selectModInfos("Coverage[CoverageCd=" + coverageCd + "]", IXMLConstants.ALL_ACTIONS); int combinedAction = ModInfo.determineCombinedActionCodes(modInfos); if (combinedAction == IXMLConstants.NO_ACTION){ System.out.println("The " + coverageCd + " coverage on auto " + vehModInfoGroup.getSequenceId() + " did not change"); } else if (combinedAction == IXMLConstants.ADD_ACTION) { System.out.println("The " + coverageCd + " coverage on auto " + vehModInfoGroup.getSequenceId() + " added"); } else if (combinedAction == IXMLConstants.CHANGE_ACTION) { System.out.println("The " + coverageCd + " coverage on auto " + vehModInfoGroup.getSequenceId() + " changed"); } else if (combinedAction == IXMLConstants.DELETE_ACTION) { System.out.println("The " + coverageCd + " coverage on auto " + vehModInfoGroup.getSequenceId() + " dropped"); } else if ((combinedAction & (IXMLConstants.ADD_ACTION | IXMLConstants.DELETE_ACTION)) > 0){ System.out.println("The " + coverageCd + " coverage on auto " + vehModInfoGroup.getSequenceId() + " changed (via a delete then add"); } return combinedAction; }
```

## determineVehicleCoverageToPrint

```
/** This loops through several known PA coverages to see if they should print * or not. * @param vehModInfoGroup is the vehicle mod info group under consideration */
private void determineVehicleCoverageToPrint(ModInfoGroup vehModInfoGroup){ final String[] coverageCodes = { "APC", "CSL", "BI", "PD", "MEDPM", "UMISP", "UMPD", "COLL", "COMP", "NO-FAULT" }

// etc, etc, etc }; for (int ix = 0; ix < coverageCodes.length; ix++){ int action = determineVehicleCoverageAction(vehModInfoGroup, coverageCodes[ix]); } /** This determines if the driver percentages for a given vehicle have changed * in some way. * @param vehModInfoGroup is the vehicle mod info group under consideration * @return true if the usage percentages have changed for this vehicle */
private boolean hasDriverPercentagesChanged(ModInfoGroup vehModInfoGroup) { MergeManager mergeManager = vehModInfoGroup.getMergeManager(); // Get all the driver vehicle usage changes for this vehicle String driverVehSearchXPath = "PersPolicy.DriverVeh[@VehRef=' + vehModInfoGroup.getIdRef() + "' && position()=@id]"; // using this section will tell AXE to get all driver veh aggregates // associated with this particular vehicle ModInfoGroup[] driverVehGroups = ModInfoGroup.selectModInfoGroups(mergeManager, driverVehSearchXPath); for (int ix = 0; ix < driverVehGroups.length; ix++){ if (driverVehGroups[ix].getAggregateActionCode() != IXMLConstants.NO_ACTION) { System.out.println("The driver use percentages associated with vehicle #" + vehModInfoGroup.getSequenceId() + " have changed"); return true; } } System.out.println("The driver use percentages associated with vehicle #" + vehModInfoGroup.getSequenceId() + " have NOT changed"); return false; }
```

## hasVehicleChangedInSomeWay

```
/** This determines at a very high level whether the vehicle has any changes associated with * it either nested or indirectly at a garaging location or at a driver percentage level. * @param vehModInfoGroup is the vehicle mod info group under consideration * @return true if the vehicle has changed in some way either directly or indirectly */
private boolean hasVehicleChangedInSomeWay(ModInfoGroup vehModInfoGroup) {
```

```
boolean changesToNestedAggregates = vehModInfoGroup.getAggregateActionCode() != IXMLConstants.NO_ACTION; if (!changesToNestedAggregates){ System.out.println("There were no changes to any of the sub aggregates nested within vehicle #" + vehModInfoGroup.getSequenceId()); } else return true; if (vehModInfoGroup.getAggregateActionCode() != IXMLConstants.NO_ACTION){ return true; } else if (howGaragingLocaleChanged(vehModInfoGroup)!= IXMLConstants.NO_ACTION){ return true; } else if (hasDriverPercentagesChanged(vehModInfoGroup)){ return true; } return false; }
```

## determineAccidentsViolationsToPrintForDriver

```
/** This determines if the accidents or violations associated with a driver should be * printed or not. * @param vehModInfo Group is the driver mod info group under consideration */
private void determineAccidentsViolationsToPrintForDriver(ModInfoGroup driverInfoGroup){ MergeManager mergeManager = driverInfoGroup.getMergeManager(); // Get all the driver vehicle usage changes for this vehicle String accidentViolationSearchXPath = "PersPolicy.AccidentViolation[@DriverRef=' + driverInfoGroup.getIdRef() + "' && position()=@id]"; // using this section will tell AXE to get all driver veh aggregates // associated with this particular vehicle ModInfoGroup[] accidentViolationGroups = ModInfoGroup.selectModInfoGroups(mergeManager, accidentViolationSearchXPath); for (int ix = 0; ix < accidentViolationGroups.length; ix++){ if (accidentViolationGroups[ix].getAggregateActionCode() != IXMLConstants.NO_ACTION) { System.out.println("The accident / violation #" + accidentViolationGroups[ix].getSequenceId() + " associated with driver #" + driverInfoGroup.getSequenceId() + " was added or changed"); } } }
```

## doChangeDetection

```
/** This contains the logic to detect the various changes made to this policy. * @throws Exception */
public void doChangeDetection() throws Exception { // get the merge manager MergeManager mergeManager = apData.getMergeManager();
```

## Retrieving the insured's mod info group

```
ModInfoGroup[] insuredModInfoGroups = ModInfoGroup.selectModInfoGroups(mergeManager, insuredBaseXPath.getExpressionWithAllPredicates()); // We expect only one mod info group for the insured ModInfoGroup insuredModInfoGroup = insuredModInfoGroups[0];
```

## Call to printInsuredName

```
printInsuredName(IXMLConstants.VIEW_ORIGINAL_DOCUMENT); // Determine if the name changed if (hasAggregateChanged(insuredModInfoGroup, "GeneralPartyInfo.NameInfo")){
printInsuredName(IXMLConstants.VIEW_CURRENT_DOCUMENT); }
```

## Call to determine whether the insured's mailing address has changed

```
// Determine if the mailing address for the insured has changed if (hasAggregateChanged(insuredModInfoGroup, "GeneralPartyInfo.Addr[AddrTypeCd='MailingAddress']") {
printInsuredMailingAddress(IXMLConstants.VIEW_ORIGINAL_DOCUMENT);
printInsuredMailingAddress(IXMLConstants.VIEW_CURRENT_DOCUMENT); }
```

Retrieving all of the vehicle mod info groups in this policy

```
// Collect all of the mod info groups for all of the vehicles ModInfoGroup[] vehicleModInfoGroups =
ModInfoGroup.selectModInfoGroups(mergeManager,
"PersAutoLineBusiness.PersVeh");
```

Vehicle mod info group process loop

```
boolean driverPercentagesChanged = false; for (int ix = 0; ix < vehicleModInfoGroups.length; ix++){ ModInfoGroup vehModInfoGroup =
vehicleModInfoGroups[ix];
```

#### Determining whether anything has changed on this vehicle

```
int overallVehicleActionCode = vehModInfoGroup.getAggregateActionCode(); if (!hasVehicleChangedInSomeWay(vehModInfoGroup)){
System.out.println("Vehicle #" + vehModInfoGroup.getSequenceId() + " had no substantive changes"); continue; }
```

#### Determining which vehicle description fields to print on this vehicle

```
determineVehicleDescriptionsToPrint(vehModInfoGroup);
```

#### Determining whether the garaging location relationship has changed or the data associated with the garaging location has changed

```
int garagingActionCode = howGaragingLocaleChanged(vehModInfoGroup); if (garagingActionCode != IXMLConstants.NO_ACTION){
System.out.println("Garaging location for vehicle #" + vehModInfoGroup.getSequenceId() + " should be rendered if it is different than the
insured's mailing address"); }
```

#### Determining whether driver percentages have changed

```
if (hasDriverPercentagesChanged(vehModInfoGroup)) driverPercentagesChanged = true; if (overallVehicleActionCode ==
IXMLConstants.CHANGE_ACTION) determineVehicleCoverageToPrint(vehModInfoGroup); else if (overallVehicleActionCode ==
IXMLConstants.DELETE_ACTION){

System.out.println("No need to print coverages for Vehicle #" + vehModInfoGroup.getSequenceId() + " since vehicle is being dropped"); } else
if (overallVehicleActionCode != IXMLConstants.ADD_ACTION){ System.out.println("Should print all coverages for Vehicle #" +
vehModInfoGroup.getSequenceId() + " since vehicle is added"); System.out.println("Should print question answers 1 thru 4 since at least one
vehicle was added"); System.out.println("Should print all additional interests for Vehicle #" + vehModInfoGroup.getSequenceId() + " since
vehicle is added"); }
```

#### Retrieve all of the additional interests for this vehicle

```
// Now get the additional interests for just for this vehicle ModInfoGroup[] additionalInterestModInfoGroups =
vehModInfoGroup.selectModInfoGroups("AdditionalInterest");
```

#### Additional Interest mod info group process loop

```
for (int ij = 0; ij < additionalInterestModInfoGroups.length; ij++){ ModInfoGroup addlInterestGroup = additionalInterestModInfoGroups[ij];
```

#### Determination of whether this additional interest has changed

```
int addlInterestActionCode = addlInterestGroup.getAggregateActionCode(); if (addlInterestActionCode == IXMLConstants.ADD_ACTION) {
System.out.println("Addl interest #" + addlInterestGroup.getSequenceId() + " for vehicle #" + vehModInfoGroup.getSequenceId() + " was
added"); } else if (addlInterestActionCode == IXMLConstants.CHANGE_ACTION) { System.out.println("Addl interest #" +
addlInterestGroup.getSequenceId() + " for vehicle #" + vehModInfoGroup.getSequenceId() + " was changed"); } else if (
addlInterestActionCode == IXMLConstants.DELETE_ACTION) { System.out.println("Addl interest #" + addlInterestGroup.getSequenceId() +
" for vehicle #" + vehModInfoGroup.getSequenceId() + " was deleted"); } else { System.out.println("Addl interest #" +
addlInterestGroup.getSequenceId() + " for vehicle #" + vehModInfoGroup.getSequenceId() + " did not change"); } }
```

Retrieve all of the driver mod info groups for this policy

```
// Collect all of the mod info groups for all of the drivers ModInfoGroup[] driverModInfoGroups =
ModInfoGroup.selectModInfoGroups(mergeManager, "PersAutoLineBusiness.PersDriver");
```

Driver mod info process loop

```
for (int ix = 0; ix < driverModInfoGroups.length; ix++){ ModInfoGroup driverModInfoGroup = driverModInfoGroups[ix]; int
overallDriverAction = driverModInfoGroup.getAggregateActionCode(); // determine if any of the drivers have been added, changed or deleted if
(overallDriverAction == IXMLConstants.NO_ACTION){ System.out.println("Driver #" + driverModInfoGroup.getSequenceId() + " did not
```

```
change"); } else if (overallDriverAction == IXMLConstants.ADD_ACTION) { System.out.println("Driver #" + driverModInfoGroup.getSequenceId() + " was added"); } else if (overallDriverAction == IXMLConstants.CHANGE_ACTION) { System.out.println("Driver #" + driverModInfoGroup.getSequenceId() + " changed"); } else if (overallDriverAction == IXMLConstants.DELETE_ACTION) { System.out.println("Driver #" + driverModInfoGroup.getSequenceId() + " was deleted"); } determineAccidentsViolationsToPrintForDriver(driverModInfoGroup); } }

testModInfoGroup1

*** JUnit test case entry point for test #1 * @throws Exception */ public void testModInfoGroup1() throws Exception { testId = "pdf-test1-pos"; // Change a bunch of data. Please note that this change logic makes assumption // upon the data which it operates. changeInsuredData(); fixGaragingData();

changeVehicleData(1); addDriver(); changeDriverPctUsage(0, 75, 25, 0); changeDriverPctUsage(1, 25, 65, 10); doChangeDetection(); } }
```

# Change Management Configuration

You may need to extend the `ModInfo` aggregate to include the `AP_DomainAggregateXPath` element in the application's schema extension if that LOB template has not already done so. ACORD policy change transactions, such as transactions like `PersAutoPolicyModRq` already have the `ModInfo` aggregate defined.

When policy change requests are introduced into an SDK-based application, the original XML document is explicitly stored in the `origination_xmlstring` column in the `xmlstore` table by the SDK because the `APDataCollection` contains a non-null original document.

To engage the change management for new business upload scenarios, the application upload writer must first explicitly make use of the `com.agencyport.upload.UploadDataManager` SDK class. Then, to make this class write the original document to the database, which is the triggering event for engaging change management, the schema definition for the new business transaction must be extended by the application to include the AXE `ModInfo` definition.

## Example

The following example includes two important extensions:

- The `ModInfo` is extended to include the `AP_DomainAggregateXPath`
- A new business transaction type (personalauto quote) is extended to include the `ModInfo` element to trigger the persistence of the original document to the database for all upload-based work items.

```
<?xml version="1.0" encoding="UTF-8"?> <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified" attributeFormDefault="unqualified" version="1.7.0"> <xsd:redefine schemaLocation="acord-pe-v1_7_0-
nodoc-nocodes.xsd"> <xsd:complexType name="ModInfo_Type"> <xsd:annotation> <xsd:documentation>ModInfo enhancement for
AgencyPort endorsement support</xsd:documentation> </xsd:annotation> <xsd:complexContent> <xsd:extension base="ModInfo_Type">
<xsd:sequence> <xsd:element ref="AP_DomainAggregateXPath"/> </xsd:sequence> </xsd:extension> </xsd:complexContent>
</xsd:complexType> <xsd:complexType name="PersAutoPolicyQuoteInqRq_Type"> <xsd:complexContent> <xsd:extension
base="PersAutoPolicyQuoteInqRq_Type"> <xsd:sequence> <xsd:element ref="ModInfo" minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence> </xsd:extension> </xsd:complexContent> </xsd:complexType> </xsd:redefine> <xsd:element
name="AP_DomainAggregateXPath" type="AP_DomainAggregateXPath_Type"> <xsd:annotation> <xsd:documentation>Contains the AXE
XPath expression to the domain aggregate</xsd:documentation> </xsd:annotation> </xsd:element> <xsd:complexType
name="AP_DomainAggregateXPath_Type"> <xsd:simpleContent> <xsd:extension base="C-Infinite_NoID"> <xsd:attribute name="id"
type="ID"/> </xsd:extension> </xsd:simpleContent> </xsd:complexType> </xsd:schema>
```

# Product Definition

In Agencyport terms, a product consists of a number of components called resources, which are used together to bring insurance products to life in Agencyport software. Information on how to define these various pieces is included in this section. Often, a product encompasses the various transactions (new business, endorsement, quick quote, etc) for a single line of business. As you explore the product, you'll see descriptions of what each type of resource does.

## Note

You will always have only one product per line of business in your product folder.

# Transaction Definition/Page Library

## Transaction Definition

The metadata contained in the transaction file is used for the following:

- building HTML pages
- building the menu bar for the transaction
- determining page navigation
- mapping the HTML data fields to the data store

The transaction definition is a set of metadata that describes a single transaction.

### Example

A new business submission, renewal or endorsement for a line of business is described by a transaction definition file.

### Note

The terms Transaction Definition File (TDF), Transaction Definition and Page Library can be used interchangeably. Historically, an entire transaction was defined within one file, hence the term TDF; however, nowadays, the pages of a transaction can be defined within page libraries that can be shared across transactions.

The `TransactionDefinitionManager` class reads the transaction definition file and builds transaction objects based on the content of the definition file. A transaction object contains a collection of page objects and an associated menu template.

Within a transaction definition file, one or more pages are defined. Each of the pages contains one or more sections, termed page elements, and each of the page elements contain one or more HTML elements.

The transaction definition describes the placement, or rendering, of each of these components on the HTML page and at the field level, describing where each piece of data lives in the data store.

The navigation menu of the page is derived from the transaction definition file.

As mentioned above, the vast majority of insurance applications can be constructed using only a few basic "HTML page types." These page types are:

- Data entry - the most common and is used for any collection of loosely related HTML elements (e.g., a page capturing basic insured information, such as name, tax ID, legal entity, mailing address, etc).
- Roster - pages that are used to capture and display repeating data (e.g., covered locations, loss history, etc).
- Display only - normally, summary type pages that have no (or very few) data entry elements (e.g., a rating/quote results page or a review page).

Refer to the [Index of TDF Elements](#) for references.

The following illustrates a simplified transaction file of a simple two-page transaction TDF. Many attributes have been omitted for display purposes.

```
<transaction id="WC" title="Workers Compensation" target="WCPolicy"><page id="generalInfo" title="General Information" type="dataEntry"><pageElement type="tips"><fieldElement type="message">For help click on the question mark icon.</fieldElement></pageElement> <pageElement type="fieldset" legend="Insured Information"> <fieldElement type="text" id="InsuredName" label="Name of Insured" /> <fieldElement type="text" id="DBA" label="Doing Business As" /> <fieldElement type="text" id="TaxId" label="FEIN/SSN" /> <fieldElement type="text" id="YearsInBusiness" label="Years In Business" /><fieldElement type="selectlist" id="LegalEntity" label="Legal Entity"> <optionList reader="xmlreader" source="codes.xml" target="legalEntity" /> </fieldElement> </pageElement> </page><page id="locations" source="Location" title="Risk Locations" type="roster"><pageElement type="roster" legend="Risk Location" maxEntries="22"> <fieldElement type="column" id="Address" label="Address" /><fieldElement type="column" id="City" label="City" /> <fieldElement type="column" id="State" label="State" /> <fieldElement type="column" id="Zipcode" label="Zip" /> </pageElement> <pageElement type="fieldset" legend="Add Location"> <fieldElement type="text" id="Address" label="Address" /><fieldElement type="text" id="Address2" label="Address Line 2" /> <fieldElement type="text" id="City" label="City" /><fieldElement type="text" id="State" label="State" /> <fieldElement type="text" id="ZipCode" label="Zip Code" /> </pageElement> </page></transaction>
```

## Page Library

You can define pages in files called page libraries. A page is created in either a page library or a transaction in the same way, but only pages defined in page libraries can be imported across transactions. Pages created within a shared product page library can be imported into multiple LOB transactions.

## Application Properties

The TDF contains a set of metadata that describes a single transaction. The following application properties provide additional support for the TDF:

Property	Description	Default Value	Recommended Setting (if any)
print_transaction_definition_file	Indicates whether the transaction definition files have to log.	false	
transaction_definition_file_dump_directory	Provides the location of the directory where to stage the transaction files.	none	
date.maxLength	Indicates the value to be used for the maxLength and size attributes of TDF date fieldElements.	10	N/A

## Index of TDF Elements

The following is a list of TDF elements:

### Collection

Attribute	Applies to Type	Default Value
id	N/A	N/A
title	All	N/A

### Correction

There can be zero to many correction child elements for a given parent field element. The correction elements that are selected at runtime depend on the presence and category of the validation result associated with the field. Correction elements are given the precedence in the order they appear under a field element in the TDF for the condition they apply to: "The first corrector to update the data source wins."

Attribute	Applies to Type	Default Value	Notes
cap	findCloses t	FALSE	
className	custom	N/A	This attribute is optional except when type is set to custom. When present, it specifies the full Java package class name of a custom implementation of the com.agencyport.data.corrections.IDataValueCorrector interface. There is no default.
condition	All	missing or incorrect	This attribute is optional. It provides a way to select the right correction element given the classification of the related validation result currently associated with the field. The following are possible values: <ul style="list-style-type: none"><li>• missing - correction element selected if the related validation result category is equal to</li></ul>

Attribute	Applies to Type	Default Value	Notes
			<p>ValidationResult.VALIDATION_CATEGORY_ENTITY_MISSING_AND_REQUIRED</p> <ul style="list-style-type: none"> <li>incorrect - correction element selected if the related validation result category is equal to ValidationResult.VALIDATION_CATEGORY_ENTITY_PRESENT_BUT_INCORRECT</li> <li>missing or incorrect - correction selected if the related validation result category is of either value</li> </ul>
direction	findClosest	up	<p>This attribute is optional. Used by the findClosest correction method to determine which direction to find the closest numeric entry. The following are possible values:</p> <ul style="list-style-type: none"> <li>up - instructs the find closest correction method to search upward in the list when finding a closest match (typically good for coverage limits).</li> <li>down - instructs the find closest correction method to search downward in the list when finding a closest match (typically good for deductible amounts)</li> </ul>
id	All	N/A	
liberty	bestMatch	conservative	<p>This attribute is optional. Used by the best match correction method to determine how much liberty to take to find the best match. The following are possible values:</p> <ul style="list-style-type: none"> <li>conservative - instructs the best match correction method to be conservative when attempting to find the best match</li> <li>moderate - instructs the best match correction method to be moderate when attempting to find the best match</li> <li>liberal - instructs the best match correction method to be liberal when attempting to find the best match</li> </ul>
message	All		<p>This attribute is optional. It provides a user friendly textual message that is ultimately associated with the correction associated with a field. There is no default.</p>
optionListReferenceTag	findClosest bestMatch custom		<p>This attribute is optional. It provides an alternate look up list ID for transformation based data corrections, typically involving select list based fields.</p>
source	useSubstitute	defaultValue	<p>The attribute is optional. This instructs the use substitute correction method where to retrieve the substitution data value. Valid values are:</p> <ul style="list-style-type: none"> <li>self - take the substitute data value from an alternate XPath in the source document in context where the value in the correction element supplies that XPath.</li> <li>prototype - take the substitute data value from the same relative location in the related prototype document (if supplied).</li> <li>defaultValue - take the substitute data value from the default value attribute in the TDF</li> </ul>
type	N/A	N/A	<p>This attribute is required and identifies the correction method to run. The following are possible values:</p>

Attribute	Applies to Type	Default Value	Notes
			<ul style="list-style-type: none"> <li>useSubstitute - engages the built-in correction method found in com.agencyport.data.corrections.UseSubstituteCorrector</li> <li>findClosest - engages the built-in correction method implementation found in com.agencyport.data.corrections.FindClosestCorrector</li> <li>bestMatch - engages the built-in correction method implementation found in com.agencyport.data.corrections.BestMatchCorrector</li> <li>delete - engages the built-in correction method implementation found in com.agencyport.data.corrections.DeleteFieldCorrector</li> <li>phone - engages the built-in correction method found in com.agencyport.data.corrections.PhoneNumberNormalizer</li> <li>custom - engages a custom application correction method where the Java class name is found in the className attribute</li> </ul>

#### Correction Logic of Built-In Data Correctors

There are five built-in data corrections. The following explains how they work and what they do:

- UseSubstituteCorrector - This built-in data corrector is typically engaged when a required value is missing and a substitute value can be taken from an alternate data source. The alternate data sources supported by this correction are:
  - The same field housed in the prototype transaction data container instance.

#### Example

If we are on the third vehicle in a data source, it attempts to find the third vehicle in the prototype document. If it is found, the field value for that same field is extracted and used as a substitute value if is not null.

If there are fewer number of aggregate data container instances in the prototype transaction data container than the source, the first aggregate of that classification is used and the field value from that first aggregate is used if not null.

- A different field in the source transaction data container itself. To support this situation where a substitute value may exist somewhere else in the data source itself.
- The default value on the TDF field element (which takes into account DTR) as the substitute value.
- FindClosestCorrector - This built-in corrector can be applied to numeric fields, which are driven as select lists only. This is the class that provides the "bumping and slotting" functionality for coverage and deductible amounts. It supports a direction flag to tell it which direction in the underlying list to search in to find the closest value. The algorithm for determining the closest numeric value is as follows:

Determine whether the direction flag on the correction element is pointing up or down

$\sum_{\text{current Value}} = \text{sum all of the current values}$  (could be a set of splits or a single number)

Set closest entry to null

Initialize running difference to  $\infty$  (Integer.MAX\_VALUE)

For each option list entry

$\sum_{\text{list entry}} = \text{sum all of the values on this option list entry}$  (could be a set of splits or a single number)

difference =  $\sum_{\text{current Value}} - \sum_{\text{list entry}}$

If (direction flag is up AND difference < 0) OR (direction flag is down AND difference > 0 )

difference = absoluteValue(Difference)

If ( difference < running difference )

running difference = difference

closest entry = list entry

```

 endif
 Endif
End for

Return closest entry

• BestMatchCorrector - This built-in corrector supports the matching of text based data values against their supporting recognized set of list entries. A liberty flag governs how much liberty it will take to find a best match. Much of the logic in this class was borrowed from the UploadFilterListOptionManager class. The best match correction with a liberty flag of conservative is the more equivalent of the UploadFilterListOptionManager class. The only difference is auditing, where the best match corrector leaves an audit trail in the correction report. The algorithm for determining the best matching value is as follows:
```

Determine the amount of liberty to take (liberal, moderate, conservative)

Attempt to convert the current data value to a numeric value

For each list entry

Perform a case insensitivity string comparison between the current data value and the list data value. If equal then return this list entry (e.g., Ford matches FORD).

If the current data value has a numeric value then attempt to convert the list entry data value to a numeric value and perform a numeric test. If the numeric test results in an equality then return this list entry (e.g., 007 matches 7.0).

Perform a case insensitivity string comparison between the current data value and the list visual text value. If equal then return this list entry (e.g., FORD matches Ford).

If the liberty governor is ‘conservative’ then advance to the next list entry (the current one doesn’t match)

Perform a case insensitive substring search of the current data value within the list data value and visa versa. If either search results in a hit then return this list entry (e.g., ‘Masonry’ matches Jointed Masonry’).

Perform a case insensitive substring search of the current data value within the list visual text value and visa versa. If either search results in a hit then return this list entry (e.g., ‘Masonry’ matches Jointed Masonry’).

if the liberty governor is ‘moderate’ then advance to the next list entry (the current one doesn’t match)

Perform a case insensitive starts with search between the current data value within the list data value reducing the starts with search down to a minimum of 3 characters. E.g. ‘Chevy’ matches ‘Chevrolet’

Perform a case insensitive starts with search between the current data value within the list visual text value reducing the starts with search down to a minimum of 3 characters (e.g., ‘Chevy’ matches ‘Chevrolet’).

End for

- DeleteFieldCorrector - This built-in corrector simply empties the value associated with a field. It does not delete the associated XML element. It is typically useful as a last resort when other corrections fail.
- PhoneNumberCorrector - This built-in corrector corrects the format of incoming phone numbers in an attempt to satisfy a phone number validator, which is associated to the field. If the fieldElement has a strict phone number validator that is configured to use the built-in US phone number format, or the GLOBAL phone number format, then a phone number corrector would take action to ensure that the incoming phone number format is conformant. The following example illustrates how to configure a corrector that will correct incoming phone numbers to the US phone number format. The phone number correction would not take any action if a US or GLOBAL phone validator is not present in the field.

```
<fieldElement type="text" id="Producer.GeneralPartyInfo.Communications.PhoneInfo[PhoneTypeCd='Phone'].PhoneNumber"
label="Producer Phone" size="14" maxLength="20"
groupId="Producer.GeneralPartyInfo.Communications.PhoneInfo[PhoneTypeCd='Phone']"> <formatMask type="phone" />
<validation type="phone">US</validation> <correction type="phone" /></fieldElement>
```

emptyRosterMessage

This is an optional element to include within a rostertype pageElement. It provides a mechanism to specify the message presented to the user when no items are presented in a roster. If this element is omitted, the default message of “No <ITEM TYPE>s found for this work item” displays. There can be one and only one emptyRosterMessage within a given roster PageElement. There are no attributes for this element.

fieldElement

<b>Attribute</b>	<b>Applies to Type</b>	<b>Default Value</b>	<b>Notes</b>
allowEmptyNode	All	FALSE	When true, disables the automatic empty aggregate management feature from this point downward throughout all sub elements.
checkedDisplayValue	checkbox	N/A	
checkedValue	checkbox	N/A	
cols	textarea	60	The HTML number of columns for a textarea element.
copy	roster	TRUE	When false, disables the field element from being copied on a roster copy action; thereby, engaging the default value registered in the TDF.
defaultValue	text selectlist textarea question hidden filterlist radio date	*****	<p>The default (if any) value of the element. This can also be an ID that implies that the data referred to by this ID will be used as the default value. A view ID can also be used. When the default value specifies an ID, this is known as a "pre-fill." If the type is link, this is used for the URL of the link. There are several built-in keyword defaultValues:</p> <ul style="list-style-type: none"> <li>• <code>GUID()</code> - causes a random guid to generate for the element or attribute value.</li> <li>• <code>SYSTEM_DATE()</code> - takes the current date as the value in the format of YYYY-MM-DD.</li> <li>• <code>SYSTEM_TIME()</code> - takes the current date time stamp as the value in the format of YYYY-MM-DDTHH:MM:SS.</li> </ul>
firstTimeDefaultValue	text selectlist textarea question hidden filterlist radio date	*****	This is an optional attribute that applies only to roster page situations when the roster is empty. This will override the <code>defaultValue</code> only in this case.
formatMask	text date	*****	
groupId	text selectlist textarea question hidden filterlist radio checkbox	N/A	Relates this field element to other field elements that share the same group ID value. This value is used in group level empty aggregate suppression and should be a valid XPath expression within the schema.
hopeless	text selectlist textarea question hidden filterlist radio date checkbox	FALSE	When true, indicates whether any lingering validation issues (from the TVR), regardless of what they are, can be repaired.

Attribute	Applies to Type	Default Value	Notes
htmlDefinitionId	All	N/A	Used to override the default HTML definition template applied. Refer to the <a href="#">Client-Side Development</a> topic for more information.
icon			Valid only if the type is 'link'; use for pictorial (e.g., gif) representation.
id	All	N/A	A unique ID assigned to the fieldElement; must be unique within a pageElement.
label	text selectlist textarea question filterlist radio file column date checkbox	N/A	The text of the field to display on the page.
length			The HTML length of the data that can be entered in the field. If built-in field validation is engaged, then a field validation rule runs to ensure that its field length equals this value.
lookupSource	column	N/A	
maxLength	text textarea column	N/A	<p>The HTML maximum length of the data that can be entered in the field. If built-in field validation is engaged, then a field validation rule runs to ensure that its field length does not extend beyond this value. Additionally, the value set is also applied to the HTML input's maxLength attribute. The majority of browsers will interpret this as an instruction to automatically prevent further data entry once the input's maxLength has been reached.</p> <p><b>Note</b></p> <p>This attribute does not apply for fieldElement type date. In this case, the value of the date.maxLength application property is used (or a default of 10).</p>
minLength			The minimum length of the data that can be entered in the field. If built-in field validation is engaged, then a valid validation rule runs to ensure that its field length does not extend below this value.
questionNumber	question	N/A	The display number of the question in a questionnaire pageElement.
readonly	text selectlist textarea question filterlist radio file date	FALSE	true, TRUE or FALSE. Determines whether the field is read only. Valid for fieldElements type="text" and type="textarea".
required	text selectlist textarea question	FALSE	true, TRUE or False. Determines whether an asterisk (*) should be rendered, indicating that the field is required. If built-in field validation is engaged, then a field validation rule runs to ensure this field is not empty.

Attribute	Applies to Type	Default Value	Notes
	filterlist radio file date checkbox		
rows	textarea	3	The HTML number of rows for a textarea element.
size	text filterlist	Default to maxLength [text], Num rows in filterbox [filterlist]	<p>The HTML size of the data entry field.</p> <p><b>Note</b></p> <p>This attribute does not apply for fieldElement type date. In this case, the value of the date.maxLength application property is used (or a default of 10).</p>
styleClass	All	N/A	
type	N/A	N/A	<p>Denotes the fieldElement type. Valid values are:</p> <ul style="list-style-type: none"> <li>• message</li> <li>• text</li> <li>• selectlist</li> <li>• filterlist</li> <li>• radio</li> <li>• hidden</li> <li>• textarea</li> <li>• question</li> <li>• column</li> <li>• link</li> <li>• checkbox</li> <li>• date</li> <li>• time</li> <li>• file</li> </ul>
uncheckedDisplayValue	checkbox	N/A	
uncheckedValue	checkbox	N/A	
uniqueId	All	N/A	An alternative to ID, unique ID can be used to include competing versions of the same field (with the same 'id' attribute) so that DTR functionality can present the correct version per application condition. It is required that duplicate IDs created in the TDF are disambiguated with the 'uniqueId' attribute, and that only one version be delivered to the browser under any given set of conditions.
viewId	text selectlist textarea question hidden filterlist radio column checkbox	N/A	The view identifier in the format of <code>view_file_name:id</code> value of the view in that file. This will engage the view engine for this field element for both the display and process sides.

Attribute	Applies to Type	Default Value	Notes
downFillFieldId	ALL	N/A	Used by the account management down fill feature to indicate the field in the account transaction to be used as the source for down fill.

#### filterlist

You can define a field within the TDF as a filterlist to allow a user to search and filter from the list of values (refer to the [Search and Filter](#) topic in the User Workflow section for more information on how this works). The following illustrates a defined filterlist:

```
<fieldElement type="filterlist" size="5"
id="WorkCompLineBusiness.WorkCompRateState.WorkCompLocInfo.WorkCompRateClass.RatingClassificationCd" uniqueId="rateClassList"
label="Class Code" required="true"
viewId="workersCompViews.xml:WorkCompLineBusiness.WorkCompRateState.WorkCompLocInfo.WorkCompRateClass.RatingClassificationCd"><optionList reader="xmlreader" source="codeListRef.xml" target="selectOne" /><optionList reader="custom"
source="com.agencyport.workerscomp.custom.ClazzCodeOptionListLoader"
target="WorkCompLineBusiness.WorkCompRateState.WorkCompLocInfo.@LocationRef" /></fieldElement>
```

You can add other similar fields to a transaction in TDF to use the same type of functionality. The above code snippet for the Class Code field in the Workers Compensation LOB is provided out of the box.

The HTML for a filterlist field is defined in the HTML element definitions template file (html\_element\_definitions.txt). You can change the way the field looks on the page by customizing the template and appropriate styles below:

```
*BEGIN FILTERLIST <div class="form-group @styleClass"><label class="col-xs-12 col-sm-3 control-label" for="@elementName"
id="@elementName_label"> @required <span id="@elementName_labelText"
@problemFieldClass>@label </label> <div class="col-xs-12 col-sm-9"> <div
class="selectListContainer"><input type="hidden" value="@value" name="@tdf-id" id="@elementName" style="width: 50%"/> <script
type="text/javascript">ap.@filterElementName = new ap.FilterList("@elementName", @options);</script> @fieldHelpers </div> </div> </div> *END
```

Refer to the HTML Template File section in the [Property File Reference](#) topic for more information on this file.

#### fieldSecurity

When a fieldElement is type="text" or "date", you can add a fieldSecurity child element to make the field secure. This element supports the following attributes:

- encryptAtRest - specifies whether the field's data value should be encrypted at rest while it's being stored in the xmystore.

#### Note

This mechanism for registering encryption fields exists in addition to the application.encryption\_fields property. The ability to specify encryption fields via properties also supports the LOB based property format (e.g., WORK.encryption\_fields or ACCOUNT.encryption\_fields).

- secureDataEntryMode - defines what type of client-side secure data entry should be enabled for this field. The following options are available:
  - disabled - the field behaves like a normal field
  - revealPrefix - a part of the value is secured while a prefix whose length is determined by the numRevealCharacters attribute is shown to the user
  - revealSuffix - a part of the value is secured while a suffix whose length is determined by the numRevealCharacters attribute is shown to the user
  - fullMask - the entire string is secured from view
- numRevealCharacters - the number of characters to reveal to the user if the secureDataEntryMode is revealPrefix or revealSuffix.

Adding copy="false" to the fieldElement of a secured field can be used to prevent the secured field from being copied during a roster item copy.

#### Example

The following excludes the driver's license number from being copied:

```
<fieldElement type="text" id="CommAutoLineBusiness.CommlDriver.DriverInfo.License.LicensePermitNumber" label="Drivers Licens e Number" size="15" maxLength="15" required="true" groupId="CommAutoLineBusiness.CommlDriver.DriverInfo.License" copy="false">
<fieldSecurity encryptAtRest="true" = "revealSuffix" numRevealCharacters="2"/> </fieldElement>
```

For more information on how to configure secure and encryption fields, refer to the [Field Security](#) topic in the Security section. For information on how secure fields displays in the browser, refer to the [Secure Fields](#) topic in the User Workflow section.

#### fieldHelper

These elements provide specialized functionality.

Attribute	Applies to Type	Default Value	Notes
defaultMonth	datePicker	"""""	Valid if the type is set to datePicker. Sets the pop-up's month to the supplied value.
defaultYear	datePicker	"""""	Valid if the type is set to datePicker. Sets the pop-up's year to the supplied value.
format	datePicker		Valid if the type is set to datePicker. Supported date formats are: <ul style="list-style-type: none"> <li>• MM/DD/YYYY (default)</li> <li>• MM/DD/YY</li> <li>• MM-DD-YYYY</li> <li>• YYYY/MM/DD</li> <li>• YYYY-MM-DD</li> <li>• MM-DD-YY</li> <li>• DD/MON/YYYY (e.g. JAN)</li> <li>• DD/MON/YY</li> <li>• DD-MON-YYYY</li> <li>• DD-MON-YY</li> <li>• DD/MONTH/YYYY (e.g. JANUARY)</li> <li>• DD/MONTH/YY</li> <li>• DD-MONTH-YYYY</li> <li>• DD-MONTH-YY</li> <li>• DD/MM/YYYY</li> <li>• DD/MM/YY</li> <li>• DD-MM-YYYY</li> <li>• DD-MM-YY</li> </ul>
function	All	N/A	
icon	All	N/A	
src	script	N/A	
type	N/A	N/A	Denotes the type of helper. Valid values are: <ul style="list-style-type: none"> <li>• datePicker (provides a pop-up calendar for selecting a date)</li> <li>• balloon (creates a help balloon adjacent to the field using the element's value as the text)</li> </ul>
linkText	N/A	N/A	The link text if type is script
htmlDefinitionId	N/A	N/A	Used to override the default HTML definition template applied. Refer to the <a href="#">Client-Side Development</a> topic for more information.

#### importPage

Attribute	Applies to Type	Default Value	Notes
id	N/A	N/A	
pageId	N/A		defaulted to page library values
pageTitle	N/A		defaulted to page library values

#### joinInfo

This is an optional element that provides a mechanism to display items in a roster list that are related via reference IDs to the principle hierarchy. There can be more than one `joinInfo` element.

Attribute	Applies to Type	Default Value	Notes
fromId	N/A	N/A	The name of the attribute that contains the reference ID. This can be thought of as the foreign key value that will be applied to retrieve the correct <code>toPath</code> instance.
fromPath	N/A	N/A	The element path that contains the reference ID.
toId	N/A	N/A	The name of the attribute that contains the ID value that identifies the <code>toPath</code> instance.
toPath	N/A	N/A	

#### optionList

This is a required subelement for a `selectlist`, `filterlist` or `radio fieldElement`.

Attribute	Applies to Type	Default Value	Notes
reader	N/A	xmlreader	A link to the processing code that is going to read the data to display in an <code>optionList</code> .
source	N/A		The source for the options in the <code>optionList</code> .
target	N/A		The name of the collection of options for this <code>optionList</code> .

#### page

Attribute	Applies to Type	Default Value	Notes
allowEmptyNode	All	FALSE	When true, disables the automatic empty aggregate management feature from this point downward throughout all sub elements.
fieldValidations	All	enabled	<p>When its value is "enabled," the built-in field validation engine is engaged for this page (both client side and server side). If this parameter is missing on the page level, then the transaction level <code>fieldValidations</code> setting governs this page. Valid values at this level are:</p> <ul style="list-style-type: none"> <li>• enabled (this value only makes sense if the <code>fieldValidation</code> parameter configured on the</li> </ul>

Attribute	Applies to Type	Default Value	Notes
			<p>transaction is missing or is set to "false," and the desire to turn on built-in field validation for a single page)</p> <ul style="list-style-type: none"> <li>• <code>disabled</code> (turns off built-in SDK client and server side field validation for a given page)</li> <li>• <code>clientSideDisabled</code> (turns off built-in SDK client side field validation for a given page)</li> <li>• <code>serverSideDisabled</code> (turns off built-in SDK server side field validation for a given page)</li> </ul> <p><b>Note</b></p> <p>The values of <code>clientSideEnabled</code> or <code>serverSideEnabled</code> are NOT recognized as valid values.</p>
htmlDefinitionId	All	N/A	Used to override the default HTML definition template used. Refer to the <a href="#">Client-Side Development</a> topic for more information.
id	All	N/A	A unique ID assigned to the page. This must be unique within a transaction definition XML document.
ignorePageVisitedCheck	All	FALSE	
menuDisplay	All	always	A value of "never" suppresses display of this page on the submission navigation menu. A value of <code>even_when_excluded</code> is for DTR based applications when there is a desire to always show the menu item, even if the behavior instructs the system to exclude the page from the transaction.
prevalidate	All	TRUE	<p>When its value is "true," the display side validation is engaged for this page in upload situations for SDK generated pages. If this parameter is missing on the page level, the transaction level pre-validate settings governs this page. Valid values at this level are:</p> <ul style="list-style-type: none"> <li>• true</li> <li>• false</li> <li>• any value not equal to "true" is treated as false</li> </ul>
slowLoader	All	FALSE	
source	roster	N/A	Identifies the storage location in the target data collection schema. This attribute is REQUIRED for roster pages and must be the dot notation path to (including) the top level element of the repeating group. This element is meaningless on other page types.
styleClass	All	N/A	
subType	All	N/A	If the page is a rosterpage of file attachments, must equal "file." Meaningless on other pages.
title	All	N/A	The title of the page to display on the page. This value is also used in the menu navigation bar.

Attribute	Applies to Type	Default Value	Notes
type	N/A	N/A	<p>Denotes the type of the page. Valid values are:</p> <ul style="list-style-type: none"> <li>• dataEntry</li> <li>• roster</li> <li>• page</li> </ul>
validateTransactionOnDisplay	All	FALSE	
downFillPageId	All	Use by account management down fill feature to indicate the page in account transaction to be used as source for down fill.	

#### pageElement

Attribute	Applies to Type	Default Value	Notes
allowEmptyNode	All	FALSE	When true, disables the automatic empty aggregate management feature from this point downward throughout all sub elements.
contentType	script	text/javascript	Usually "text/javascript." Valid only on pageElements of type 'script.'
customRosterReaderClass Name	roster	N/A	A way to override the default roster display mechanism.
dataFilter	roster	N/A	<p>Specifies a simple data include filter list. For simple filtering requirements, this may be enough so that a custom roster reader Java class is not needed. This prescribes the field ID and the value criterion that each rosteritem is evaluated to determine whether the roster item should display.</p> <p><b>Example</b></p> <pre>dataFilter="PersAutoLineBus.PersVeh.ModelYear='2001'  '2002'  '2003'  '2004'  '2005'"</pre> <p>Interpretation: Show the vehicle if its model year is 2001 or 2002 or 2003 or 2004 or 2005; otherwise, don't show it.</p> <p>Note how each value is single quoted and multiple values are delimited with a    (the OR) qualifier. Only equality expressions are supported. Only OR delimiters are supported. If your filtering requirements go beyond the scope of this feature, then consider inheriting from com.agencyport.data.apdata.RosterReader class and overriding either the getNavigationHint or the passesFilterCriterion method.</p>

Attribute	Applies to Type	Default Value	Notes
enableRecover	roster	TRUE	When false, disables the recovery of roster entries on policy change transactions in case the roster entry is held on the original document view.
hopeless	fieldset roster questionnaire	FALSE	When true, indicates whether any lingering validation issues (from the TVR), regardless of what they are, can be repaired.
htmlDefinitionId	All	N/A	Used to override the default HTML definition template used. Refer to the <a href="#">Client-Side Development</a> topic for more information.
id	All	N/A	A unique ID assigned to the <code>PageElement</code> ; must be unique within a transaction definition XML document and is REQUIRED for roster <code>pageElements</code> .
index	index	N/A	
legend	fieldset roster questionnaire	N/A	Legend of the <code>pageElement</code> displayed on the page.
maxEntries	roster		<p>The maximum number of active roster line items that can display on a roster page. This attribute is NOT REQUIRED and can only apply to roster page elements. It defaults to <code>Integer.MAX_VALUE</code>.</p> <p>If the entry is specified, the behavior of the application is as follows:</p> <ul style="list-style-type: none"> <li>Regular add scenario - If the <code>maxEntries</code> threshold is not reached, the user is presented with a data entry fieldset at the bottom of the roster page, along with an Add button each time the page displays. After the user reaches the <code>maxEntries</code> threshold, the data entry fieldset and the Add button are suppressed on display. The user will not be able to add additional roster line items. If the user deletes a roster line item, the next time the page displays, the data entry portion and Add button display.</li> <li>Upload scenario - If the <code>maxEntries</code> attribute is specified and more roster line items are uploaded than the <code>maxEntries</code> threshold, ALL uploaded roster line items display. The data entry fieldset and the Add button are suppressed.</li> </ul> <p>There are several techniques that can be used to have custom messages display to alert the user that the roster is limited. One technique is to use a tips <code>pageElement</code> at the top of the page. The text for the tip is custom. Another technique is to use the messaging infrastructure in place, along with the connector subsystem. A connector can be written to check the number of roster line items and compare that to the <code>maxEntries</code> attribute. A message can be generated by the connector when the threshold is reached.</p>
maxEntriesMessage	roster		The validation message associated with the max entries validation. Defaults to "Number of actual \${pageTitle} detected: \${numEntries} exceeds the maximum number allowed of \${maxEntries}" where <code>pageTitle</code> is replaced with the title of the page, <code>numEntries</code> is replaced with the current number of roster entries and <code>maxEntries</code> is the value configured on the <code>maxEntries</code> attribute. This message is currently only used serverside.

Attribute	Applies to Type	Default Value	Notes
minEntries	roster	0	The minimum number of active roster line items that must display on a roster page. This attribute is NOT REQUIRED and can only apply to roster page elements. It defaults to 0.
minEntriesMessage	roster		The validation message associated with the minEntries validation. Defaults to "Number of actual \${pageTitle} detected: \${numEntries} does not meet the minimum number expected of \${minEntries}" where pageTitle is replaced with the title of the page, numEntries is replaced with the current number of roster entries and minEntries is the value configured on the minEntries attribute. This message is currently only used server side.
relatedIndex	index	N/A	
source	roster	N/A	
src	script	N/A	For use on page elements of type 'script,' a pointer to the target script (e.g., workerscomp/agencyInfoHelper.js).
styleClass	All	N/A	
type	N/A	N/A	<p>Denotes the type of pageElement. Valid values are:</p> <ul style="list-style-type: none"> <li>fieldset - used to specify a related group of data entry fields. This corresponds to the HTML fieldset element.</li> <li>tips - a fieldset that is limited to message fields and hidden fields only.</li> <li>questionnaire - a fieldset that is limited to question fields and textarea fields only.</li> <li>script - used to include a reference to a custom JavaScript file on the page. Refer to the <a href="#">Client-Side Development</a> topic for more information.</li> <li>roster - used to specify column fields and the available roster actions that are to be shown on the roster overview table.</li> <li>index - used on roster pages in cases where the roster source APXPath needs to be predicated wither dynamically or statically. Refer to the <a href="#">Index Management</a> topic for more information.</li> </ul>

#### Transaction

Attribute	Applies to Type	Default Value	Notes
allowEmptyNode	All	FALSE	When true, disables the automatic empty aggregate management feature from this point downward throughout all sub elements.
autoMaintainIdAttributes	All	FALSE	When true, makes APDataCollection internally manage ID attribute values on all ACORD elements that have ID attributes defined as optional (or required). A value of false DOES NOT OVERRIDE a true setting at application scope.

Attribute	Applies to Type	Default Value	Notes
enableAllMenuEntries	All	FALSE	When true, all entries on the menu bar are active, regardless of whether the page has been visited. This is primarily for use when developing a new business application. It should be turned on for policy change based applications.
fieldValidations	All	enabled	When its value is "enabled," the built-in field validation engine is engaged for this transaction (both client and server side).
hasAvailableUI	All	TRUE	
id	All	N/A	A unique ID assigned to the transaction; must be unique within an agency portal application.
lob	All	N/A	The line of business acronym.
nameSourceForWIP	All	N/A	<p>This tells the framework where to get the insured (normally) name that displays on the user's work in progress (WIP) queue for the work item. The following three syntaxes are supported:</p> <ul style="list-style-type: none"> <li>Syntax 1: Assume the name is a commercial (company) type name, stored in the <code>Comm1Name</code> field. The syntax is:</li> </ul> <pre>nameSourceForWIP="Comm1Name".</pre> <ul style="list-style-type: none"> <li>Syntax 2: Assume the name is a personal type name and the desired format is similar to "Bob Smith," and the data store fields are <code>Person.FirstName</code> and <code>Person.LastName</code>. The syntax is:</li> </ul> <pre>nameSourceForWIP="Person.FirstName Person.LastName"</pre> <ul style="list-style-type: none"> <li>Syntax 3: Assume the same personal type name, but the desired display format is something like "Smith, Bob." The syntax is:</li> </ul> <pre>nameSourceForWIP="Person.LastName, Person.FirstName"</pre>
prevalidate	All	N/A	<p>When its value is "true," the display side validation is engaged for this transaction in upload situations for SDK generated pages. Display side validation is enabled. Valid values at this level are:</p> <ul style="list-style-type: none"> <li>true</li> <li>false</li> <li>any value not equal to "true" is treated as false</li> </ul>
relatedTransactionId	All	N/A	Designates the transaction ID to morph a quick quote transaction to when the user selects the Convert to application secondary action.
subType	All	N/A	The sub type of the transaction describes whether the transaction is a package or a mono-line. It is available via the <code>Transaction.getSubType()</code> API. Currently recognized values are:

Attribute	Applies to Type	Default Value	Notes
			<ul style="list-style-type: none"> <li>• cpp (use IWebsharedConstants.CPP_TRANSACTION_SUB_TYPE programatically)</li> <li>• new_business (use IWebsharedConstants.MONO_TRANSACTION_SUB_TYPE programatically)</li> </ul>
summaryPageId	All	N/A	For policy change based transactions, this specifies the ID of the page that supports the secondary action ("Finished with this endorsement").
supportsDefaults	All	TRUE	When true, engages the user preferences subsystem, including the Save to defaults secondary action and the Apply Defaults top frame menu selection.
supportsUploadDataCorrection	All	TRUE	When true, engages data correction at upload writer time.
target	All	N/A	The top element name found in the data collection schema to use for data storage. The name must match a name found in the properties file, under property name DATA_SCHEMAS. In the following portion of the DATA_SCHEMA entry, the corresponding target name would be Sample1: ...xsd[Element:Sample1].
title	All	N/A	The title of the transaction.
type	All	N/A	<p>The type of transaction. The value is available via the Transaction.getType () API and can be used by custom code to determine the type of the current transaction. Currently recognized values are:</p> <ul style="list-style-type: none"> <li>• quick_quote (use IWebsharedConstants.QUICK_QUOTE_TRANSACTION_TYPE programatically)</li> <li>• endorsement (use IWebsharedConstants.ENDORSEMENT_TRANSACTION_TYPE programatically)</li> <li>• new_business (use IWebsharedConstants.NEW_BUSINESS_TRANSACTION_TYPE programatically)</li> <li>• renewal (use IWebsharedConstants.RENEWAL_TRANSACTION_TYPE programatically)</li> <li>• reissue (use IWebsharedConstants.REISSUE_TRANSACTION_TYPE programatically)</li> </ul>
connectorVersion	All	v4	A value of v3 should be used if running 3.X connectors

#### Validation

There can be zero to many validation child elements for a given parent field element. Some of the validation types require the validation element to contain a value, which is used by the validation method as a parameter. The following methods require an element value setting:

- maxLength - integer that provides the maximum length parameter for the test
- minLength - integer that provides the minimum length parameter for the test
- minValue - number that provides the minimum value parameter for the test
- maxValue - number that provides the maximum value parameter for the test

- format - regular expression that provides the pattern parameter to match for the test
- length - integer that provides the length parameter for the test
- custom - Java package class name of the custom class that implements the com.agencyport.fieldvalidation.validators.IValidator interface
- pattern - regular expression - literal syntax to use in text

Attribute	Applies to Type	Default Value	Notes
id	All	N/A	
message	All		This attribute is optional. This provides a way to override the message, which defaults by the validation method.
negate	pattern	FALSE	
type	N/A	N/A	<p>This attribute is required and identifies the validation method to run. The following are possible values:</p> <ul style="list-style-type: none"> <li>• *required</li> <li>• *maxLength</li> <li>• *minLength</li> <li>• phone</li> <li>• ssn</li> <li>• fein</li> <li>• email</li> <li>• minValue</li> <li>• maxValue</li> <li>• ACORDDateFormat</li> <li>• USDateFormat</li> <li>• format</li> <li>• *length</li> <li>• numeric</li> <li>• alphanumeric</li> <li>• USZipCode</li> <li>• verifyListSelection</li> </ul> <p><b>Note</b></p> <p>This method is implicitly always engaged for any select list field element type on the server side only. Therefore, it is not necessary to explicitly configure this validation method in the TDF except in the situation when you want to turn this method off for a given field.</p> <ul style="list-style-type: none"> <li>• custom</li> <li>• pattern</li> </ul> <p>*these validation methods can be configured as fieldElement attributes or as validation element types. The only reason to configure them as validation element types over the field element attribute approach is to render a custom validation message.</p>

# Option Lists

Option lists are generally display-value/data-value pairs driving an HTML selectlist. Good examples of fields where option lists are used are U.S. State or Occupation Type. A bunch of these individual option lists can exist in a collection. A collection, in fact, is just an arbitrary grouping.

Option lists can be classified into the following three types based on where the content is read from:

1. XML based static option list: option list are read in from XML option list field. The optionlist file has to be registered in the productbase as an artifact.

```
<optionList id="billingMethodCd"> <option value="AB">Agency Billed</option> <option value="CAB">Company Account Billed</option> <option value="CPB">Company Policy Billed</option> </optionList>
```

It is referenced in a TDF as shown below:

```
<optionList reader="xmlreader" source="persautoCodeListRef.xml" target="billingMethodCd" />
```

The above produces an HTML select list as shown below:

```
<select id="PaymentBillingMethod" name="PaymentBillingMethod" class="form-control" tabindex="26"> <option value="">Select One</option> <option value="AB">Agency Billed</option> <option selected="" value="CAB">Company Account Billed</option> <option value="CPB">Company Policy Billed</option> </select>
```

2. Dynamic Option List: Dynamic lists are used to build a meaningful caption by concatenating several pieces of data from work item XML.

Dynamic list template is used to configure which data fields are to be concatenated to build the display and return values for the list.

A collection of dynamic templates is defined in the dynamic template list file that is loaded as a productdatabase artifact.

## Example

```
<dynamicListTemplate id="GaragingLocations" title="Garaging Address Dynamic List"> <field type="id">Location.Addr[AddrTypeCd='GaragingAddress'].Addr1</field> <field type="singleSpace"/><field type="id">Location.Addr[AddrTypeCd='GaragingAddress'].City</field> <field type="formatInfo">, </field> <field type="id">Location.Addr[AddrTypeCd='GaragingAddress'].StateProvCd</field> <field type="singleSpace"/><field type="id">Location.Addr[AddrTypeCd='GaragingAddress'].PostalCode</field> <groupId>Location</groupId> <returnValueId>Location.@id</returnValueId> <listBuilderClassName> </listBuilderClassName> </dynamicListTemplate>
```

In the example above, a selectlist is built by concatenating a locations address line 1, city, state code and zip code separated by a comma.

The returned is defined as the ID of the location aggregate. It is referenced in a TDF as shown below:

```
<optionList reader="dynamic" source="persautoDynamicListTemplates.xml" target="GaragingLocations" />
```

The HTML above will look like the following:

```
<select id="VehicleGarageLocation" name="VehicleGarageLocation" class="form-control" tabindex="13"> <option selected value="N125">Tullig Spa, KS 02212</option> <option value="N381">51 Sleeper st Boston, MA 02210</option> </select>
```

3. Custom option list: These types of lists are generated using custom code. The class implementing the custom list has to implement the interface com.agencyport.customlist.ICustomListBuilder.

## Interface methods:

```
/** * Builds a custom list * @param dataManager is the data manager * @param key is the key for the list to be built. * @param existingListToUpdate can be null in which case a new list will be returned. If one is passed in it is appended to. Custom list builders need to accommodate both situations (null and non-null) for maximum efficiency for TVR and otherwise. * @return an option list */ public OptionList generate(DataManager dataManager, OptionList.Key key, OptionList existingListToUpdate);
```

```
/** * Gets the default value for the custom list * @param dataManager is the data manager * @param key is the key for the list to be built. * @param displayList is the option list * @return the default value */ public String getDefaultValue(DataManager dataManager, OptionList.Key key, OptionList displayList);
```

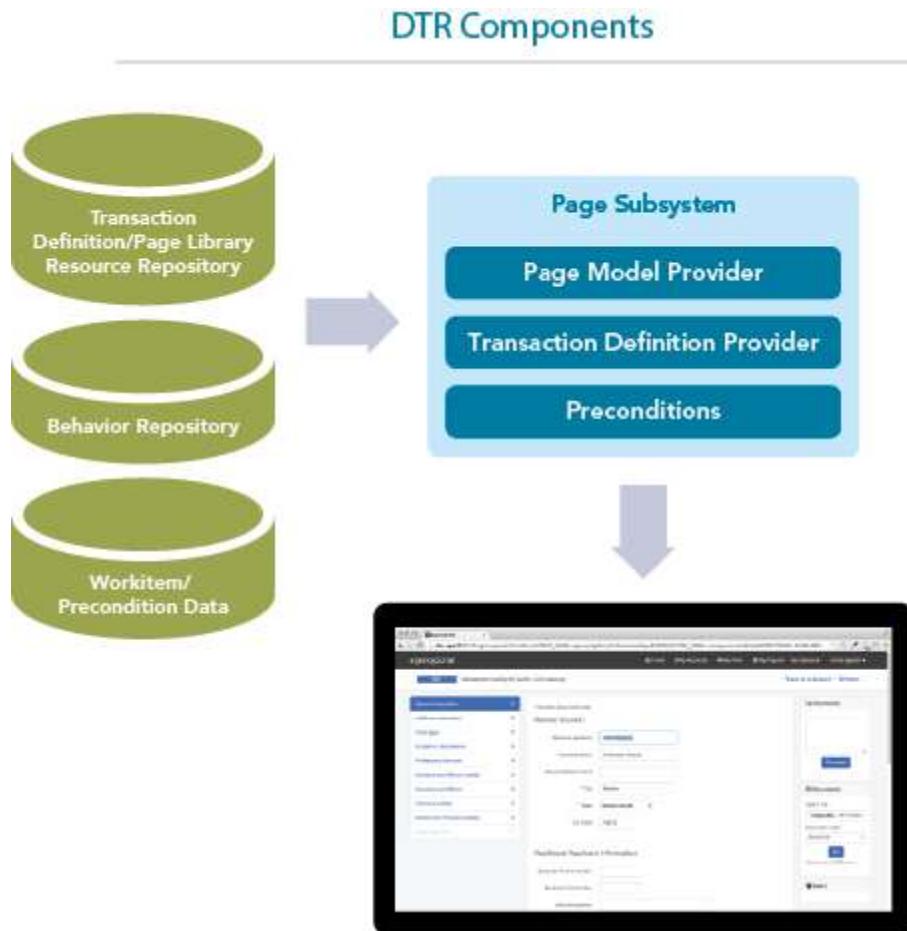
```
<optionList reader="custom" source="com.agencyport.intkit.core.product.LegalEntityBuilder" target="" />
```

# Dynamic Transaction Rendering / Behaviors

Dynamic transaction rendering (DTR) provides the necessary infrastructure in the SDK that serves as the basic foundation for applications to conditionally display a set of page entities, or alter such page entities, and to govern those decisions with a set of preconditions or conditions whose source is typically data from the current work item itself (although other sources are supported). The following lists summarizes the basic features of this subsystem:

- retain the currently established TDF authoring process at Agencyport.
- promote / improve page entity reuse
- support the notion of page entity inclusion, exclusion or alteration based on one or more preconditions/conditions with minimal custom Java code and minimal configuration. The following lists some of the commonly used preconditions:
  - US State
  - Policy Effective Date
  - User Role
  - Company / Carrier
  - Transaction type
  - Quote / New business
  - Endorsement / renewals

The following diagram depicts the major components within DTR:



## Behaviors Repository

The behavior repository contains the "rules" used by the Agencyport transaction definition provider on how to modify a transaction image given a particular set of precondition or conditions. This repository stores whether a page entity is included or excluded, depending on a condition and, if included, optionally, how to alter it from how that page entity is declared in the transaction definition file. For intra-page DTR, the repository also defines the hot field(s) on a page; meaning, the page composition needs to change in some way when a user changes a hot field.

## Transaction Definition Provider

Without the transaction definition provider add-on component, the web page rendered mirrors its respective physical page definition found in the TDF. When engaged, the page engine utilizes the transaction definition provider to transform the page composition, dynamically, by leveraging the behavior repository in conjunction with the current set of preconditions and conditions that are currently in context.

### PreConditions

The PreCondition framework serves as the central go to spot for precondition and condition state. The work item XML is typically the main source for precondition data values that this component serves. An application written LOB PreCondition class extends an SDK base class, tying that implementation to a specific LOB. Its main responsibility is to serve up data values for preconditions that the base PreConditions class itself cannot handle.

The areas managed by the product are the creation and the dispensing of PreConditions instances, loading the preconditions state, installing it on the thread at the entry point of the HTTP request, de-installing it off of the thread close to the end of the HTTP request and serving up of precondition values to the transaction definition provider during page rendering.

### IntraPage Support Module

This module supports the transformation on an SDK page when that page is in context, based on changes the user has made to one or more selected fields. This engages an AJAX trigger, which ends up morphing the current page into something different than it was before the user made the change. The fields that cause these events are called hot fields and are expressed in the behavior repository.

There are several areas of functionality on the server side in support of intra-page DTR that are delegated to the application via the implementation established in an SDK interface called `IIntraPageDTRAssistant`. The area of support here is list content refresh decisions. It assists the Intra-Page Support Module in the following areas:

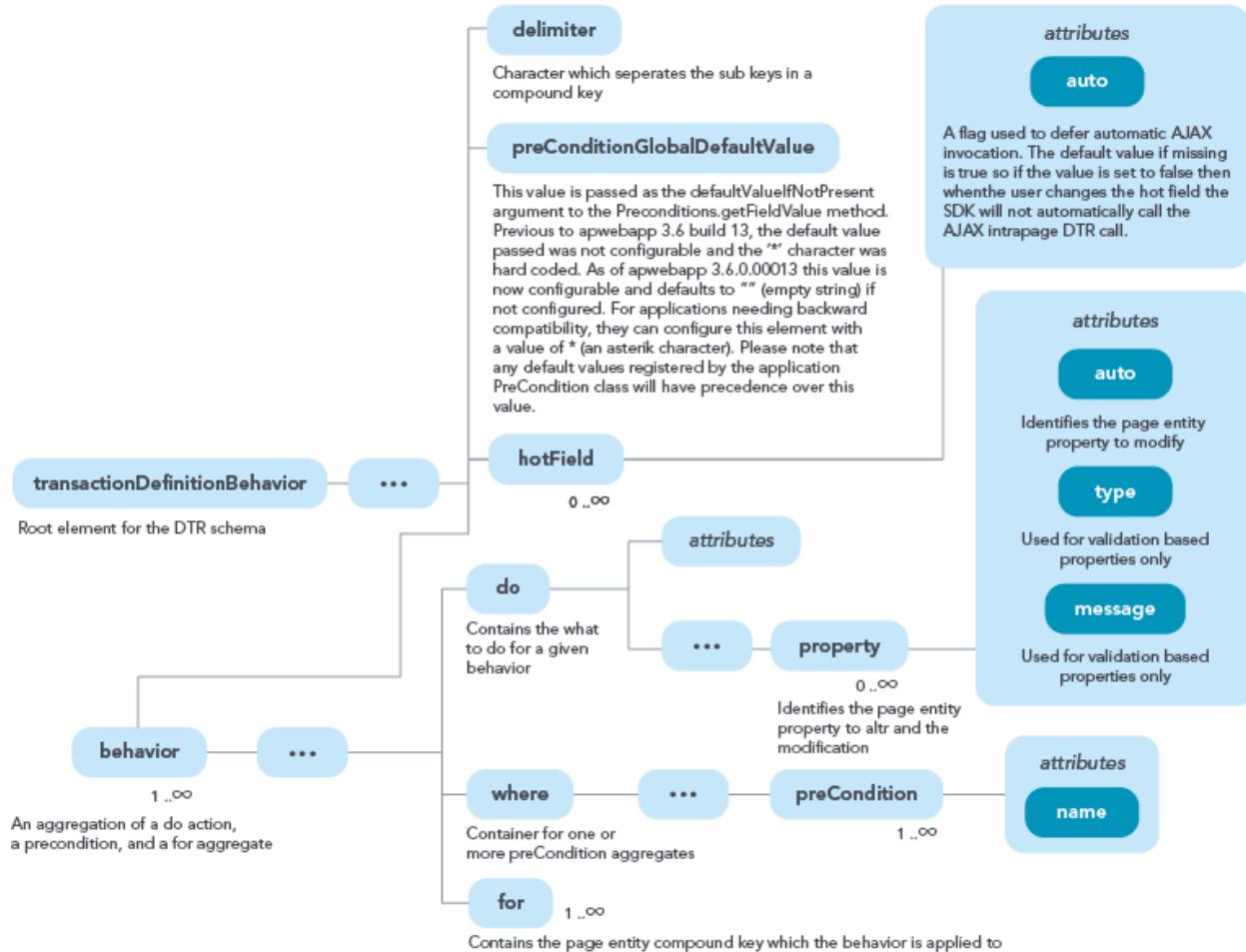
- sets up index management for rosters with multiple repeating hierarchies. Note that this is only necessary for situations when the underlying roster page is missing an index configuration.
- returns a yes/no answer on whether a select list field should have its contents flushed and rebuilt, depending on the orig in hot field. Typically this type of functionality can be handled with the refresh list event configuration on a given hot field.
- the reassignment of a field's values and/or default value for targeted text based fields.

# Behavior XML Syntax

This section describes the XML based syntax that is supported by the built-in SDK TDP.

An application can register one or more behavior files, each containing the behaviors pertaining to one or more transactions (TDF). These behavior files are XML documents that conform to a schema specified in an XML schema with a name of `transactionDefinitionBehavior.xsd`.

The following is a diagram of the `transactionDefinitionBehavior.xsd` schema:



## <hotField>

Each file can contain zero or more hot fields. Hot fields engage the intra-page DTR support module on those pages of a transaction that have one or more hot fields configured.

```

<hotField key="*/vehicle/noFaultCovCoverages/pipCoverage"><refreshListFieldEvent key="*/vehicle/noFaultCovCoverages/funeralLimit"
selectionHint="PRESERVE_CURRENT_USER_VALUE" /><refreshListFieldEvent key="*/vehicle/noFaultCovCoverages/medicalLimit"
selectionHint="PRESERVE_CURRENT_USER_VALUE" /><refreshListFieldEvent
key="*/vehicle/noFaultCovCoverages/tortOptionNoFaultCoverage" selectionHint="PRESERVE_CURRENT_USER_VALUE" />
<refreshListFieldEvent key="*/vehicle/noFaultCovCoverages/deductibleNoFaultCov" selectionHint="PRESERVE_CURRENT_USER_VALUE" />
<refreshListFieldEvent key="*/vehicle/noFaultCovCoverages/additionalPIPILimit" selectionHint="PRESERVE_CURRENT_USER_VALUE" />
<refreshListFieldEvent key="*/vehicle/noFaultCovCoverages/additionalPIPOption" selectionHint="PRESERVE_CURRENT_USER_VALUE" />
</hotField> <hotField assistant="com.agencyport.shared.preconditions.EffectiveDatePDTRAssistant"
key="*/generalInfo/policyTerm/EffectiveDate"/> <hotField key="*/vehicle/vehicleDescription/PersAutoLineBusiness.PersVeh.HotYear"
auto="false" />

```

By default, when a user changes a field that has been configured as hot, an IPDTR AJAX call to the server is automatically invoked. If an application wants to manage that invocation, then a hot field can be marked with `auto="false"` to defer this invocation, leaving the decision and responsibility of calling the AJAX IPDTR up to the application custom JavaScript code. For instance, the hot model field in the above example prevents the SDK's JavaScript from automatically engaging IPDTR when the hot model field is changed.

The following sample application custom code shows how to manually register the `customevent` and then invoke the IPDTR AJAX method:

### Example

```
function onHotYearChanged(){ alert("Calling intrapage DTR AJAX method"); var hotModelField = page.getField("PersAutoLineBusiness.PersVeh.HotModel"); hotModelField.makeIntraPageDTRRequest(); } page.onInitialize = function onCustomInitialize(){ var hotModelField = page.getField("PersAutoLineBusiness.PersVeh.HotModel"); if (hotModelField == null) return; // The following getElements() will return the DOM elements // regardless of the field's current interest level. var elements = hotModelField.getElements(); for (var ix in elements){ var element = elements[ix]; if (element) { registerEvent(element, 'change', onHotYearChanged); } } }
```

The hot field element also allows one or more child `refreshList` event elements where you can enumerate which select lists to refresh on a hot field event, as well as a hint on what to preselect in the list once it is refreshed.

### <behavior>

Each behavior file can contain one to n behaviors whose order is not important to the built-in TDF. Within each behavior, there is always one, and only one, `do` element, optionally one `where` element and one or more `for` elements.

- The `do` element specifies what to do. This is required and only one `do` element is permitted within a behavior.
- The `where` element specifies the condition when to apply this behavior. Multiple `where` elements are ORed together. Multiple precondition elements within a `where` clause are ANDed together. `Where` clauses can also be placed at the beginning of a behavior XML file, which is the same as appropriating that `where` clause on all behaviors within that resource.
- The `for` element(s) specify(ies) the page entities to apply the behavior to.

The following is a sample behavior:

```
<behavior><do action="exclude"/> <where> <preCondition name="StateProvCd">MA</preCondition> </where> <for>*/incident</for>
<for>*/employmentInfo</for> <for>*/priorCoverage</for> <for>*/generalInfo/coapplicantInfo</for>
<for>*/generalInfo/previousAddress</for> <for>*/generalInfo/mailingAddress/yearsAtCurrentAddress</for>
<for>*/generalInfo/mailingAddress/residenceOwnedRented</for> <for>*/generalInfo/paymentPlan/mailPolicyTo</for> <for>*/driver</for>
<for>*/vehicle/vehicleDescription/symbolAgeGroup</for> <for>*/vehicle/vehicleDescription/garagingType</for>
<for>*/vehicle/vehicleDescription/principleRatedDriver</for> <for>*/vehicle/vehicleDescription/vehicleClass</for>
<for>*/vehicle/vehicleUsage/territoryCode</for> <for>*/vehicle/vehicleUsage/distanceOneWay</for>
<for>*/vehicle/vehicleUsage/daysPerWeek</for> <for>*/vehicle/vehicleUsage/numberWeeksMonths</for>
<for>*/vehicle/vehicleUsage/performance</for> <for>*/vehicle/vehicleUsage/vehicleUseCode</for>
<for>*/vehicle/vehicleUsage/carPoolInd</for> <for>*/vehicle/safetyFeatures</for> <for>*/vehicle/nonMACoverages</for>
<for>*/questionnaire</for> </behavior>
```

The above behavior can be read as: *Exclude all of the page entities listed in all of the <for> statements if the precondition value with a field name of "StateProvCd" is equal to 'Ma.'*

### <do> statement

A `do` element contains one active attribute, which can be one of the following interest levels:

- include
- exclude
- alter

If the `action` attribute is equal to alter, then the following page entities are candidates for alteration:

Property Name	Example	Description	SDK API Engaged	Supported by IntraPage DTR
<code>title</code>	<code>&lt;property name="title"&gt;Page title&lt;/property&gt;</code>	Changes the title of the page	<code>Page.setTitle</code>	No

Property Name	Example	Description	SDK API Engaged	Supported by IntraPage DTR
<i>legend</i>	<property name="legend">Legend text</property>	Applies the element text value to the legend of the page element specified in the <for> statement	BasePageElement.setLegend()	Yes
<i>javascript</i>	<property name="javascript">javascript reference</property>	Applies the JavaScript reference to the page element specified in the <for> statement	BasePageElement.setScriptSource()	No
<i>maxEntries</i>	<property name="maxEntries">2</property>	Sets or overrides the max entries attribute on a roster page element	RosterPageElement.setMaxEntries()	No
<i>minEntries</i>	<property name="minEntries">1</property>	Sets or overrides the min entries attribute on a roster page element	RosterPageElement.setMinEntries()	No
<i>dataFilter</i>	<property name="dataFilter">Location.Addr.AddrTypeCd='GaragingAddress'</property>	Sets or overrides the data filter attribute on a roster page element	RosterPageElement.setDataFilterExpression()	No
<i>emptyRosterMessage</i>	<property name="emptyRosterMessage">No locations currently specified</property>	Sets or overrides the empty roster message element on a roster page element	RosterPageElement.setEmptyRosterMessage()	No
<i>enableRecover</i>	<property name="enableRecover">false</property>	Sets the enable recover attribute on the rosterpage element to true or false	RosterPageElement.setEnableRecover()	No

If the action is alter, then the following property alterations for field elements are supported:

<i>message</i>	<property name="message">Message text</property>	Applies the element text as the message text of the field element	MessageElement.setMessage()	Yes
<i>required</i>	<property name="required">true or false</property>	Applies the Boolean value to the required property of the field element	BaseElement.setRequired()	Yes
<i>readonly</i>	<property name="readonly">true or false</property>	Applies the Boolean value to the readonly property of the field element	BaseTextElement.setReadonly() OR BaseListElement.setReadonly()  Depending on the field element type. Another side effect of this is that all data	Yes

			pickers are also removed.	
<i>optionList Reference Tag</i>	<property name="" optionListReferenceTag"> xmlreader:codeListRef.xml:selectOne xmlreader:codeListRef.xml:incidentType </property>	Applies the option list tag to the select list field element	BaseListElement.setOptionListReferenceTag()	Yes
<i>label</i>	<property name="label">label text</property>	Applies the element text to the label of the field element	BaseElement.setLabel()	Yes
<i>helpText</i>	<property name="helpText">help text</property>	Updates an existing, or creates a new, balloon helper with the element text for the given field element	BalloonFieldHelper.setBalloonText()	Yes
<i>defaultValue</i>	<property name="defaultValue">\${StateProvCd}</property>	Updates the default value property on the field element. This example dynamically takes the precondition value with the field name "StateProvCd" and applies that as the default value for the given field element.	BaseElement.setDefaultValue()	Yes
<i>maxLength</i>	<property name="maxLength">max length value</property>	Updates the maxLength property on the text based field element in the element value.	BaseTextElement.setMaxLength()	Yes , validation only
<i>minLength</i>	<property name="minLength">min length value</property>	Updates the minLength property on the text based field element to the element value	BaseTextElement.setMinLength()	Yes , validation only
<i>length</i>	<property name="length">length value</property>	Updates the length property on the text based field element to the element value	BaseTextElement.setLength()	Yes , validation only
<i>validation</i>	Adds a required validation: <pre>&lt;property name="validation"&gt;&lt;validation type="required" message="This field is required"/&gt;&lt;/property&gt;</pre> Removes a validation: <pre>&lt;property name="validation" remove="true"&gt;&lt;validation id="validation id"/&gt;&lt;/property&gt;</pre>	Updates an existing validation element or adds a new one of the given <i>type</i> for the given field element or removes an existing one		Yes

<i>correctio n</i>	<p>Adds a correction:</p> <pre>&lt;property name="correction"&gt;&lt;correction type="phone" /&gt;&lt;/property&gt;</pre> <p>Removes a correction:</p> <pre>&lt;property name="correction" remove="true"&gt;&lt;correction id="correction id" /&gt;&lt;/property&gt;</pre>	Updates an existing correction element or adds a new one of the given <i>type</i> for the given field element or removes an existing one.		Not relevant
<i>formatMa sk</i>	<p>Adds a formatMask:</p> <pre>&lt;property name="formatMask"&gt;&lt;correction type="integer" /&gt;&lt;/property&gt;</pre> <p>Removes a format mask:</p> <pre>&lt;property name="formatMask" remove="true"&gt;&lt;/property&gt;</pre>	Adds a new one of the given <i>type</i> for the given field element or removes an existing one.		Yes
<i>id</i>	<pre>&lt;property name="id"&gt;\${StateProvCd}&lt;/property&gt;</pre>	<p>Substitutes a replacement parameter within an ACORD path with the precondition value of that name:</p> <p>Assume StateProvCd='MA'</p> <p>Before ACORD path:</p> <pre>id="PersAutoLineBusiness.PersDriver.DriverInfo.License[LicenseTypeCd='Driver' &amp;&amp; StateProvCd='\${StateProvCd}'].LicensePermitNumber"</pre> <p>After ACORD path:</p> <pre>PersAutoLineBusiness.PersDriver.DriverInfo.License[LicenseTypeCd='Driver' &amp;&amp; StateProvCd='MA'].LicensePermitNumber</pre> <p>Assume that the field element contains a uniqueId attribute</p>	BaseElement.setId() BaseElement.setElementName()	No
<i>groupid</i>	<pre>&lt;property name="groupId"&gt;\${StateProvCd}&lt;/property&gt;</pre>	Same as "id" item above	BaseElement.setGroupId()	No
<i>styleClass</i>	<pre>&lt;property name="styleClass"&gt;my style class&lt;/property&gt;</pre>	Changes the CSS style class for a message element	MessageElement.setStyleClass()	Yes
<i>Value</i>	<pre>&lt;property name="value"&gt;new value&lt;/property&gt;</pre>	Changes the value of the field	BaseElement.setValue()	Yes

<i>suppressMessageDisplay</i>	<property name="suppressMessageDisplay"/>	Suppresses the connector's messaging	IConnector.suppressMessageDisplay()	No
<i>messageClassFilter</i>	<property name="messageClassFilter">message class filter value</property>	Alters the message class filter on the connector	Connector.setMessageClassFilter	No
<i>xarcRules</i>	<property name="xarcRules"> <xarcRules ruleLibraryId="bopRules" id="BOPRuleCollection" ruleId="EffectiveDate7DaysOut" /> </property>  <property remove="true" name="xarcRules"> <xarcRules id="BOPRuleCollection" /> </property>	Adds or removes an XARC connector	XArcConnector.update or remove	No

#### <where> clause

A where clause can contain one or more precondition sub elements. Each precondition element has a name attribute that specifies the precondition or condition field name that is passed as the first parameter to the `PreCondition.getFieldValue()`. Multiple preconditions are ANDed together. To achieve ORing across preconditions, you can add multiple where clauses.

The following is an example:

```
<behavior> <do action="include"/> <where> <preCondition name="PolicyEffectiveDt">!Date Before('2008-01-01')</preCondition> </where>
<for>*<questionsFor2008</for> </behavior>
```

Read the above set of behaviors as *Include the 2008 questions page if either the policy effective date >=01/01/2008 OR the current user id == agent2.*

#### <for>

Multiple for statements are supported and can also leverage the MATCH\_ANY (\*) or NOT (!) operators on portions of the compound key.

#### Supported Operators

Operator Type	Operator tag	Example	Description	Comment
Equality	<i>Implicit</i>	<where> <preCondition name="StateProvCd">MA</preCondition> </where>	Engage behavior action if the precondition StateProvCd data value is equal to 'MA'	The equality test is the default operator and should not be expressed as ==MA or something of that kind. Also, no quotes should be used in this situation.
In	<code>in()</code>	<where> <preCondition name="StateProvCd">in('MA','FL')</preCondition> </where>	Engage behavior action if the precondition StateProvCd data	The list of values must have single quotes and each item delimited

Operator Type	Operator tag	Example	Description	Comment
			value is equal to either 'MA' or 'FL'	with a comma while the whole list has parenthesis encircling it.
Numeric less than	lt()	<pre>&lt;behavior&gt; &lt;do action="include"/&gt; &lt;where&gt; &lt;preCondition name="NumberOfIncidents"&gt;!lt(2)&lt;/preCondition&gt; &lt;/where&gt; &lt;for&gt;*/questionnaire/Questionnaire/badDriverQuesti on1&lt;/for&gt; &lt;/behavior&gt; &lt;behavior&gt; &lt;do action="exclude"/&gt; &lt;for&gt;*/questionnaire/Questionnaire/badDriverQuesti on1&lt;/for&gt; &lt;/behavior&gt;</pre>	Include the bad driver question if there are three or more incidents on this work item	lt(n), where n is a numeric field
Numeric greater than	gt()	<pre>&lt;behavior&gt;     &lt;do action="include"/&gt; &lt;where&gt; &lt;preCondition name="NumberOfIncidents"&gt;g t(2)&lt;/preCondition&gt; &lt;/where&gt;      &lt;for&gt;*/questionnaire /Questionnaire/badDriverQu estion1&lt;/for&gt;     &lt;/behavior&gt;     &lt;behavior&gt;         &lt;do action="exclude"/&gt;      &lt;for&gt;*/questionnaire /Questionnaire/badDriverQu estion1&lt;/for&gt;     &lt;/behavior&gt;</pre>	Include the bad driver question if there are three or more incidents on this work item.	gt(n), where n is a numeric field
Date equal	DateEqual()	<pre>&lt;where&gt;     &lt;preCondition name="PolicyEffectiveDt"&gt;D ateEqual('2008-01- 01')&lt;/preCondition&gt; &lt;/where&gt;</pre>	Engage behavior action if the precondition PolicyEffectiveDt data value is equal to 01-01-2008	Date must be expressed as YYYY-MM-DD with single quotes and with encircling parenthesis.

<b>Operator Type</b>	<b>Operator tag</b>	<b>Example</b>	<b>Description</b>	<b>Comment</b>
DateBefore	DateBefore()	<where> <preCondition name="PolicyEffectiveDt">DateBefore('2008-01-01')</preCondition> </where>	Engage behavior action if the precondition PolicyEffectiveDt data value is before 01-01-2008	Date must be expressed as YYYY-MM-DD with single quotes and with encircling parenthesis
Date after	DateAfter()	<where> <preCondition name="PolicyEffectiveDt">DateAfter('2008-01-01')</preCondition> </where>	Engage behavior action if the precondition PolicyEffectiveDt data value is after 01-01-2008	Date must be expressed as YYYY-MM-DD with single quotes and with encircling parenthesis.
Match any	*	<for>endorsePersonalAuto/generalInfo/agencyInformation/*</for>	Apply the behavior to all of the field elements in the agency information section on the general info page for endorsements	The '*' can only be used by itself as an operand.
NOT	!	<where> <preCondition name="StateProvCd">!MA</preCondition> </where>	Reverses or negates equality test. The NOT symbol is the '!' character	Can be applied to equality, DataBefore, and In operators

# Behaviors: Built-in Preconditions and Operators

The following is a list of built-in preconditions:

Available built-in precondition	Description
<b>hasDataChanged</b>	Returns true if data has changed on the current page. This precondition can and should be only applied to connectors and instructions and only makes sense on the process side of things.
<b>hasPermission</b>	Returns all of the names of the permissions for all of the permissions that the user has been granted.
<b>inIPDTRMode</b>	Returns true only when an Intrapage DTR AJAX call is in context.
<b>inPreValidationMode</b>	Returns true if pre-validation is currently active. This could be on if a TVR is also running.
<b>inRosterEditMode</b>	Returns true on the display side and not on the process side when a roster is being edited.
<b>inUploadWriterMode</b>	Returns true if in an upload writer context.
<b>isCurrentPageVisited</b>	Returns true if the current page has been visited.
<b>isDataAvailable</b>	Returns true if data is available on the current page.
<b>isUserInRole</b>	Returns true if the current user is in the role group namepates in.
<b>locale_country</b>	Returns the country for the locale currently in context.
<b>locale_language</b>	Returns the language for the locale currently in context.
<b>locale_variant</b>	Returns the variant for the locale currently in context.
<b>recordLockStatus</b>	Returns the record lock status. Possible values are LOCKED_BY_THIS_USER, LOCKED_BY_OTHER_USER, LOCKING_NOT_ACTIVE or UNKNOWN.
<b>statusCode</b>	Returns the status mnemonic of the status code for the current work item.

The following is a list of precondition operations:

Available Operators	Description	XML Example
<b>Is Equal To</b>	Equals the value specified.	<preCondition name="isUserInRole">agent</preCondition>
<b>Is Not Equal To</b>	Not Equal to the value specified.	<preCondition name="isUserInRole">!agent</preCondition>

Available Operators	Description	XML Example
<b>Is In</b>	Is in one of the values specified.	<preCondition name="isUserInRole">in('agent','agentcl','agentpl')</preCondition>
<b>Is Not In</b>	Is not in one of the values specified.	<preCondition name="isUserInRole">!in('agent','agentcl','agentpl')</preCondition>
<b>Is Less Than</b>	Is less than the value specified.	<preCondition name="NumberOfIncidents">lt(2)</preCondition>
<b>Is Not Less Than</b>	Is not less than the value specified.	<preCondition name="NumberOfIncidents">!lt(2)</preCondition>
<b>Is Greater Than</b>	Is greater than the value specified.	<preCondition name="NumberOfIncidents">gt(2)</preCondition>
<b>Is Not Greater Than</b>	Is not greater than the value specified.	<preCondition name="NumberOfIncidents">!gt(2)</preCondition>
<b>Is After the Date</b>	Is after the date specified.	<preCondition name="PolicyEffectiveDt">DateAfter('2008-01-01')</preCondition>
<b>Is Not After the Date</b>	Is not after the date specified.	<preCondition name="PolicyEffectiveDt">!DateAfter('2008-01-01')</preCondition>
<b>Is Before the Date</b>	Is before the date specified.	<preCondition name="PolicyEffectiveDt">DateBefore('2008-01-01')</preCondition>
<b>Is Not Before the Date</b>	Is not before the date specified.	<preCondition name="PolicyEffectiveDt">!DateBefore('2008-01-01')</preCondition>

## Behaviors and Hotfields

### Hotfield

It's possible to implement behaviors to support "if this, show that" type of behavior on an intra-page basis (i.e., real-time). This is done using hotfields. Fields marked as hotfields 'call home' when their values change, the impact is assessed and on the fly; behaviors act accordingly to alter other page entities based on the results. A behavior coupled with a hotfield makes it possible to support a requirement, such as "show the prior address section if the years at current address is less than 3." In this case, an inclusion behavior targeted at the years at current address page entity coupled with a years at current address field marked as hot makes this happen.

### Behavior

An individual behavior does one of three things:

- includes one or more page entities when it fires
- excludes one or more page entities when it fires
- alters one or more page entities when it fires

### Example

For a particular type of user, you may choose to exclude a particular set of fields.

A behavior makes this happen by modifying, at runtime, the superset of fields prescribed in the base transaction definition. Common cases of variability supported by behaviors are variation by U.S. State, User Type or Effective Date.

# Enabling IntraPage DTR

This section explains of the details concerning intrapage Dynamic Transaction Rendering (DTR), especially around implementing an `IIIntraPageDTRAssistant` class and some of the subtleties.

## Operational Assumptions

An IntraPage DTR AJAX call is launched when the user changes the value of a visible field that has been configured as a hot field in the DTR behavior repository. The request is structured as an XML body on the client side to be shipped via HTTP to the server. It is compromised of the following artifacts:

- For each non-excluded field and hot field, a field element contains:
  - unique id
  - interest level
  - hotfield flag
  - origin flag
  - the field's current value
  - all HTTP cookie values associated with the current domain

The following is an example of what is sent to the server during the DTR AJAX:

```
<intraPageDTRRequest><workItemId>1216</workItemId><fieldElement uniqueId="SP.DeleteVehicleTrigger" interestLevel="2"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.@id" interestLevel="2">GUID()</fieldElement><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.VehIdentificationNumber" interestLevel="2"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.HotAssistedEntry" interestLevel="2" isHot="true">1</fieldElement><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.HotYear" interestLevel="2" isHot="true">2007</fieldElement><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.HotMake" interestLevel="2" isOrigin="true" isHot="true">CHEVROLET</fieldElement><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.HotModel" interestLevel="1" isHot="true"><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.HotVehBodyTypeCd" interestLevel="1" isHot="true"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.LeasedVehInd" interestLevel="2"/><fieldElement uniqueId="SP.LeasedPurchaseDate" interestLevel="2"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.NewVehInd" interestLevel="2">1</fieldElement><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.CostNewAmt.Amt" interestLevel="2"/><fieldElement uniqueId="symbolAgeGroup" interestLevel="2"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.@LocationRef" interestLevel="2">A20AE301CB11DFF89F928D3DA71F8C069A</fieldElement><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.Registration.StateProvCd" interestLevel="2">ME</fieldElement><fieldElement uniqueId="garagingType" interestLevel="2"/><fieldElement uniqueId="principleRatedDriver" interestLevel="2"/>ACA1D4CC7E992CCD37D3867048F0BA263A</fieldElement><fieldElement uniqueId="vehicleClass" interestLevel="2"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.QuestionAnswer[QuestionCd='AUPMA07'].YesNoCd" interestLevel="2"/><fieldElement uniqueId="territoryCode" interestLevel="2"/><fieldElement uniqueId="distanceOneWay" interestLevel="2"/><fieldElement uniqueId="daysPerWeek" interestLevel="2"/><fieldElement uniqueId="numberWeeksMonths" interestLevel="2"/><fieldElement uniqueId="performance" interestLevel="2"/><fieldElement uniqueId="vehicleUseCode" interestLevel="2"/><fieldElement uniqueId="carPoolInd" interestLevel="2"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.OdometerReading.NumUnits" interestLevel="2"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.EstimatedAnnualDistance.NumUnits" interestLevel="2"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.AntiTheftDeviceInfo.AntiTheftDeviceCd" interestLevel="2"/><fieldElement uniqueId="SeatBeltTypeCd" interestLevel="2"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.AirBagTypeCd" interestLevel="2"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.AntiLockBrakeCd" interestLevel="2"/><fieldElement uniqueId="LiabilityCoverage" interestLevel="2">100000/300000</fieldElement><fieldElement uniqueId="PropertyDamageCoverage" interestLevel="2">50000</fieldElement><fieldElement uniqueId="MedicalPaymentCoverage" interestLevel="2">5000</fieldElement><fieldElement uniqueId="UninsuredCoverage" interestLevel="2">50000</fieldElement><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.Coverage[CoverageCd='UMPD'].Limit.FormatInteger" interestLevel="2"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.Coverage[CoverageCd='UMPD'].Deductible[DeductibleTypeCd='FL'].FormatInteger" interestLevel="2"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.Coverage[CoverageCd='COMP'].Deductible[DeductibleTypeCd='FL'].FormatInteger" interestLevel="2">500</fieldElement><fieldElement uniqueId="CollisionDeductible" interestLevel="2">250</fieldElement><fieldElement uniqueId="TowingLaborCoverage" interestLevel="2"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.Coverage[CoverageCd='RREIM'].Limit[LimitAppliesToCd='PerDay'].FormatInteger" interestLevel="2"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.Coverage[CoverageCd='RREIM'].Limit[LimitAppliesToCd='MaxAmount'].FormatInteger" interestLevel="2"/><fieldElement uniqueId="PersAutoLineBusiness.PersVeh.HotAdditionalInterestInfoType" interestLevel="2" isHot="true"/><page compoundKey="personalAuto/vehicle" isRoster="true"/><index>PersAutoLineBusiness.PersVeh[2]</index><target>PersAutoPolicyQuoteInqRq</target></intraPageDTRRequest>
```

After the server receives the request, it processes the request as follows:

- Initialization

- o Parses the incoming XML body assumed to conform to the root element: `intraPageDTRRequest` defined in DTR schema and creates a map of `com.agencyport.trandef.intrapage.IntraPageDTRField` instances for each field passed in the request.
- o Initializes the `PreConditions` subsystem and brings that work item's instance into scope.
- o For DTR relating to indexed roster pages and having roster sources comprised of more than one repeating level, the application is required to implement the `IIntraPageDTRAssistant.setupIndexManager()` method. The reason for this is similar to why the application has to establish the index manager on the process side of roster page submissions when there is more than one level of repetition in the roster source. Applications need to refactor existing `NestedAggregateHelper` logic to allow the reuse from both call points.
- o All of the hot field values in the request, regardless of interest level, are mapped into the `PreConditions` instance via the use of `PreConditions.setFieldValue()`. This is the default behavior. If an application desires only the hot fields not currently excluded to be mapped into the `PreConditions` instance, the following application override of the `PreConditions.setFieldValue()` method is in order:

```
public void setFieldValue(String preConditionFieldName, String value, int interestLevel){ if (interestLevel == TransactionDefinitionProvider.INTEREST_LEVEL_EXCLUDE) return; super.setFieldValue(preConditionFieldName, value, interestLevel); }
```

The application's `IIntraPageDTRAssistant.postInitialize()` is called, affording the application to do any custom initialization necessary.

- **JavaScript Response Rendering**

After initialization, the precondition instance is prepared to deliver the precondition/condition data values when the page image is rendered. The page instance for the current web page in context is rendered and evaluated to create dynamic JavaScript, which operates against the JavaScript object model after it's delivered back to the client in the HTTP response body. While the page object model is being evaluated, the application has several opportunities to alter the content of the page when return is given back to the client. The application can modify the values of text fields and the contents of the select lists and related entry. The two methods on the `IIntraPageDTRAssistant` class [`IIntraPageDTRAssistant.reassignValue()` and `IIntraPageDTRAssistant.shouldRefreshListContents()`] serve as the necessary hooks respectively. Change a Text Field: Calculating the expiration date, which is read only, from the effective date. The framework invokes the `IntraPageDTRAssistant.reassignValue()` method on every text field on the page. The application implementation of this method determines which fields it should alter, the conditions under which to alter those fields and then uses the `BaseElement.setValue()` method to set the new value and return a true to signal to the framework that this method has altered the field currently under consideration.

```
public boolean reassignValue(BaseElement baseElement, BaseElement origin, IntraPageDTRBehaviorManager intraPageDTRBehaviorManager) { if (baseElement.getUniqueId().equals("policyExpirationDate")){ if (origin.getUniqueId().equals("PolicyEffectiveDt")){ String effectiveDate = (String) getFieldValue("PolicyEffectiveDt", null); if (!BaseValidator.checkIsEmpty(effectiveDate)){ try { APDate expirationDt = new APDate(effectiveDate); Calendar calendar = expirationDt.get(); calendar.add(Calendar.YEAR, 1); baseElement.setValue(expirationDt.getStringDate(APDate.DATE_FORMAT_MM_DD_YYYY)); return true; // signal to framework that we want the value to be carried to the client } catch (ParseException pe){ } } } } return false; }
```

Change the contents of a select list: Rebuild the vehicle make list if the user changes the vehicle year. This is the old year drives make drives model drives body type use case. If the vehicle year changes, then you need to rebuild the vehicle make, model and body type select lists. The following code is in order: The framework invokes the `IntraPageDTRAssistant.shouldRefreshListContents()` method for every select list and filter list field on the page where the interest level is not excluded. The application returns a true to signal that the contents for a particular list field should rebuild and false to leave the current contents for a list alone.

```
public class PersonalAutoPreConditions extends PreConditions implements IIntraPageDTRAssistant { public static final ElementPathExpression persVehBaseXPath = new ElementPathExpression("PersAutoLineBusiness.PersVeh"); public static final String hotAssistedEntryElementName = "HotAssistedEntry"; public static final String hotYearElementName = "HotYear"; public static final String hotMakeElementName = "HotMake"; public static final String hotModelElementName = "HotModel"; public static final String hotAdditionalInterestType = "HotAdditionalInterestInfoType"; public static final String hotVehBodyTypeCd = "HotVehBodyTypeCd"; private boolean retainValue = false; /* (non-Javadoc) * @see com.agencyport.trandef.intrapage.IIntraPageDTRAssistant#shouldRefreshListContents(com.agencyport.html.elements.BaseListElement, com.agencyport.html.elements.BaseElement, com.agencyport.trandef.intrapage.IntraPageDTRBehaviorManager) * The logic contained in this method instructs the SDK IntraPage framework which lists to refresh depending * on which hot field triggered the request to begin with. */ public boolean shouldRefreshListContents(BaseListElement baseListElement, BaseElement origin, IntraPageDTRBehaviorManager intraPageDTRBehaviorManager){ if (!retainValues){ IntraPageDTRField field = intraPageDTRBehaviorManager.getField(baseListElement.getUniqueId()); if (field == null || field.getCurrentInterestLevel() == TransactionDefinitionProvider.INTEREST_LEVEL_EXCLUDE){ baseListElement.setValue(baseListElement.getDefaultValue()); } } String originId = origin.getUniqueId(); String candidateRefreshId = baseListElement.getUniqueId(); if (originId.endsWith(hotModelElementName)){ return candidateRefreshId.endsWith(hotVehBodyTypeCd); } else if (originId.endsWith(hotMakeElementName)){ return candidateRefreshId.endsWith(hotModelElementName) || candidateRefreshId.endsWith(hotVehBodyTypeCd); } else if (originId.endsWith(hotYearElementName)){ return candidateRefreshId.endsWith(hotMakeElementName) || candidateRefreshId.endsWith(hotModelElementName); } }
```

```

candidateRefreshId.endsWith(hotModelElementName) || candidateRefreshId.endsWith(hotVehBodyTypeCd); } return false; }Other
methods omitted for brevity. }

```

Typical implementation (like the one above) inspects the identity of the hot field that triggered the intrapage DTR AJAX call to begin with (`BaseElement` origin parameter above) to determine the dependent fields. If an application returns a true, then the framework invokes the appropriate list builder(s) registered with that TDF field element. In regards to which entry in the list is initially selected: The basic behavior of the framework preserves previously selected values in lists, where possible, when the list contents are rebuilt.

## Note

If a user selects 2007 for the year and Ford for the make, and then selects 2006, the vehicle make list is rebuilt and the framework, by default, attempts to select the Ford entry in the new vehicle make list as the selected entry if it is present going forward.

If the previously selected entry cannot be resolved, then the default value registered on that TDF field element is used. If the application wants to alter this basic behavior, it can choose to update the current value on the `BaseListElement` to a non-null value, which overrides the framework's attempt to preserve the previously selected entry. This is the reason for the code at the prologue of the method above, which is included for illustrative purposes.

- [How to Gain Access to Hot Field Data from Custom List Builders, etc.](#)

All hot field values are mapped into the precondition instance (via `PreConditions.setFieldValue`) during the server initialization of an intrapage DTR request and are, thereby, available. To get addressability to the `PreConditions` instance, you can either use the `DataManager.getPreConditions()` method if you have addressability to a data manager instance or you can always use `PreConditionsStore.getPreConditions()` if you don't have a data manager in hand. The following is an example of a custom list builder that supports the yeardrives make drives model use case:

```

/* * Created on Mar 30, 2007 by norm AgencyPort Insurance Services, Inc. */ package com.agencyport.custom.persauto; import
java.sql.Connection; import java.sql.PreparedStatement; import java.sql.ResultSet; import java.util.LinkedList; import java.util.List;
import com.agencyport.customlists.ICustomListBuilder; import com.agencyport.data.DataManager; import
com.agencyport.database.DatabaseAgent; import com.agencyport.fieldvalidation.validators.BaseValidator; import
com.agencyport.html.optionutils.ParseValueText; import com.agencyport.logging.ExceptionLogger; import
com.agencyport.preconditions.PersonalAutoPreConditions; import com.agencyport.port.preconditions.PreConditions; import
com.agencyport.utils.PerfObject; import com.agencyport.utils.PerfObjectCollector; import
com.agencyport.webshared.DBCloseObjectMgr; import com.agencyport.webshared.TranManager; /** * The
YearMakeModelListBuilder class supports the year drives make drives model * interface on the vehicle screen. */ public class
YearMakeModelListBuilder implements ICustomListBuilder { /* * The DatabaseAgent is stateless. We only need a single instance that
* can be acquired once and shared across threads/requests. */ static protected DatabaseAgent databaseAgent =
DatabaseAgent.getDatabaseInstance(); private static final String yearSelectStatement = databaseAgent.normalizeSQLStatement("select
distinct year from ${db_table_prefix}yearmake model order by year desc"); private static final String makeSelectStatement =
databaseAgent.normalizeSQLStatement("select distinct make from ${db_table_prefix}yearmake model where year = ? order by make");
private static final String modelSelectStatement = databaseAgent.normalizeSQLStatement("select distinct model from
${db_table_prefix}yearmake model where year = ? and make = ? order by model"); private List queryYearMakeModel(DataManager
dataManager, String listId) { TranManager tranMgr = new TranManager(); Connection dbConnection = null; PreparedStatement pstmt =
null; ResultSet rs = null; List data = new LinkedList(); try { PreConditions pc = dataManager.getPreConditions(); dbConnection =
tranMgr.getConnectionByJNDI(); if (listId.endsWith("year")) { pstmt = dbConnection.prepareStatement(yearSelectStatement); } else
if (listId.endsWith("make")){ String year = (String)
pc.getFieldValue(PersonalAutoPreConditions.hotYearXPath.getExpressionWithOnlyElementHierarchy(), null); if (
!BaseValidator.checkIsEmpty(year)){ pstmt = dbConnection.prepareStatement(makeSelectStatement); pstmt.setInt(1,
Integer.parseInt(year)); } } else if (listId.endsWith("model")){ String year = (String)
pc.getFieldValue(PersonalAutoPreConditions.hotYearXPath.getExpressionWithOnlyElementHierarchy(), null);

if (!BaseValidator.checkIsEmpty(year)){ String make = (String)
pc.getFieldValue(PersonalAutoPreConditions.hotMakeXPath.getExpressionWithOnlyElementHierarchy(), null); if (
!BaseValidator.checkIsEmpty(make)){ pstmt = dbConnection.prepareStatement(modelSelectStatement); pstmt.setInt(1,
Integer.parseInt(year)); pstmt.setString(2, make); } } if (pstmt != null){ rs = pstmt.executeQuery(); while (rs.next()) { String value =
rs.getString(1); String listValue = ParseValueText.createValueTextPair(value, value); data.add(listValue); } } } catch (Exception
exception){ ExceptionLogger.log(exception, YearMakeModelListBuilder.class, "queryYearMakeModel"); } finally {
DBCloseObjectMgr.closeResultSet(rs); DBCloseObjectMgr.closeStatement(pstmt); tranMgr.closeConnection(); } return data; } /* *
(non-Javadoc) * @see com.agencyport.customlists.ICustomListBuilder#generate(com.agencyport.data.DataManager, java.lang.String,
java.util.List) */ public List generate(DataManager dataManager, String customListId, List existingListToUpdate) { PerfObject
perfObject = PerfObjectCollector.startNew(YearMakeModelListBuilder.class.getName() + ".generate"); List contents =
queryYearMakeModel(dataManager, customListId); existingListToUpdate.addAll(contents); perfObject.stop(); return
existingListToUpdate; } /* (non-Javadoc) * @see
com.agencyport.customlists.ICustomListBuilder#getDefaultValue(com.agencyport.data.DataManager, java.lang.String, java.util.List)
*/ public String getDefaultValue(DataManager dataManager, String customListId, List displayList) { int size = displayList.size(); if (
size > 0){ String firstEntry = (String) displayList.get(0); return ParseValueText.getValue(firstEntry); } return ""; } }

```

- [Optimizing/Reducing Intra-page DTR Based Web Content](#)

A feature is available to determine whether an excluded page entity (field set or field) is related to a precondition that is hot field related. If the entity is not related to a hot field precondition, then that entity is physically removed from the server side page image.

If a page entity is excluded and is solely based on precondition data not related to a hot field on that same page, the page entity does not need to be streamed to the browser at all. This optimizes client side loading at the expense of some additional server side overhead.

Target applications that can benefit from this feature are:

- o pages that contain a lot of content that is excluded more often than not
- o applications running under IE 6

To enable this feature, you must be at the SDK level 3.6.0.00030 build or greater, and add the following line to your application properties:

```
optimize_IPDTR_content_size=true
```

Other characteristics include:

- o This feature only works if page entities are related directly to hotfield based preconditions versus indirectly through some other set of preconditions
- o In cases where the product's build-in algorithm for determining whether a page entity is tied to a hot field does not yield the desired behavior (excludes when it shouldn't), applications can always resort to overriding the `TransactionDefinitionProvider.interestedLevelIsRelatedToHotFieldPreConditionOnPage()` method and write a targeted solution. Is OF by default for backward compatibility.
- o Applies at application scope, meaning you can't vary this feature by page or transaction.

# Enhanced DTR Processing

Several improvements to the precondition and behavior processors accompany the AgencyPortal5.1 release. Many of these improvements were based on feedback from services projects collected over a long period of time since the inception of DTR. The overall goals of these improvements is to reduce the amount of custom code necessary to support complex preconditions, to reduce the complexity and number of behaviors while maintaining functional parity, improve the DTR development and debugging experience in general and to improve the run time response time and throughput of the DTR page rendering engine.

## PreCondition Configuration and Synchronization

Since the inception of DTR, preconditions employed an independently managed data schema, access and storage facility from the XML associated with the related work item. Initially, the rationale for this separation was rooted in the assertion that the data values necessary to drive dynamic content were limited to a handful of data entities that could be easily shredded along various points along the transaction workflow. Please remember that previous to the introduction of DTR, applications were fairly limited in their ability to render dynamic content. Unfortunately, this assertion did not play out in the end and this design ultimately led to the following issues:

1. Precondition data values that were out of synchronization with the related work item XML. The `updatePersistableState()` method had to be called to synchronize precondition data with the work item XML data. Application custom code, especially custom connector logic (but not limited), ran into subsequent page rendering issues if updates to the work item XML were not followed up with explicit calls to this method, leading to developer inefficiencies, frustration and ultimately pressure on both product and services technical support groups.
2. Additional runtime overhead was incurred to support separately managed preconditions data storage. This included additional memory, XML parsing, shredding, cloning and database access time.

To address the above issues, the AgencyPortal 5.1 PreConditions subsystem facilitates a mechanism so that preconditions and their related work items could share the same `APDataCollection` instance reference. This would solve the ubiquitous synchronization issue and give substantial boost to performance by eliminating the extra overhead of memory, reading, writing, parsing, shredding and/or cloning the precondition XML.

The conveyance of synchronization intent, as well as other configuration intent, was introduced in AgencyPortal 5 through a set of SDK specific annotations applied to the application's PreCondition extension point class.

## PreConditions Class Annotations

Several annotations have been introduced to support various facets of precondition configuration. The application properties based mechanisms used in 3.x/4.x to convey which TDF transactions a preconditions class supports and the XML schema that it uses have been deprecated. The following is the list of annotations that are supported for preconditions:

- **SynchronizationInfo** - conveys which synchronization mode the SDK should employ to maintain preconditions and work item XML synchronization. If this annotation is missing, then the `USE_DEFAULT` mode is assumed. The following synchronization modes are supported:

<b>DIRECT</b>	Used to instruct the framework to share the same identical <code>APDataCollection</code> instance between the work item XML and precondition instance. This is the most efficient and preferred mode, but requires that the work item XML and the precondition share the XML schema. Under this mode, updates to preconditions are immediately reflected in the work item XML and vice versa. Under this mode, the method <code>updatePersistableState</code> becomes unnecessary and, if called, is merely a no-op. By using this mode, the need for a preconditions database table becomes necessary. This is the default mode of synchronization when the work item XML and the preconditions share the same schema.
<b>DYNAMIC</b>	Used to instruct the framework to dynamically synchronize the precondition data collection in real time as updates are applied to the work item XML data collection, two independently managed data collection instances. Updates applied to the work item XML are immediately picked up in the preconditions using the <code>com.agencyport.domXML.IFieldAccessMonitor</code> mechanism introduced in 4.2. However, updates made to the preconditions are not transferred to the work item XML. This assumes that the XML schemas that support both the work item XML and the preconditions are compatible, although not the same. Any updates applied to the work item XML that fail when applied to the precondition data collection are ignored. Under this mode, like the mode of DIRECT, the method <code>updatePersistableState</code> method becomes unnecessary and, if called, is a no-op. This is the default mode of synchronization when the work item XML and the preconditions do NOT share the same schema.
<b>CLONE</b>	used to instruct the framework to clone the work item XML when the <code>updatePersistableState</code> method is called. This assumes that both the work item XML and preconditions share the same schema. This mode may be appropriate for older applications that overrode <code>updatePersistableState</code> method to avoid the overhead of fine

	grain shredding or other issues encountered with the 3.x/4.x automatic precondition shredding mechanism. When updating 4.x applications, teams are strongly encouraged to use the DIRECT mode instead of this mode.
<b>SHRED</b>	the closest thing to how synchronization was managed by the 3.x/4.x framework. When upgrading 4.x applications, teams are strongly encouraged to use the DIRECT mode instead of this mode. If DIRECT mode cannot be used because preconditions uses a different schema than the work item XML DOM, then DYNAMIC is the next best thing. This mode should be used only as a last resort as a stop gap measure during the process of upgrading a 4.x application to 5.x.
<b>USE_DEFAULT</b>	not really a synchronization mode. This mode merely instructs the framework to select the best fitting mode on its own. The framework will select a realized synchronization mode of DIRECT if the preconditions implementation and work item data collection use the same underlying XML schema. The framework will select a mode of DYNAMIC if the preconditions implementation and work item data collection use different underlying XML schemas.

The following are examples illustrating the use of this annotation:

### Example 1

```
@SynchronizationInfo (mode=SynchronizationMode.USE_DEFAULT) public class HomeownersPreConditions extends PreConditions { }
```

In this example, the homeowners precondition class is instructing the framework to decide itself on the best fitting synchronization mode. Since this preconditions class employs the same XML schema s the work item, due to the absence of a SchemaSupported annotation, the framework will select a mode of DIRECT.

### Example 2

```
@SynchronizationInfo (mode=SynchronizationMode.DYNAMIC) @SchemaSupported("PersonalAutoPreConditions") public class PersonalAutoPreConditions extends PreConditions { }
```

In this example, the personal auto precondition class is instructing the framework to use a synchronization mode of DYNAMIC. The use of a SchemaSupported annotation prescribes a different XML schema, PersonalAutoPreConditions, used by any of the supported transactions (TDFs) and since that mode of DIRECT cannot be used. Had the application used the USE\_DEFAULT mode, the framework would have also selected the DYNAMIC mode.

- **LOBSupported** - The affinity between a precondition class and its supported line(s) of business is conveyed through this annotation. This annotation contains two properties of interest: the lob and the version properties. The lob and version properties are linked to the product.@type and product.@version values in the application's product database, respectively. For all intents and purposes, the type attribute on a product element entry in the product database is the moral equivalent to the LOB designation on related TDFs. For all of the transactions (artifacts of type TDF) within this product type/version combination so designated, a preconditions class is assumed to serve.

### Example

This following example illustrates the use of this annotation:

```
@LOBSupported(lob = "AUTOB", version = @Version({5,0,0,0})) public class CommAutoPreConditions extends PreConditions { }
```

Assuming the following product database configuration:

```
<product type="AUTOB" version="5.0.0.0" title= "Commercial Auto" path="commAuto" description="Commercial Auto" id="N148"> <artifact type="tdf" resourceId="commAuto.xml" title= "New Commercial Auto Application" /> <artifact type="tdf" resourceId="quickQuoteCommAuto.xml" title= "Commercial Auto Quick Quote" /> <artifact type="tdf" resourceId="endorseCommAuto.xml" title= "Commercial Auto Endorsement" /> <artifact type="tdf" resourceId="renewalCommAuto.xml" title= "Commercial Auto Renewal" /> </product>
```

In this example, this preconditions class will support the five versions of these four transactions (TDFs) within the AUTOB product type. Please note that the 3.x/4.x means for relating a precondition's class to one or more transactions via application properties (as shown here) is no longer necessary and is deprecated:

```
commAuto_preconditions_javaclassname=com.agencyport.commAuto.preconditions.CommAutoPreConditions
preconditions_factory_list+=commAuto,CommAutoPolicyQuoteInqRq,${commAuto_preconditions_javaclassname}
preconditions_factory_list+=endorseCommAuto,CommAutoPolicyModRq,${commAuto_preconditions_javaclassname}
preconditions_factory_list+=renewalCommAuto,CommAutoPolicyRenewRq,${commAuto_preconditions_javaclassname}
```

- **TransactionSupported** - The affinity between a precondition class and its supported transaction(s) is conveyed through this annotation. In general, this annotation is not necessary when the LOBSupported annotation is used and only becomes necessary if different precondition classes support different transactions within a single LOB. The case for using this annotation is extremely rare.

### Example

The following example illustrates the use of this annotation:

```
@TransactionSupported(id = "endorseCommIAuto", version = @Version({4,2})) public class CommIAutoEndorsementPreConditions extends PreConditions { }
```

In this example, the commercial auto endorsement preconditions class will only support the 4.2 version of the transaction where the transaction.id = "endorseCommIAuto."

- **SchemaSupported** - provides a way for a precondition extension class to convey the supporting XML schema (data collection name) other than what is expressed by the related transaction's target(transaction.@target). This should only be configured if the precondition class and the work item XML share different XML schemas, which is assumed to be a rare occurrence and should be avoided where possible.

### Example

The following is an example illustrating the use of this annotation:

```
@LOBsSupported({@LOBSupported(lob = "TESTTDP", version = @Version({5,0})), @LOBSupported(lob = "XARCTEST", version = @Version({5,0})), @LOBSupported(lob = "TESTTDP", version = @Version({5,1})), @LOBSupported(lob = "AUTOP", version = @Version({5,0})))} @SynchronizationInfo(mode=SynchronizationMode.DYNAMIC) @SchemaSupported("PersonalAutoPreConditions") public class PersonalAutoPreConditions extends PreConditions { }
```

In this example, the personal auto preconditions class uses its own XML schema distinct from the schema used by the work items.

- **LOBsSupported** - the array form for conveying more than one LOBSupported LOB/version combination.

### Note

The lower case 's,' between the 'B' and 'S' character.

### Example

The following illustrates where two LOBSupported versions are conveyed:

```
@LOBsSupported({@LOBSupported(lob = "HOME", version = @Version({5,0,0,0})), @LOBSupported(lob = "HOME", version = @Version({5,1,0,0}))) public class HomeownersPreConditions extends PreConditions { }
```

In this situation, the application has two versions of the HOME LOB product type. This one HomeownersPreConditions class will serve both products.

- **TransactionsSupported** - the array form for conveying more than one TransactionSupported ID/version combination.

### Note

The lower case 's,' between the 'n' and 'S' character.

### Tip

If you add annotations to a carrier specific PreConditions class, you must update the framework.properties file (or any property file loaded by the system) to let the SDK know where to find your custom PreConditions class if it is not included in com.agencyport (e.g., you placed it under the carrier's package file).

### Example

```
#####
Add the annotation search package path list to include
your carrier specific # package in addition to com.agencyport (with a semi-colon delimiter)
#####
application.annotation_search_package_path=com.agencyport;comprestiege
```

## Note

You can only have one PreConditions class associated with a given LOB. If you add your own custom PreCondition class, you will either have to remove the template version for that LOB or comment out the @LOBSupported annotation in the template class so that the SDK only loads the new PreCondition class. Otherwise, the SDK will not recognize your custom PreCondition class.

## PreCondition getFieldValue Optimization

Analysis of past performance tests has shown that there was an opportunity to improve the response time of this method. Depending on the number and complexity of behaviors involved, this method can be called hundreds, if not thousands, of times during the course of page rendering - a real hot spot. Several applications have built caching harnesses into their application's custom precondition getFieldValue mechanism.

As of AgencyPortal 5.1, a robust caching mechanism has been added to the base PreCondition class to optimize the performance of this method. The key into each cached value is a composite key comprised of three items: the precondition field name, its related int [] index, and the APDataCollection view type in context when the call is being made. The cache is a HashMap that is built up as calls to getFieldValue are made by the DTR subsystem. The cache is cleared on the following events:

1. Any time the IndexManager.setCurrentIdArray method is called.
2. Any time any of the following PreConditions class methods are called:
  - clearCache
  - attach
  - clone
  - setFieldValue
  - shred
  - synchronize
  - updatePeristableState

Applications can utilize the same caching mechanism for derived values that are calculated by customJava code. In the following example, a custom implementation of the getFieldValue illustrates its proper usage:

```
public Object getFieldValue(String preConditionFieldName, String defaultValueIfNotPresent, WhereClause whereClause) { if(preConditionFieldName.equals(ICommAutoConstants.COMML_RATE_STATE_PROV_CD_XPATH)) { IndexManager indexManager = getIndexManager(); int[] currentIdArray = null; if(indexManager != null) { currentIdArray = indexManager.getCurrentIdArray(ICommAutoConstants.COMML_RATE_STATE); } PreConditionsCacheKey cacheKey = PreConditionsCacheKey.create(preConditionFieldName, currentIdArray, this); String stateProvCd = (String) this.getCacheValue(cacheKey); if (StringUtilities.isEmpty(stateProvCd)){ // grab that state assoiated with the selected location index and return String locationIndex = (String) getFieldValue(ICommAutoConstants.COMML_VEHICLE_XPATH + ".HotGaragedLocationRef", ""); String stateProvCdXPath = "Location[@id=" + locationIndex + "].Addr.StateProvCd"; stateProvCd = data.getFieldValue(stateProvCdXPath, ""); this.putCacheValue(cacheKey, stateProvCd); } return stateProvCd; } else { return super.getFieldValue(preConditionFieldName, defaultValueIfNotPresent, whereClause); } }
```

## Public APIs

Several notable changes regarding the PreConditions class and supporting classes were added in AgencyPortal 5:

- To fix a long standing support issue relating to IPDTR and the exclusion of hot fields containing values, the PreConditions.setFieldValue method's base implementation was changed to clear the data value in preconditions for any hot field that has been excluded due to a prior IPDTR call. Previous to this, applications may have needed to override this method and apply special logic when the interested level = exclude.
- In support of new synchronization modes:
  - The com.agencyport.preconditions.SynchronizationMode enumeration contains all of the supported synchronization modes.
  - The PreConditions.attach method, a new method used by the framework to make the work item APDataCollection instance known to the PreConditions instance.
  - The PreConditions.get/setSynchronizationMode methods provide access to get or set the synchronization mode active for a given PreConditions instance. Only under rare cases should the application code ever need to call the set method.
  - The PreConditionsStore.read method has been fully deprecated. Any custom code using this method will need to be identified, re-mediated and changed to use the new PreConditions.acquire method during the upgrade process.
  - The PreConditionsStore.acquire method is the functional equivalent to what used to be handled by the deprecated method PreConditionsStore.read.
- In support of precondition caching optimization:
  - The com.agencyport.preconditions.PreConditionsCacheKey class is the type for the key into the getFieldValue cache.
  - The PreConditions.createCacheKey method creates a PreConditionsCacheKey given a precondition field name.

- o The `PreConditions.getCacheValue` method returns the cached value for the given key. Null is returned if none is currently found in the cache.
- o The `PreConditions.putCacheValue` method adds/updates the value for the given key.
- o The `PreConditions.clearCache` method should be used to clear the cache. Typically, the application does not need to call this.

## Behavior Configuration and Interoperability

One of the reasons custom precondition `getFieldValue` logic was historically needed is driven by the fact that many conditions have relatively complex logic (at least logic that supersedes a simple value retrieval from the `APDataCollection` store that the preconditions class is managing). Many applications in the past customized their `Preconditions.getFieldValue` methods to support W3C XPath. As of AgencyPortal 5.1, W3C 1.0 XPATH using Jaxen is supported on behavior where clauses on both the precondition name attribute and on the text value on the precondition element itself. This feature should eliminate the need to implement custom Java code for many of the derived preconditions.

XPath processing has been enhanced with several custom XPATH functions to supplement date and index management functionality. XML behavior markup has also been upgraded to support behaviors that are associated with a particular grouping or segmentation of content known as "classes." The hope of this feature is to reduce the number and complexity of behaviors.

Another feature introduced allows applications to control the relative weighting a behavior should be given. In the past, applications may have needed to make some exclusions of content supersede explicit inclusions under certain circumstances. Their only resource at the time was to write a custom TDP. Using names, such as "uber" or "super" excludes were attributed to this pattern by some. Now, with this feature, there is no longer a need to write a custom TDP just to have control over behavior weighting.

Lastly, behavior debugging has been extremely difficult in the past. A new behavior markup syntax to engage debugging output was designed to provide the developer with a log into how an interest level for one or more page entities is derived.

## Custom TransactionDefinitionProvider Class Annotations

Several annotations have been introduced to support custom transaction provider configuration. The application properties based mechanisms used in 3.x/4.x to convey which TDF transactions a TDP class supports. The following is a list of annotations that are supported for preconditions:

- **LOBSupported** - The affinity between a custom TDP class and its supported line(s) of business is conveyed through this annotation. This annotation contains two properties of interest: the LOB and the version properties. The LOB and version properties link to the `product.@type` and `product.@version` values in the application's product database respectively. For all intents and purposes, the type attribute on a product element entry in the product database is the moral equivalent to the LOB designated on related TDFs. For all of the transactions (artifacts of type TDF) within this product type/version combination so designated, a custom TDP class is assumed to serve.

### Example

The following illustrates the use of this annotation:

```
@LOBSupported(lob = "WORK", version = @Version({5,0,0,0})) public class WorkersCompTDP extends
XMLTransactionDefinitionProvider { }
```

Similar as to how the `LOBSupported` annotation pertains to the `PreConditions` applies here as well.

- **TransactionSupported** - The affinity between the custom TDP class and its supported transaction(s) is conveyed through this annotation. In general, this annotation is not necessary when the `LOBSupported` annotation is used and only becomes necessary if different TDP classes support different transactions within a single LOB. The case for using this annotation is extremely rare:

### Example

The following illustrates the use of this annotation:

```
@TransactionSupported(id = "workersComp", version = @Version({4,2})) public class WorkersCompTDP extends
XMLTransactionDefinitionProvider { }
```

In this example, the workers compensation custom TDP class will only support the 4.2 version of the transaction where the `transaction.@id` = "workersComp."

- **LOBsSupported** - the array form for conveying more than one `LOBSupported` LOB/version combination.
- **TransactionsSupported** - the array form for conveying more than one `TransactionSupported` ID/version combination.

## Behavior XML Markup to Support W3C XPath Evaluation on Where Clauses

W3C XPath statements are distinguished from other instructions by encompassing the XPath statement within an XPath{} sequence.

### Example

```
<behavior id= "includeOBELCoverageLimit" title ="Include OBEL Limit if in NY AND OBEL = 'Yes'"> <do action ="include" /><where>
<preCondition name ="HotRateState"> in('NY')</ preCondition> <preCondition name ="XPATH{
count(Comm1AutoLineBusiness/Comm1RateState[StateProvCd='NY']/Comm1Veh/Comm1Coverage[CoverageCd='OBEL']) }">gt(0)
</preCondition > </where > <for> */optionalCoverages/supplementalCoverages/obelCoverageLimit</for > </behavior>
```

It is important to mention that the XPath statement within an XPath{} sequence must be 100% W3C XPath and cannot contain TDF field element unique IDs. The framework makes no adjustments to the XPath expression.

The following functions are provided to facilitate better date handling and index management processing. All functions are in namespace under the URL of <http://www.agencyport.com>. The prefix to use for this is ap. XPath used in XARC Rules also now shares the use of these same XPath custom functions supplied by the framework:

Function	Description	Sample
ap:dateAdd	<p>Adds a duration argument to a date argument and returns the calculated date. The duration argument is in the format conforming to the XPath date period as in PyYmMdD where P, Y, M and D are fixed constants, while y is the number of years, m is the number of months and d is the number of days. The date argument is assumed to be a valid date in the yyyy-MM-dd format. It can also be one of the following values for designating current relative date values:</p> <ol style="list-style-type: none"> <li>1. #today</li> <li>2. #yesterday</li> <li>3. #tomorrow</li> </ol> <p>You can use negative numbers to designate subtraction.</p>	<pre>ap:dateAdd(AccidentViolationDt, 'P3Y0M0D')  - add 3 years to the accident violation date element</pre>
ap:dateAfter	Evaluates two yyyy-MM-dd date arguments and returns true if the first one is after the second.	<pre>ap:dateAfter(ap:dateAdd(Accident ViolationDt, 'P3Y0M0D'), '#today')  - add 3 years to the accident violation date and return true if that result falls after today</pre>
ap:dateBefore	Evaluates two yyyy-MM-dd date arguments and returns true if the first one is before the second.	<pre>ap:dateBefore(ap:dateAdd(Acciden tViolationDt, 'P3Y0M0D'), '#today')  - add 3 years to the accident violation date and return true if that result falls before today</pre>
ap:dateEquals	Evaluates two yyyy-MM-dd date arguments and returns true if the first one equals the second.	<pre>ap:dateEquals(EffectiveDt, TransactionEffectiveDt)  - compare the effective date of the policy and the transaction effective date returning true if they are equal</pre>

Function	Description	Sample
ap:dateAfterOrEquals	Evaluates two yyyy-MM-dd date arguments and returns true if the first one is equal to or later than the second.	
ap:dateBeforeOrEquals	Evaluates two yyyy-MM-dd date arguments and returns true if the first one is equal to or before the second.	
ap:currentPosition	Useful for writing XPaths that need to focus in on a specific XML entity most likely linked to the current entity in context. Accepts an AP AXE XPath argument (note one with the '.' dot delimiter) and returns the index of that argument normalized to a 1 based position().	<pre>sum(PersPolicy/AccidentViolation [@DriverRef = //PersAutoLineBusiness/PersDriver[position() = ap:currentPosition('PersAutoLineBusiness.PersDriver')]/@id]/DamageTotalAmt/Amt)  - determine the current position (1 based) of the current driver that is currently in context and sum up all of the violation amounts for that particular driver</pre>
ap:getVar	A corollary to ap:currentPosition. For each repeating element in the current IndexManager instance for the in flight request, this provides access to the @id attribute value of each of those specific repeating elements currently in context. The naming pattern used for the single parameter that this custom function accepts is <element name>Id.	<pre>//PersAutoLineBusiness/PersDriver[@id = ap:getVar('ap:PersDriverId')]  - For the driver whose id attribute value matches the variable named PersDriverId which is initialized with the PersDriver.@id value for the current driver in context in index manager</pre>

## Where Clauses Applied at Behavior XML Resource Scope

Historically, where clauses could only appear within a <behavior> statement. As of AgencyPortal 5.1, you can configure a where clause at the beginning of a behavior resource. A where clause at this scope is functionally equivalent to configuring that same where clause upon each behavior within the resource. If you segment your behaviors into resources by a common condition, then you can dramatically cut down on the number of redundant where clauses that are required.

### Example

The following is an example illustrating the statement above:

Relevant snipped from a behavior file called myBehaviors.xml:

```
<transactionDefinitionBehavior xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://reference.agencyport.com/schemas/5.0/behavior/transactionDefinitionBehavior.xsd" ><!-- the
following where clause will be applied to all behavior entries in this resource --> <where > <preCondition name ="StateProvCd">MA</
preCondition> </where > </transactionDefinitionBehavior>
```

For all behaviors configured in myBehaviors.xml, the above where clause will be added.

### ORing Where Clause Capability

Previous to this feature, multiple behaviors with the same do.@action and <for> statements with different where clauses had to be configured in order to achieve an ORed outcome. As of AgencyPortal 5.1, multiple where clauses can be applied within one behavior statement. This can reduce the number of behaviors for applications that follow this pattern.

```

<behavior title="Driver level at risk question: If the total
damage amount across all accidents for the current driver in context
within the last 3 years was greater than $3500.00 for any driver over
the age of 25 OR greater than $2000.00 for drivers younger than 25
years of age"
 debug= "true" debugFilter ="*/**/driverAtRisk" >
 <do action ="alter" >
 <property name ="required">true</ property>
 </do >
 <where>
 <!-- Make the driver at risk field required if
there are any accidents within the last 3 years totaling over 3500
dollars -->
 <preCondition
 name= "XPath{
sum(PersPolicy/AccidentViolation[ap:dateAfter(ap:dateAdd(AccidentViola
tionDt, 'P3Y0M0D'), '#today') and
@DriverRef =
//PersAutoLineBusiness/PersDriver[position() =
ap:currentPosition('PersAutoLineBusiness.PersDriver')]/@id]/DamageTota
lAmt/Amt) }" >gt(3500.00)</ preCondition>
 </where >
 <!-- OR -->
 <where>
 <!-- Make the driver at risk field required if
there are nny accidents by drivers 25 years or less within the last 3
years totaling over 2000 dollars -->
 <preCondition
 name="XPath{
sum(PersPolicy/AccidentViolation[ap:dateAfter(ap:dateAdd(AccidentViola
tionDt, 'P3Y0M0D'), '#today') and
@DriverRef = ap:getVar('ap:PersDriverId') and
ap:dateAfter(ap:dateAdd(/PersAutoLineBusiness/PersDriver[@id =
ap:getVar('ap:PersDriverId')] /DriverInfo/PersonInfo/BirthDt, '25y'),
'#today')]/DamageTotalAmt/Amt) }" >gt(2000.00)</ preCondition>
 </where >
 <for >*/**/driverAtRisk </for >
 </behavior >

```

## Applying Behaviors to Content within a Shared Class

With 3.x/4.x, the page entity targets of all behaviors were limited to one or more `<for>` statements, each prescribing the unique IDs of the designated page content. The asterisk (\*) wild card character helped this go a long way. Now with 5, page entities can be grouped or segmented into classes of content using the new `class` attribute. Behavior `<for>` statements can then be written against these classes. Classifications of content can be at `page`, `pageElement` or `fieldElement` scope. A class attribute can contain one or more tokens where tokens are space delimited. On the `<for>` statements, classes are distinguished from typical TDF unique IDs since they begin with the dot(.) character. This feature should help reduce the amount of behavior content and simplify behavior authoring/debugging life cycle and, perhaps, revolutionize how behaviors are conceptualized.

### Example

This example leverages the classification of behaviors into US state segments:

```

<fieldElement id="f1" class="StateSpecific CT" /> <fieldElement id="f2" class="StateSpecific CT" /> <behavior> <do action ="exclude" />
<for >*.StateSpecific </for > <for >*/*.StateSpecific </for > <for >*/*/*.StateSpecific </for > </behavior> <behavior> <do action ="include" />
<for >*/.CT </for > <for >*/*.CT </for > <for >*/*/*.CT </for > </behavior>

```

In the above example, when the US state is anything other than CT, both fields are excluded.

#### Note

The starting dot in the `<for>` command distinguishes those tokens as class directives versus unique ID specifications.

#### Controlling Relative Weighting of Behaviors

Normally, the default weighting of behaviors during conflict resolution does not need to be altered and is sufficient for most situations. There may arise situations where behaviors that exclude content should supersede behaviors that explicitly include content under certain conditions. Weighting is conveyed numerically with the bigger the number, the more weight. The default numerical weight values assigned to the interest levels are as follows:

- implicit include - 0
- alter - 10
- exclude - 20
- explicit include - 30

Given the above default weight assignments, if both of the following behaviors were to evaluate to true, the inclusion behavior would win over the exclusion behavior. This may not be the desired outcome.

```

<behavior> <do action = "exclude" /> <where> <preCondition name = "Program">!UPSCALE</preCondition> </where> <for >*/content
</for> </behavior> <behavior> <do action = "include" /> <where> <preCondition name = "StateProvCd">in('NY', 'PA, 'NJ')</preCondition>
</where> <for >*/content </for> </behavior>

```

A business requirement might exist, stipulating that if the insurance program selected is not equal to the value 'UPSCALE' then the content should be excluded, regardless of any other conditions that might bring it into view. Hence, the notion of "uber" exclude. As of AgencyPortal 5.1, behaviors can have a weight attribute wherein the author can override the default weight value that is assigned to that action/interest level.

```

<behavior weight="40" >
 <do action = "exclude" />

 <where>
 <preCondition name = "Program">!UPSCALE</preCondition>
 </where>
 <for >*/content </for>
</behavior>

<behavior>
 <do action = "include" />

 <where>
 <preCondition name = "StateProvCd">in('NY', 'PA,
'NJ')</preCondition>
 </where>
 <for >*/content </for>
</behavior>

```

#### Debugging Experience

Additional behavior markup has been added to allow the developer to turn on debugging at the behavior level. There is a debug Boolean flag that, when true, turns on the debugging feature for the specific behavior. There is also an optional `debugFilter` that works similarly to a `<for>` statement, allowing the debug output to only be applicable for the page entity expression; thereby, cutting down on the "noise" in the log.

There is an additional application properties Boolean flag called `application.behavior_debugging_flag` that must also be turned to true in order for the debug output to become active. This allows debug markup to remain on behaviors even in production, but can be all turned off with one application property flag so it does not impede runtime performance.

## Example

The following is a sample of the type of debugging output generated. In the following example, the `StateProvCd` value was MA.

The first evaluation is determining the applicability of the behavior to the `speedingCoverage` field element. Note that without the `debugFilter` there would be a lot of noise here for each field on this page in the TDF.

The following evaluation returns a true meaning this behavior is applicable to the content specified in the `<for>` statement:

```
Apr 24, 2014 12:23:53 PM com.agencyport.trandef.behavior.BehaviorComparator compare

INFO: Preliminary evaluation of behavior: com.agencyport.trandef.behavior.BehaviorEntry [id=N10,
key=~/vehicle/nonMACoverages/speedingCoverage, behavior=INTEREST_LEVEL_INCLUDE, weight=30 (set to default),
where clauses=[com.agencyport.trandef.behavior.WhereClause[name=StateProvCd, value=CT,
isSetRequested=false, isCountRequested=false, xPath=null],
com.agencyport.trandef.behavior.WhereClause[name=XPATH{ PersPolicy/ContractTerm/EffectiveDt },
value=DateAfter('2006-12-31'), isSetRequested=false, isCountRequested=false,
xPath=XPathExpression [xPathExpression= PersPolicy/ContractTerm/EffectiveDt]],
com.agencyport.trandef.behavior.WhereClause[name=XPATH{ PersPolicy/ContractTerm/EffectiveDt },
value=DateBefore(#today), isSetRequested=false, isCountRequested=false, xPath=XPathExpression
[xPathExpression= PersPolicy/ContractTerm/EffectiveDt]],
com.agencyport.trandef.behavior.WhereClause[name=XPATH{ PersPolicy/ContractTerm/EffectiveDt },
value=DateEqual(XPATH{PersPolicy/ContractTerm/EffectiveDt}), isSetRequested=false,
isCountRequested=false, xPath=XPathExpression [xPathExpression=
PersPolicy/ContractTerm/EffectiveDt]], com.agencyport.trandef.behavior.WhereClause[name=XPATH{
substring(PersPolicy/ContractTerm/EffectiveDt, 1, 4) }, value=XPATH{
substring(PersPolicy/ContractTerm/EffectiveDt, 1, 4) }, isSetRequested=false,
isCountRequested=false, xPath=XPathExpression [xPathExpression=
substring(PersPolicy/ContractTerm/EffectiveDt, 1, 4)]]] against the current evaluation key:
tdpTest/vehicle/nonMACoverages/speedingCoverage evaluated to true
```

The second evaluation is engaged only if the first evaluation is true; this is where precondition evaluation comes into play.

```
Apr 24, 2014 12:23:53 PM com.agencyport.trandef.behavior.BehaviorComparator compare
```

```
INFO: Secondary evaluation of behavior: com.agencyport.trandef.behavior.BehaviorEntry [id=N10,
key=~/vehicle/nonMACoverages/speedingCoverage, behavior=INTEREST_LEVEL_INCLUDE, weight=30 (set to default),
where clauses=[com.agencyport.trandef.behavior.WhereClause[name=StateProvCd, value=CT, isSetRequested=false, isCountRequested=false,
xPath=null], com.agencyport.trandef.behavior.WhereClause[name=XPATH{ PersPolicy/ContractTerm/EffectiveDt },
value=DateAfter('2006-12-31'), isSetRequested=false, isCountRequested=false, xPath=XPathExpression [xPathExpression= PersPolicy/ContractTerm/EffectiveDt]],
com.agencyport.trandef.behavior.WhereClause[name=XPATH{ PersPolicy/ContractTerm/EffectiveDt },
value=DateBefore(#today), isSetRequested=false, isCountRequested=false, xPath=XPathExpression [xPathExpression= PersPolicy/ContractTerm/EffectiveDt]],
com.agencyport.trandef.behavior.WhereClause[name=XPATH{ PersPolicy/ContractTerm/EffectiveDt },
value=DateEqual(XPATH{PersPolicy/ContractTerm/EffectiveDt}), isSetRequested=false, isCountRequested=false, xPath=XPathExpression
[xPathExpression= PersPolicy/ContractTerm/EffectiveDt]], com.agencyport.trandef.behavior.WhereClause[name=XPATH{
substring(PersPolicy/ContractTerm/EffectiveDt, 1, 4) }, value=XPATH{
substring(PersPolicy/ContractTerm/EffectiveDt, 1, 4) }, isSetRequested=false,
isCountRequested=false, xPath=XPathExpression [xPathExpression=
substring(PersPolicy/ContractTerm/EffectiveDt, 1, 4)]]]
```

XPathVariables [{}]

Comparison of where clause: com.agencyport.trandef.behavior.WhereClause[name=StateProvCd, value=CT, isSetRequested=false,
isCountRequested=false, xPath=null] against the preconditions data store value: MA evaluated to false

# Overriding Default Data Cleanup Behavior

Data cleanup runs based on the following default behaviors:

- When a page is visited. Pages that haven't been visited yet do not need their associated data cleaned up.
- Pages upstream from a page being processed does not require data cleanup. For example, if a user is revisiting page 3 and makes a substantial change that may cause other data to be excluded, the data will be related to downstream pages and not upstream ones.
- Added data in roster add operations does not cause other data to become excluded.
- Upload scenarios are handled differently. Downstream pages are included in data cleanup operations.

To override these behaviors, you must extend the class that is responsible for carrying out when the data cleanup operation is engaged. The default behavior outlined above is carried out by the `com.agencyport.preconditions.BasicDataCleanupPageFilter` class. Perform the following to override the default behavior:

1. Extend the `BasicDataCleanupPageFilter` class. This interface was introduced to optimize when data cleanup should run. One of the default assumptions that the class makes is that roster add scenarios do NOT cause a data cleanup operation to be performed. The following is a sample extension. This provides a generic way to alter the base SDK data cleanup decision making process. Check the `CleanupOverrideRule` inner class, specifically the `loadCleanupOverrideRules()` method, for details.

```
/* * Created on Oct 19, 2015 by nbaker AgencyPort Insurance Services, Inc. */ package com.carrier.cleanup; import java.util.ArrayList; import java.util.Arrays; import java.util.Collections; import java.util.List; import java.util.logging.Logger; import com.agencyport.data.DataManager; import com.agencyport.logging.LoggingManager; import com.agencyport.menu.MenuController; import com.agencyport.preconditions.BasicDataCleanupPageFilter; import com.agencyport.tranDef.Transaction; import com.agencyport.websHared.IWebsHaredConstants; /** * The CarrierDataCleanupPageFilter class illustrates how to modify the scenarios which drive when * and under which conditions data cleanup operations are engaged. */ public class CarrierDataCleanupPageFilter extends BasicDataCleanupPageFilter { /* * The <code>LOGGER</code> is our logger. */ private static final Logger LOGGER = LoggingManager.getLogger(CarrierDataCleanupPageFilter.class.getPackage().getName()); /* * The CleanupOverrideRule class models a generic way to override the default data cleanup controller. */ private static final class CleanupOverrideRule { /* * The <code>dataAccessType</code> is the data access type operation that will be overridden. */ private final int dataAccessType; /* * The <code>pageBeingProcessed</code> is the current page in context. The one being processed. */ private final String pageBeingProcessed; /* * The <code>pagesToInclude</code> is the list of downstream pages that should be included for * data cleanup activities that may not be. */ private final List<String> pagesToInclude = new ArrayList<>(); /* * Constructs an instance. */ @param pageBeingProcessed see {@link #pageBeingProcessed} * @param dataAccessType see {@link #dataAccessType} * @param pagesToInclude see {@link #pagesToInclude} */ private CleanupOverrideRule(String pageBeingProcessed, int dataAccessType, String... pagesToInclude){ this.dataAccessType = dataAccessType; this.pageBeingProcessed = pageBeingProcessed; this.pagesToInclude.addAll(Arrays.asList(pagesToInclude)); } /* * Determines whether the current page being processed and the current data access type is a match. If it is then this will schedule the effected pages to be included. */ @param filter is the parent filter instance. * @param transaction is the transaction. * @param menuController is the menu controller. * @param dataManager is the data manager (access to control data if necessary). * @param currentPageId is the current page being processed. * @param dataAccessType is the data access type for the current request. */ private void apply(BasicDataCleanupPageFilter filter, Transaction transaction, MenuController menuController, DataManager dataManager, String currentPageId, Integer dataAccessType){ if (this.pageBeingProcessed.equals(currentPageId) && this.dataAccessType == dataAccessType){ for (String pageToInclude : pagesToInclude){ if (menuController.hasPageBeenVisited(pageToInclude)){ LOGGER.info(String.format("Targeting page with id of '%s' for data cleanup", pageToInclude)); filter.getExcludedPages().remove(pageToInclude); } } } } /* * @Override */ public String toString(){ return "CleanupOverrideRule [dataAccessType=" + dataAccessType + ", pageBeingProcessed=" + pageBeingProcessed + ", pagesToInclude=" + pagesToInclude + "]"; } /* * The <code>CLEANUP_OVERRIDE_RULES</code> is the list of rules that will override the SDK's default * data cleanup controller. */ private static final List<CleanupOverrideRule> CLEANUP_OVERRIDE_RULES = loadCleanupOverrideRules(); /* * Initializes the rule set that will override the default SDK cleanup rules. */ private static List<CleanupOverrideRule> loadCleanupOverrideRules(){ List<CleanupOverrideRule> cleanupOverrideRules = new ArrayList<>(); // Applications can decide on the method to initialize this data structure.// Here we are basically just hard wiring it. cleanupOverrideRules.add(new CleanupOverrideRule("DriversPage", IWebsHaredConstants.DATA_ACCESS_TYPE_INSERT, "VehiclesPage")); return Collections.unmodifiableList(cleanupOverrideRules); } /* * Constructs an instance. */ @param transaction is the transaction. * @param menuController is the menu controller. * @param dataManager is the data manager (access to control data if necessary). * @param currentPageId is the current page being processed. * @param dataAccessType is the data access type for the current request. */ public CarrierDataCleanupPageFilter(Transaction transaction, MenuController menuController, DataManager dataManager, String currentPageId, Integer dataAccessType){ super(transaction, menuController, dataManager, currentPageId, dataAccessType); for (CleanupOverrideRule cleanupOverrideRule : CLEANUP_OVERRIDE_RULES){ cleanupOverrideRule.apply(this, transaction, menuController, dataManager, currentPageId, dataAccessType); } } }
```

2. Register the `BasicDataCleanupPageFilter` class either at the application or LOB level.

## Example

The following is an example of registering `BasicDataCleanupPageFilter` for the personal auto LOB:

```
#Snippet from WEF-INF/autop/properties/autop.properties # Data cleanup page filter
override registration.

AUTOP.datacleanup_page_filter=com.carrier.cleanup.CarrierDataCleanupPageFilter.
```

# Overriding Default Data Cleanup Behavior

Data cleanup runs based on the following default behaviors:

- Pages not yet visited do not need their associated data cleaned up. This is based on the assumption that a user has not visited the page yet, so there is no data to cleanup.
- Pages upstream from a processed page do not require data cleanup. For example, if a user is revisiting page 3 and makes a substantial change that may cause other data to be excluded, the data is related to downstream pages (not upstream pages). An exception to this is an upload scenario, which does include downstream pages in data cleanup operations.
- Data added during rosteradd operations is not cleaned up since it is assumed that the added data does not cause other data to become excluded.

There may be scenarios where you will want to override this behavior. For example, data may need to be cleaned up in certain circumstances where a driver entity is added and causes fields on downstream vehicles to become excluded.

To override these behaviors, you must extend the class that is responsible for carrying out when the data cleanup operation is engaged. The default behavior outlined above is carried out by the `com.agencyport.preconditions.BasicDataCleanupPageFilter` class. Perform the following to override the default behavior:

1. Extend the `BasicDataCleanupPageFilter` class. This interface was introduced to optimize when data cleanup should run. One of the default assumptions that this class makes is that rosteradd scenarios do NOT cause a data cleanup operation to be performed.

## Example

The following is a sample extension. It provides a generic way to alter the base SDK data cleanup decision making process.

Refer to the `CleanupOverrideRule` inner class and the corresponding `loadCleanupOverrideRules()` method for details. The `cleanupOverrideRules.add(new CleanupOverrideRule("DriversPage", IWebsharedConstants.DATA_ACCESS_TYPE_INSERT, "VehiclesPage"))`; line highlighted below specifically forces a cleanup on the Vehicles page (TDF `pageID="VehiclesPage"`) when an update occurs on the Drivers page (TDF `pageID="DriversPage"`). You can customize this code to select the parent page that triggers a cleanup and any children page(s) that also trigger a cleanup.

```
/* * Created on Oct 19, 2015 by nbaker AgencyPort Insurance Services, Inc. */ package com.carrier.cleanup; import java.util.ArrayList; import java.util.Arrays; import java.util.Collections; import java.util.List; import java.util.logging.Logger; import com.agencyport.data.DataManager; import com.agencyport.logging.LoggingManager; import com.agencyport.menu.MenuController; import com.agencyport.preconditions.BasicDataCleanupPageFilter; import com.agencyport.trandef.Transaction; import com.agencyport.webshared.IWebsharedConstants; /** * The CarrierDataCleanupPageFilter class illustrates how to modify the scenarios which drive when * and under which conditions data cleanup operations are engaged. */ public class CarrierDataCleanupPageFilter extends BasicDataCleanupPageFilter { /** * The <code>LOGGER</code> is our logger. */ private static final Logger LOGGER = LoggingManager.getLogger(CarrierDataCleanupPageFilter.class.getPackage().getName()); /** * The CleanupOverrideRule class models a generic way to override the default data cleanup controller. */ private static final class CleanupOverrideRule { /** * The <code>dataAccessType</code> is the data access type operation that will be overridden. */ private final int dataAccessType; /** * The <code>pageBeingProcessed</code> is the current page in context. The one being processed. */ private final String pageBeingProcessed; /** * The <code>pagesToInclude</code> is the list of downstream pages that should be included for * data cleanup activities that may not be. */ private final List<String> pagesToInclude = new ArrayList<>(); /** * Constructs an instance. * @param pageBeingProcessed see {@link #pageBeingProcessed} * @param dataAccessType see {@link #dataAccessType} * @param pagesToInclude see {@link #pagesToInclude} */ private CleanupOverrideRule(String pageBeingProcessed, int dataAccessType, String... pagesToInclude){ this.dataAccessType = dataAccessType; this.pageBeingProcessed = pageBeingProcessed; this.pagesToInclude.addAll(Arrays.asList(pagesToInclude)); } /** * Determines whether the current page being processed and the current data access type is a match. If it is then this will schedule the effected pages to be included. * @param filter is the parent filter instance. * @param transaction is the transaction. * @param menuController is the menu controller. * @param dataManager is the data manager (access to control data if necessary). * @param currentPageId is the current page being processed. * @param dataAccessType is the data access type for the current request. */ private void apply(BasicDataCleanupPageFilter filter, Transaction transaction, MenuController menuController, DataManager dataManager, String currentPageId, Integer dataAccessType){ if(this.pageBeingProcessed.equals(currentPageId)&& this.dataAccessType == dataAccessType){ for(String pageToInclude : pagesToInclude){ if(menuController.hasPageBeenVisited(pageToInclude)){ LOGGER.info(String.format("Targeting page with id of '%s' for data cleanup", pageToInclude)); filter.getExcludedPages().remove(pageToInclude); } } } } /** * The <code>CLEANUP_OVERRIDE_RULES</code> is the list of rules that will override the SDK's default * data cleanup controller. */ private static final List<CleanupOverrideRule> CLEANUP_OVERRIDE_RULES = loadCleanupOverrideRules(); /** * Initializes the rule set that will override the default SDK cleanup rules. */ private static List<CleanupOverrideRule> loadCleanupOverrideRules(){ List<CleanupOverrideRule> cleanupOverrideRules = new ArrayList<>(); // Applications can decide on the method to initialize this data structure.// Here we are basically just hard wiring it. cleanupOverrideRules.add(new
```

```
CleanupOverrideRule("DriversPage", IWebsharedConstants.DATA_ACCESS_TYPE_INSERT, "VehiclesPage")); return
Collections.unmodifiableList(cleanupOverrideRules); } /** * Constructs an instance. * @param transaction is the transaction. *
@param menuController is the menu controller. * @param dataManager is the data manager (access to control data if necessary). *
@param currentPageId is the current page being processed. * @param dataAccessType is the data access type for the current request. */
public CarrierDataCleanupPageFilter(Transaction transaction, MenuController menuController, DataManager dataManager, String
currentPageId, Integer dataAccessType) { super(transaction, menuController, dataManager, currentPageId, dataAccessType); for
(CleanupOverrideRule cleanupOverrideRule : CLEANUP_OVERRIDE_RULES){ cleanupOverrideRule.apply(this, transaction,
menuController, dataManager, currentPageId, dataAccessType); } } }
```

2. Register the BasicDataCleanupPageFilter class either at the application or LOB level.

### Example

The following is an example of registering BasicDataCleanupPageFilter for the personal auto LOB:

```
#Snippet from WEF-INF/autop/properties/autop.properties # Data cleanup page filter
override registration.

AUTOP.datacleanup_page_filter=com.carrier.cleanup.CarrierDataCleanupPageFilter.
```

# Field Validation

Within the AgencyPortal framework, field validations are expressed within the transaction definition file, giving TDF authors the ability to directly express zero to many validation rules for a given field. AgencyPortal field validations have a client-side implementation written in JavaScript, as well as a server-side implementation written in Java.

- The client-side implementation of field validations in AgencyPortal is designed to give the end user instant feed back if he/she has entered invalid data in a field on the page. Page submission is prevented until data on the page is well formed per its expressed validation rules. Client-side field validations are intended to enhance the user's experience within the AgencyPortal application and has been explicitly architected not to inhibit the user's ability to do work within the application.
- The server-side field validations are implemented for the purposes of a final-check of the data from within the controlled server-operating environment to ensure that corrupt or invalid data is not inadvertently committed to the data store.

The intent of the field validation engine is to provide a lightweight, user-friendly mechanism for simple validation rules. The field validation engine, and the validation rules implemented within it, are simple in nature and are designed only to address a finite number of common validation routines.

The scope of the methods implemented within the validation is limited to individual fields. Cross-field validation (i.e., validation rules that take into account more than one field) is not inherently supported by the validation engine. (However, there is no reason a custom validator cannot be written to perform cross-field validation programmatically.)

A common Java interface is published, which supports server side validations, both built-ins, as well as user-defined. The implementations of this interface built into the product as pre-canned validation components adheres to the stateless design to reduce the number of built-in object creations required on a per page per thread basis.

*The field validation infrastructure is not a field formatting one. Field validations enforce formats, but leave the task of actually updating the field to the end user or to format mask decorations configured on those particular fields.*

The order in which server side validations are fired will follow the field element order configured on a given TDF page.

Validations are only fired for fields that are present in the `HTMLDataContainer` unless the field element is one of the following types:

- question
- radio
- selectlist
- filterlist

# Available Validation Methods

Item	Parameter	Description	Technical Definition	Default SDK generated side error message
required	<none>	A value must be entered	String length of 0 or null string reference raises an exception.	\${field_label} is required
numeric	<none>	If entered, a field must be numeric. Examples of data values that will pass this test are: <ul style="list-style-type: none"><li>• 0</li><li>• 10</li><li>• -10</li><li>• +40</li><li>• +40.01</li></ul>	Double.parseDouble(value) does not throw an exception	\${field_label} must be numeric
length	(length)	If entered, the field must be exactly length characters	String length must equal to configured length parameter	\${field_label} must be \${length} characters in length
minLength	(length)	If entered, the field must be no less than length characters	String length must be greater than or equal to configured length parameter	\${field_label} must be at least \${min_length} characters in length
maxLength	(length)	If entered, the field must be no more than length characters	String length must be less than or equal to configured length parameter	\${field_label} can't exceed \${max_length} characters in length
minValue	(value)	(implies that this field is numeric) - If entered, the value submitted must be no less than value	Numeric value must be greater or equal to configured value parameter	\${field_label} must be a number greater than or equal to \${min_value}
maxValue	(value)	(implies that this field is numeric) - If entered, the value submitted must be no more than value	Numeric value must be less than or equal to configured value parameter	\${field_label} must be a number less than or equal to \${max_value}
alphanumeric	<none>	If entered, the value entered must be comprised of only letters and numbers - al-pha-nu-mer-ic (alf?-nu-mer'ik, -nyu-) also al-pha-mer-ic (-f?-mer'ik) adj. Consisting of both letters and numbers.	If the regular expression pattern of '\W' does not find a match on the data field, then no error is raised. This particular interpretation only accepts alphabet letters (upper and lower case), numeric digits and the underscore character (_). No punctuation, spaces or other symbols are accepted.	\${field_label} must be alphanumeric
format	(regularExpression)	If entered, the value entered must match the pattern expressed in regularExpression	If the regular expression does not have at least one match on the input data, then an error condition is raised.	\${field_label} does not conform to the expected format
ACORDDate Format	<none>	If entered, the value must conform to a yyyy-MM-dd format and must be a valid date. Month and day portions must be left zero padded where needed.	If SimpleDateFormat("yyyy-MM-dd") throws a	\${field_label} must be entered in \${date_format} format

Item	Parameter	Description	Technical Definition	Default SDK generated side error message
			ParseException, then an error condition is raised.	
USDateFormat	<none>	If entered, the value must conform to a MM-dd-yyyy format and must be a valid date. Month and day portions must be left zero padded where needed.	If SimpleDateFormat ("yyyy-MM-dd") throws a ParseException, then an error condition is raised.	\${field_label} must be entered in \${date_format} format
phone	<ul style="list-style-type: none"> <li>• &lt;none&gt;</li> <li>• US</li> <li>• GLOB AL</li> </ul>	<p>Where no parameter value is configured and if entered, the value must comply with one of the following formats:</p> <ul style="list-style-type: none"> <li>• US phone format of (nnn) nnn-nnnn</li> <li>• International ACORD phone format, complying with regular expression "\+ \\$+\-\\$+\-\\$+\(\+\\$+\)"</li> </ul> <p>If the US parameter value is configured, then only the US format is accepted.</p> <p>If the GLOBAL parameter is configured, then only the global/international format is accepted.</p>	If either of the regular expression `^(?([2-9]\d{2})\)\?-?\d{3}-? \d{4}((  \.)?[ext\.\.]+\d+)?` or `^[\+\\$+\-\\$+\-\\$+\(\+\\$+\)]?` finds a match, then an error condition is NOT raised	\${field_label} is not a recognized phone format
ssn	<none>	If entered, the value must comply with the following format: 'nnn-nn-nnnn'	If the regular expression of '\d{3}-\d{2}-\d{4}' is not found and the length is not equal to 11, then an error condition is raised	\${field_label} must be entered in the SSN format of nnn-nn-nnnn
fein	<none>	If entered, the value must comply with the following format: 'nn-nnnnnnnn'	If the regular expression of '\d{2}-\d{7}' is not found and the length is not equal to 1,0 then an error condition is raised	\${field_label} must be entered in the FEIN format of nn-nnnnnnn
USZipCode	<none>	If entered, the value must comply with either of the following formats: 'nnnnn' or 'nnnnn-nnnn'	If the regular expression of '\d{5}' or '\d{5}-\d{4}' is not found or the length is not equal to 5 or 10, then an error condition is raised	\${field_label} must be a valid US zip code
email	<none>	Must be a valid email address string	If the regular expression of `^\w[a-zA-Z0-9-]+(\.[a-zA-Z0-9-]+)*@[a-zA-Z0-9-]+\.(a-zA-Z){2,3}\)(aero coop info museum name)\)\$` is not found, then an error condition is raised	\${field_label} must be a valid email address
pattern	regular express - literal syntax	If entered, the value must match the pattern expressed in	If the regular expression does not have at least one match on	\${field_label} does not conform to the expected format

<b>Item</b>	<b>Parameter</b>	<b>Description</b>	<b>Technical Definition</b>	<b>Default SDK generated side error message</b>
		regularExpression literal syntax	the input data, then an error condition is raised.	
custom	(Java class name that implements the com.agency.port.field.validation.validator.s.IValidator interface)	Custom	Signals an error condition by returning a ValidationError instance	Custom
dateCompare	date comparison expression	If entered, the date's value is verified against the date of reference	<p>Supports a before or after date comparison between a user date value and a date of comparison, which can be one of the following:</p> <ul style="list-style-type: none"> <li>• static date value</li> <li>• dynamic date value</li> <li>• relative date value, as in today, tomorrow or yesterday</li> </ul> <p>The general format of a date compare validation element value is [!] before OR after (date_of_comparison [+ OR -] [n units])</p> <p>(where ! negates the comparison, n is a whole number and units can be either m (months), d (days) or y (years - default if missing) The following are some examples:</p> <pre> &lt;validation type="dateCompare"&gt;before( '1960-01-01' ) &lt;/validation&gt; interpreted as: verify date is before Jan-01-1960 &lt;validation type="dateCompare"&gt;after( '1960-01-01' ) &lt;/validation&gt; interpreted as: verify date is after Jan-01-1960 &lt;validation type="dateCompare"&gt;before( #today ) &lt;/validation&gt; interpreted as: verify date is before today's date &lt;validation type="dateCompare"&gt;!before( #today + 3m) &lt;/validation&gt; interpreted as: verify date is not before 3 months from today &lt;validation type="dateCompare"&gt;before( PersPolicy.ContractTerm.EffectiveDt - 15y) &lt;/validation&gt; interpreted as: verify date is 15 years or more before the effective date of the policy </pre>	

Item	Parameter	Description	Technical Definition	Default SDK generated side error message
			<pre> &lt;validation type="dateCompare"&gt;before( #today - 15d) &lt;/validation&gt; interpreted as: verify date is 15 days or more before today's date &lt;validation type="dateCompare"&gt;before( #tomorrow ) &lt;/validation&gt; interpreted as: verify date is before tomorrow &lt;validation type="dateCompare"&gt;after( #yesterday)&lt;/validation&gt; interpreted as: verify date is after yesterday &lt;validation type="dateCompare"&gt;after( PersPolicy.ContractTerm.Effect iveDt + 1y) &lt;/validation&gt; interpreted as: verify date at least 1 year after the effective date of the policy </pre> <p><b>Important</b></p> <p>The corollary client side validation implementation is not supported. Only the server side is rendered.</p>	

# Built-in Field Validation Connector

The server side field validations are implemented as an SDK connector. The difference is that explicit configuration is not required in the application's connector configuration file to engage the field validations.

## Process Side Field Validations

When server side field validations are enabled, the connector that hosts the built-in field validation method to run is "injected" dynamically into the connector list for a given page as the first connector to run. This forgoes the need to explicitly configure the built-in field validation connector to run. The default implementation of this connector is thought to serve the majority of needs by applications; however, you may need to extend the default base class, `com.agencyport.fieldvalidation.connector.BuiltinFieldValConnector`, if you need to change one or more of the following behaviors:

1. The preconditions that contribute toward the decision whether to run the field validation methods for an entire page. The method that can be overridden/extended is Boolean `continueWithExecute(Map dataBundle)`. The base implementation tests for the following "true" conditions:
  - The `fieldValidations="enabled"` is set at the transaction level in the TDF
  - The `fieldValidations="serverSideDisabled"` is not configured on the page under consideration
  - The page is an SDK framework based page
  - If the page is not a roster page
  - If the page is a roster page and the actions are EDIT or COPY

The above logic is contained in the method protected boolean `continueWithExecute(Map dataBundle)`. Derived forms can override this if the above logic should be altered. This logic is also available for use via a static method on the `shouldRunFieldValidation(Map dataBundle)` class public static Boolean, making this logic available outside the scope of this instance.

2. The list of fields that have their field validation methods run. Before the validation methods are run, each field in the TDF is checked to determine its candidacy for running its set of validation methods. The base implementation tests for the following "true" conditions:
  - If a field has a data value on the servlet request (has a field in the `HTMLDataContainer`), then any field validation method configured on that field will run.
  - Some HTML controls like radio button or check box controls (question-answer elements) do not have input data parameters streamed into the server when the user does not answer the question at all. The base implementation makes a further check on the control type so that required validations will still run on these types of controls, regardless of whether there is data value present in the `HTMLDataContainer`.

## Display Side Field Validations / Connectors

Display side field validation methods should be engaged to enhance the upload experience. The feature can be enabled by applying the `prevalidate="true"` attribute at either the transaction or page level of the TDF. The idea behind this notion is to give the user feedback about the validity of the upload data as soon as possible rather than waiting for the user to press 'Continue.' Data entry, roster display and roster edit pages qualify for this experience. Both data entry and roster edit pages run pre-validation on first view of the data.

For data entry pages, the notion of "first view" is tied directly to the page visited flag. For roster edit pages, the repeating aggregate's saved state attribute is used when a value of false triggers pre-validation on entry into the roster edit page.

On roster display pages, the framework iterates through all of the roster items and runs pre-validation for each entry whose saved attribute value is not yet set (neither true nor false). When an error situation is encountered for that particular roster item, the visual rendering for that particular roster item is altered to provide a way for the user to distinguish between valid and invalid roster items. Furthermore, the detailed error messages generated by any connectors run on the display side of a display roster page are in fact suppressed.

In the following example the second driver has something wrong, which requires the user's attention. To force the user to visit all invalid roster items, the `forceViewOnUpload="true"` must be configured for that page in the TDF.

\* Denotes Required Fields

**Please list all rating classifications for all locations**

Rating Information					Help with this List
For Location	Class Code	Full Time Empl	Part Time Empl	Est. Annual Exposure	
162 Main Street Peabody, MA 01960	0005	4	6	70000	<a href="#">Edit</a> <a href="#">Delete</a>
162 Main Street Peabody, MA 01960	9521	2	3	5000	<a href="#">Edit</a> <a href="#">Delete</a>
(2)	0913	3	2	50000	<a href="#">Edit</a>

[Continue](#) [Add New](#) [More Actions...](#) [▼](#)

User-defined connectors can also be engaged on the display side by configuring those connectors with the `supportsPrevalidation="true"` attribute.

### Note

For roster display pages, the connectors are assumed to be `HTMLDataContainer` based and not `transaction(APDataCollection)` based. If a user-defined connector is not `HTMLDataContainer` based, then you should probably wait to run it on the process side of the roster display.

# Custom Field Validation

The ability to create custom field validators allows the application developer to extend the library of built-in validators. Every field validator, built-in or custom, must implement to SDK Java interface, `com.agencyport.fieldvalidation.validators.IValidator`. All of the built-in SDK validators extend an abstract class `com.agencyport.fieldvalidation.validators.BaseValidator`.

Application groups can also choose to extend this class rather than implement the entire interface from scratch. The `BaseValidator` class simplifies the JavaScript generation to some degree and includes an optimized JavaScript method cache. It also separates the validation invocation into an `isValid()` method call followed by an error message rendering call when the `isValid()` returns a false.

The following examples detail two different custom field validators:

- One fully implements the `IValidator` interface, although it does not serve up client side JavaScript.
- The other extends the `BaseValidator` abstract class in a manner consistent to the built-in SDK validators.

Both sets of source code are located in the `src.jar` file that is part of the full `apwebapp.zip` package.

## Custom Field Validation Example #1

The following sample code provides an example of a custom validator that handles both SSN and FEIN formats. Internally, it uses the SSN and FEIN built-in validators based on the section of another field. In some ways, this is a cross-field validation test.

Assume the following TDF is configured:

```
<pageElement> <fieldElement type="selectlist"
id="InsuredOrPrincipal[InsuredOrPrincipalInfo.InsuredOrPrincipalRoleCd='Insured'].GeneralPartyInfo.NameInfo.TaxIdentity.TaxIdTypeCd"
label="Tax Id Type" required="false" help="" defaultValue=""> <optionList reader="xmlreader" source="codeListRef.xml"
target="selectOne"/><optionList reader="xmlreader" source="codeListRef.xml" target="taxIdType"/></fieldElement> <fieldElement
type="text"
id="InsuredOrPrincipal[InsuredOrPrincipalInfo.InsuredOrPrincipalRoleCd='Insured'].GeneralPartyInfo.NameInfo.TaxIdentity.TaxId"
label="Tax Id" required="false" size="14" maxLength="11" defaultValue=""> <validation
type="custom">test.agencyport.validators.FEINAndSSNValidator</validationElement> </field Element> </pageElement>
```

Assume the following is a fragment in the `codeListRef.xml`:

```
<optionList id="taxIdType"> <option enabled="true" value="FEIN">FEIN</option> <option enabled="true" value="SSN">SSN</option>
</optionList>
```

The following is the Java code:

```
/* * Created on Jun 27, 2006 by norm AgencyPort Insurance Services, Inc. */ package test.agencyport.validators; import
com.agencyport.fieldvalidation.ValidationContext; import com.agencyport.fieldvalidation.ValidationError; import
com.agencyport.fieldvalidation.ValidationManager; import com.agencyport.fieldvalidation.javascript.JavaScriptMethodBodyInfo; import
com.agencyport.fieldvalidation.javascript.JavaScriptMethodInvocationAndErrorInfo; import
com.agencyport.fieldvalidation.validators.BaseValidator; import com.agencyport.fieldvalidation.validators.IValidator; import
com.agencyport.html.elements.ValidationElement; import com.agencyport.webshared.HTMLDataContainer; /** * The FEINAndSSNValidator
class looks at the TaxIdType field in the HTML data* container and decides whether to apply the SSN or FEIN builtin * validator against the
current data value. It also makes the Tax Id type * field required if the user entered a tax id. */ public class FEINAndSSNValidator implements
IValidator { private static final String taxIdTypeFieldId =
"InsuredOrPrincipal[InsuredOrPrincipalInfo.InsuredOrPrincipalRoleCd='Insured'].GeneralPartyInfo.NameInfo.TaxIdentity.TaxIdTyp eCd"; /* *
(non-Javadoc) * @see com.agencyport.fieldvalidation.validators.IValidator#isStateless() * If a class returns true then the validation framework
will cache this instance* and reuse freely and concurrently across threads from that point onward. */ public boolean isStateless(){ return true; //*
We are promising that there is no state managed by this class } /* * (non-Javadoc) * @see
com.agencyport.fieldvalidation.validators.IValidator#validate(com.agencyport.fieldvalidation.validators.ValidationContext) * This is an example
of a simple custom validation method since the real validation is done by one of the AgencyPort's * own builtin validators. */ public
ValidationError validate(ValidationContext validationContext){ HTMLDataContainer dataContainer =
validationContext.getHTMLDataContainer(); String taxIdType = dataContainer.getStringValue(taxIdTypeFieldId); String taxId =
validationContext.getValue(); // If the user told us a tax id without telling us the tax id type then the // tax id type now becomes required
(dynamically) since we can only // perform the format test once the user has told us the tax id type. if (!BaseValidator.isEmpty(taxId) &&
BaseValidator.isEmpty(taxIdType)){ return new ValidationError("Tax Id Type is required when Tax Id is entered",this, taxIdType,
taxIdTypeFieldId, "Tax Id Type"); } // Determine which builtin validator to dispatch out to. IValidator formatTaxIdValidator = null; if (
taxIdType.equals("SSN")) formatTaxIdValidator =
ValidationManager.acquireBuiltinValidator(IValidator.BUILTIN_SS N_VALIDATOR_TYPE); else formatTaxIdValidator =
ValidationManager.acquireBuiltinValidator(IValidator.BUILTIN_FEIN_VALIDATOR_TYPE); return
formatTaxIdValidator.validate(validationContext); } /* * (non-Javadoc) * @see
```

```

com.agencyport.fieldvalidation.validators.IValidator#generateJavaScriptHook(com.agencyport.html.elements.ValidationElement) * Defer to
server - NO CLIENT SIDE validation logic provided on this custom class */ public String generateJavaScriptHook(ValidationElement
validationElement){ return null; // let server side do this } /* (non-Javadoc) * @see
com.agencyport.fieldvalidation.validators.IValidator#generateJavaScriptInfo(com.agencyport.fieldvalidation.ValidationContext) * Defer to
server - NO CLIENT SIDE validation logic provided on this custom class */ public JavaScriptMethodInvocationAndErrorInfo
generateJavaScriptInfo(ValidationContext validationContext) { return null; } /* (non-Javadoc) * @see
com.agencyport.fieldvalidation.validators.IValidator#getJavaScriptValidationMethod() * Defer to server - NO CLIENT SIDE validation logic
provided on this custom class */ public JavaScriptMethodBodyInfo getJavaScriptMethodInfo() { // TODO Auto-generated method stub return
null; } }

```

## Custom Field Validation Example #2

The following sample code provides an example of a custom validator that handles MASS driver's license number formats.

Assume the following TDF is configured:

```

<fieldElement type="text" id="PersAutoLineBusiness.PersDriver.DriverInfo.License[LicenseTypeCd='Driver' &&
StateProvCd='MA'].LicensePermitNumber" label="License Number" required="false" size="30" maxLength="30" defaultValue=""
groupId="PersAutoLineBusiness.PersDriver.DriverInfo.License[LicenseTypeCd='Driver' && StateProvCd='MA']"> <validation
type="custom">test.agencyport.validators.MassDriversLicenseNumberValidator</validation> </fieldElement>

```

The following is what the Java code would look like:

```

/* * Created on Jul 12, 2006 by norm AgencyPort Insurance Services, Inc. */ package test.agencyport.validators; import java.util.Properties;
import com.agencyport.fieldvalidation.ValidationContext; import com.agencyport.fieldvalidation.javascript.JavaScriptMethodBodyInfo; import
com.agencyport.fieldvalidation.javascript.JavaScriptWriter; import com.agencyport.fieldvalidation.validators.IValidator; import
com.agencyport.fieldvalidation.validators.builtin.RegExValidator; /** * The MassDriversLicenseNumberValidator class validates a MASS
driver's license * number. */ public class MassDriversLicenseNumberValidator extends RegExValidator { /** * The
<code>defaultErrorMessageTemplate</code> is the error message that will be * used if the validation element on this TDF field element does
not have a message * attribute to override this. */ private static final String defaultErrorMessageTemplate = "${field_label} must be a recognized
Massachusetts's driver license format"; /** * The <code>massDriversLicenseRegExpressionToMatch</code> supports both the old * SSN (no
dash)format of 'nnnnnnnnn' and the newer systemgenerated numbers that * begin with the letter 'S'. */ private static final String
massDriversLicenseRegExpressionToMatch = "S\\d\\d{8}"; /** * The <code>requiredLength</code> is the length test that will be applied
regardless * of whether it is the old style or new style license formats. */ private static final int requiredLength = 9; /* (non-Javadoc) * @see
com.agencyport.fieldvalidation.validators.BaseValidator#getMessageTemplate(com.agencyport.fieldvalidation.validators.Validation
Context) */ protected String getMessageTemplate(ValidationContext validationContext){ return defaultMessageTemplate; } /*
(non-Javadoc) * @see
com.agencyport.fieldvalidation.validators.BaseValidator#getMessageTemplate(com.agencyport.fieldvalidation.validators.Validation
Context) */ protected Properties getMessageTemplate(ValidationContext validationContext){ return
createStartingMapWithFieldLabelVariable(validationContext); } /* (non-Javadoc) * @see
com.agencyport.fieldvalidation.validators.BaseValidator#isValid(com.agencyport.fieldvalidation.validators.Validation Context) * This generates
an error if the value does not pass as a valid MASS drivers * license format. */ protected boolean isValid(ValidationContext validationContext) {
// Get the data value to test String value = validationContext.getValue(); if (!isEmpty(value)) { // Do a length test and a regular expression test
return value.length() == requiredLength && findPattern(massDriversLicenseRegExpressionToMatch, value); } else // driver license number
must be empty return true; } /* (non-Javadoc) * @see com.agencyport.fieldvalidation.validators.BaseValidator#getType() */ protected String
getType() { return IValidator.CUSTOM_VALIDATOR_TYPE; } /* (non-Javadoc) * @see
com.agencyport.fieldvalidation.validators.BaseValidator#appendJavaScriptValidationBody(com.agencyport.fieldvalidation.javascript.JavaScript
MethodBodyInfo) * This method is responsible for providing the client Java script for this validation * test. The base class (BaseValidator)
creates the function header and ending curly brace. */ protected void appendJavaScriptValidationBody(JavaScriptMethodBodyInfo jsmi) {
JavaScriptWriter jsw = jsmi.getMethodBodyWriter(); jsw.beginLine(1); jsw.addContent(javascriptIsEmptyTest); jsw.endLine();
jsw.beginLine(1); jsw.addContent("return (value.length == "); jsw.addContent(requiredLength); jsw.addContent(") && (value.search(/"));
jsw.addContent(massDriversLicenseRegExpressionToMatch); jsw.addContent("/)> -1);"); jsw.endLine(); } /** * The Java script method name
must be unique. */ @see com.agencyport.fieldvalidation.validators.BaseValidator#getName() */ protected String
getJavaScriptMethodName() { return "verifyMassDriversLicenseNumber"; } }

```

# Transaction Validation

Transaction validation provides a way to run SDK based field validations and other compliant connectors against an entire work item. The main output artifact from this feature is the [Transaction Validation Report \(TVR\)](#). The following three scenarios warrant the execution of a TVR:

- at the point when an uploaded work item is introduced to the application
- on quote or policy summary pages to ensure the data integrity of the work item
- when a user has answered a question that now requires data collection from other pages, which had previously been excluded (the corollary of DTR data cleanup)

The SDK itself runs a TVR when an uploaded work item is introduced into the application to determine the data integrity of the incoming stream. An application can also run a TVR itself from custom code. The following is an example of a quote summary page that runs a TVR explicitly:

```
/** Runs a Transaction Validation Report for this work item. The results of this will dictate whether or not rating should be called. * This also illustrates on how to get all of the correction reports which have been applied to this work item. * @param conn is a database connection. */
protected void executeCustomReadDataAccess(Connection conn) throws APException {
 WorkItem workItem = new WorkItem();
 WorkItemManager workItemManager = new WorkItemManager(conn);
 workItemManager.getWorkItem(workItem, controlData);
 APDataCollection apData = (APDataCollection) dataBundle.get(IWebsharedConstants.APDATA_COLLECTION);
 PolicySummary summary = makePolicySummary(workItem, apData);
 request.setAttribute("APP_POLICY_SUMMARY", summary);
 // Run a TVR using the
 APCCommand.runTransactionVerification()
 wrapper method TransactionValidationReport tvr = runTransactionVerification(true, true);
 if (tvr.isClean()) {
 // The TVR ran clean meaning that the work item is in a good state
 PremiumCalculator premCalc = new PremiumCalculator(apData);
 premCalc.calculatePremium();
 System.out.println(premCalc.toString());
 WCRatingResults ratingResults = premCalc.getRatingResults();
 request.setAttribute("RATING_RESULTS", ratingResults);
 } else {
 // The TVR did not run clean so we don't run rating but // dump the errors on to the page
 String[] errors = tvr.getAllValidationTextMessages();
 if (errors.length > 0) {
 request.setAttribute("ERRORS", errors);
 }
 }
}
```

In the above example, the application is using an APCCommand wrapper method `runTransactionVerification()`. Besides the creation of a TVR, this method can adjust the menu display for pages with issues and alter the saved attribute on roster entries to false to make certain roster entries with issues stand out when that roster page is rendered. This is evident by the two Boolean arguments that are supplied to this wrapper method. Refer to [JavaDoc](#) for more details.

## Custom Transaction Validation

Both SDK based field validations and XARC rule based connectors (as in `com.agencyport.arcrule.connector.ArcConnector` in `arcweb.jar` and marked as `<executeWhen modifiers="ExecuteForPreValidation" />` in the connector configuration file) are engaged when transaction validation runs. A custom connector can also be adapted so that it will be fired during transaction validation. The following things need to happen in order to make this to occur:

- its corresponding connector configuration entry needs to have the `modifiers` attribute marked with the `ExecuteForPreValidation` value
- the custom connector needs to implement the `com.agencyport.trandef.validation.IPageDataValidator` interface

The following is a sample of the configuration and source code:

### XML

```
<transaction id="quickQuoteWorkersComp"> <definitions> <namedConnectors>
<connector name="ArcRuleUnderwritingManager" class="com.agencyport.arcrule.connector.ArcConnector" />
<connector name="MessageCollectionManager" class="com.agencyport.examples.connector.MessageCollectionManager" />
<connector name="MyCustomConnector" class="test.agencyport.connector.MyValidator" />
</namedConnectors> <customActions/> <customModifiers/> </definitions>
<page id="WCGeneralInfo">
<connector name="ArcRuleUnderwritingManager" > <executeWhen actions="ContinueSave" />
<arcRules ruleSetsFile="workersComp" ruleSet="WCGeneralInfo_underwriting" writeStatistics="true" />
</connector>
<connector name="MyCustomConnector" > <executeWhen actions="All" modifiers="ExecuteForPreValidation" />
</connector>
<connector name="MessageCollectionManager" > <executeWhen actions="All" />
</connector>
</page>
</transaction>
```

### Java

```
/* * Created on Jan 10, 2008 by nbaker AgencyPort Insurance Services, Inc. */
package test.agencyport.connector;
import java.util.Iterator;
import java.util.Map;
import com.agencyport.connector.ConnectorManager;
import com.agencyport.connector.IConnectorConstants;
import com.agencyport.data.DataManager;
import com.agencyport.data.containers.AggregateDataContainer;
import com.agencyport.data.containers.PageDataContainer;
import com.agencyport.domXML.APDataCollection;
import com.agencyport.html.elements.BaseElement;
import com.agencyport.shared.APException;
import com.agencyport.trandef.provider.CompoundKey;
import com.agencyport.trandef.validation.IPageDataValidator;
import com.agencyport.trandef.validation.TransactionValidationReport;
import com.agencyport.trandef.validation.ValidationResult;
import com.agencyport.trandef.validation.ValidationResults;
import com.agencyport.webshared.HTMLDataContainer;
import
```

```

com.agencyport.webshared.IWebsharedConstants;/** * The MyValidator class illustrates how a custom connector can be adapted to * be
engaged by transaction based validation. */ public class MyValidator extends ConnectorManager implements IPageDataValidator { /** * This
implementation is left out. If you want to see how both execute * and verify methods can leverage the same core logic then look at *
com.agencyport.fieldvalidation.connector.BuiltinFieldValConnectorManager as an example. * Note that this is this method (execute()) is called
by prevalidation on the actual display of web page. * @see com.agencyport.fieldvalidation.connector.BuiltinFieldValConnectorManager * @see
com.agencyport.connector.ConnectorManager#execute(java.util.Map) */ public void execute(Map dataBundle) throws APEException {
HTMLDataContainer htmlDataContainer = (HTMLDataContainer)dataBundle.get(IWebsharedConstants.HTML_DATA_CONTAINER); Map
fields = htmlDataContainer.getCopyOfFields(); Iterator fieldsIterator = fields.entrySet().iterator(); while (fieldsIterator.hasNext()){
Map.Entry entry = (Map.Entry) fieldsIterator.next(); String fieldAcordPath = (String) entry.getKey(); String fieldValue =
htmlDataContainer.getStringValue(fieldAcordPath); // Get the base element so that we can get its compound key and field label BaseElement
fieldElement = htmlDataContainer.getAssociatedField(fieldAcordPath); // Validate the field value String errorTextMessage =
validateField(fieldElement, fieldAcordPath, fieldValue); if (errorTextMessage != null){
this.messageMap.addMessage(IConnectorConstants.MESSAGE_ERROR_LITERAL, getClass().getName(), getClass().getName(),
errorTextMessage); } } /*(non-Javadoc) * @see com.agencyport.connector.ConnectorManager#postProcess(java.util.Map) */ public boolean
postProcess(Map dataBundle) { return true; } /*(non-Javadoc) * @see
com.agencyport.trandef.validation.IPageDataValidator#verify(com.agencyport.data.containers.PageDataContainer,
com.agencyport.domXML.APDataCollection, com.agencyport.data.DataManager,
com.agencyport.trandef.validation.TransactionValidationReport) */ public void verify(PageDataContainer pageDataContainer,
APIDataCollection apData, DataManager dataManager, TransactionValidationReport transactionValidationReport) { AggregateDataContainer[]
aggregateDataContainers = pageDataContainer.getAggregateDataContainers(); for (int ij = 0; ij < aggregateDataContainers.length; ij++){ //
There is one aggregate data container for a data entry page and 'n' // of them for each roster entry on roster pages. AggregateDataContainer adc =
aggregateDataContainers[ij]; // Get this aggregate's HTML data container HTMLDataContainer htmlDC = adc.getHTMLDataContainer(); //
Create a container for this aggregate's validation result instances. ValidationResults vRS = new ValidationResults(this, adc); // Validate the data
on the data container - it is more probably that // custom validators will take specific fields from the data container rather // than iterating through
the whole data container here below. Map fields = htmlDC.getCopyOfFields(); Iterator fieldsIterator = fields.entrySet().iterator(); while
(fieldsIterator.hasNext()){ Map.Entry entry = (Map.Entry) fieldsIterator.next(); String fieldAcordPath = (String) entry.getKey(); String
fieldValue = htmlDC.getStringValue(fieldAcordPath); // Get the base element so that we can get its compound key and field label BaseElement
fieldElement = htmlDC.getAssociatedField(fieldAcordPath); // Validate the field value String errorTextMessage = validateField(fieldElement,
fieldAcordPath, fieldValue); if (errorTextMessage != null){ CompoundKey compoundKeyOfFieldInError =
fieldElement.getFieldElementKey(); // We only support this validation category and leave the other one up to // the SDK built-in required field
validation methods. int validationCategory = ValidationResult.VALIDATION_CATEGORY_ENTITY_PRESENT_BUT_INCORRECT;
vRS.addResult(compoundKeyOfFieldInError, validationCategory, errorTextMessage, getClass().getName()); } } if (vRS.hasErrors()){
transactionValidationReport.addResults(vRS); } } /** * Validates a field. This implementation is non-sensical. * @param fieldElement is the
TDF field element. * @param fieldAcordPath is the field element id (versus the uniqueId). * @param fieldValue is the field value. * @return a
null if OK else the error message. */ private String validateField(BaseElement fieldElement, String fieldAcordPath, String fieldValue){ return
null; } }

```

# Transaction Validation Report (TVR)

The Transaction Validation Report (TVR) provides a way to run SDK based field validations and other compliant connectors (XARC Connector, for example) against the entire work item.

- Think of TVR as “ensuring” data integrity, reporting on and presenting messages for “incorrect” data
- TVR runs the same field validation and other compliant connectors (i.e., XARC Connector) that the user fires when they press continue on each page
- TVR does not perform data correction (for example, bumping and slotting)
- TVR does not perform data clean up (for example, deleting data)
- The application tracks (persists) how the user advances or jumps around through the work flow of the transaction
- The application tracks (persists) when TVR is run - will reduce the need to rerun TVR when data changes have not been applied to the work item

Making the assumption that users visit pages in sequential order (in order of the menu – top to bottom), data for a given page is validated on “Continue” for a data entry page and upon “Add” of a roster item on a roster page. If the user continues through the application without going back and making changes to a previous page, all data should be validated up until this point. Since TVR now tracks the pages that have been last updated, TVR determines what pages have been validated up until that point. Only the page on which TVR is invoked would require validation. TVR no longer needs to go back through every page within the application.

## Sample Transaction

The following is a sample transaction with TVR configured for notable milestones, such as Premium Summary and Policy Issuance, along with several scenarios to illustrate when TVR is run.



### Scenario 1 - Quote pages - Straight-thru processing

- User navigates sequentially through the transaction pages – 1,2,3,4,5,6,7
- TVR page 7 => TVR does not run
- User saves and exits quote

### Scenario 2 - Quote & Issue pages - Straight-thru processing

- User navigates sequentially through the transaction pages – 1,2,3,4,5,6,7,8,9,10,11
- TVR page 7 => TVR does not run
- TVR page 11 => TVR does not run

### Scenario 3 - Re-quote

- User opens saved quote and changes data on page 2, continues off of page 2 and then accesses 3,4,5,6,7 sequentially
- TVR page 7 => TVR does not run

### Scenario 4 - Re-quote

- User opens saved quote and changes data on page 2, continues off of page 2, skips page 3 and then accesses 4,5,6,7 sequentially
- TVR page 7 => TVR runs for pages 3,4,5,6

#### Scenario 5 - Re-quote

- User opens saved quote, changes data on page 4, changes data on page 2, does not hit continue on page 2, goes to 5, 6
- TVR page 7 => TVR runs for pages 2,3,4,5,6

#### Scenario 6 - Re-quote

- User enters data on pages 1,2,3,4, goes back to page 2 and changes data and clicks continue, goes back to 4,5,6,7 sequentially
- TVR page 7 => TVR runs for pages 2,3,4,5,6

#### Scenario 7 - Quote pages - Straight-thru processing and revisits pages 2 and 4 without change (i.e., no impactful change)

- User navigates sequentially through the transaction pages – 1,2,3,4,5,6,7
- TVR page 7 => TVR does not run
- User goes back to view page 2, user goes to view page 4 and then goes back to Premium Summary
- TVR page 7 => TVR does not run

#### Scenario 8 - Quote pages - Straight-thru processing and revisits a page with a page impactful change

- User navigates sequentially through the transaction pages – 1,2,3,4,5,6,7
- TVR page 7 => TVR does not run
- User goes back to page 2, changes data, clicks continue and then goes back to Premium Summary
- TVR page 7 = 3,4,5,6

### Additional Notes

There are some cases when a user may submit a request and then go back and modify data on a certain page. If the user changes some information on a page in the middle of the application and jumps to the last available page before the quote page, the pages that are between this last updated page and the quote require validation since the data integrity may have been impacted. TVR will validate these pages. Any pages upstream of the last updated page are still assumed valid and do not require re-validation.

Another change to TVR is the persistence of the results to the database. This is to ensure that TVR is not run again if there were no changes to the application since TVR was last executed. The TVR results have a timestamp that is compared to the timestamp of the current ACORD document. If the ACORD document was not updated since the last TVR timestamp, TVR is pulled from the database. The persistence of TVR is optional.

### TVR Tracking and Impactful Fields

The basic goal of TVR tracking is to minimize the execution of redundant validations when certain workflow patterns are followed. The notion of impactful fields provides the ability to further control when validations should be engaged. This pertains to when the user revisits a previous page in the transaction and makes one or more substantive or impactful changes to the work item and then skips to a page downstream in the workflow they already visited.

#### Example

If a user is on the quote summary page and jumps back and fixes a typo to the application's name on page one and then jumps right back to the quote summary page, you probably wouldn't want TVR to run again. However, the default TVR tracking algorithm would indeed run it again because it can't distinguish between a field that has an impact on referential integrity (hence the notion of an impactful field) and one that doesn't. You would agree that the application name in and of itself should not affect referential integrity of the work item. If, however, the user is on the quote summary page and jumps back and changes the effective date, then you would most definitely want TVR to rerun.

The following are some important guidelines around the configuration of impactful pages and fields.

- You can control "impactfulness" at page or field granularity with the application properties: <LOB>.pages\_impacting\_TVR and <LOB>.fields\_impacting\_TVR respectfully.

#### Important

Out of the box, the SDK assumes that ALL pages and fields are impactful. Once you start configuring impactful pages or fields, YOU MUST ACCOUNT FOR ALL OF THEM. If <LOB>.pages\_impacting\_TVR property is missing, then all pages in the transaction are considered impactful. If <LOB>.fields\_impacting\_TVR property is missing, then all fields in the transaction are considered impactful. If <LOB>.pages\_impacting\_TVR property is configured, then ONLY those pages in the transaction are considered impactful. If <LOB>.fields\_impacting\_TVR property is configured, then ONLY those fields in the transaction are considered impactful.

- A decision to utilize this optimization will introduce some brittleness into the application since any new fields subsequently added to the transaction may need to be accounted for in the aforementioned properties.
- <LOB>.pages\_impacting\_TVR refers to a semi-colon delimited list of page ids. The order is not important.
- <LOB>.pages\_impacting\_TVR refers to a semi-colon delimited list of AP XPaths. Please note that for fields that have their own unique ids, their AP XPath values must be used. Again order is not important.
- Additional refinement of what constitutes a substantive change can be done by extending the SDK's `com.agencyport.trandef.validation.BasicPageTracker.substantiveDifference()` method. Your extension's class name is registered with the framework via the `<LOB>.transaction_validation_tracker_class_name` application property.
- Programmatic access to the TVR tracking record instance for a work item is available from the ControlData structure and accessible by calling the `ControlData.getTransactionValidationTrackingRecord()` method. This tracking record will provide that last page in the transaction a substantive change was made and the last time TVR was ran, as well as a few other pieces of information.

# Connectors

Connectors give you the ability to do some work as the user transitions between the pages of a transaction. The work that you do is entirely up to you (within reason), but usually falls into one of the following three categories:

- Business rules
- 1st/3rd party service callouts (like rating)
- Workflow management (connectors and [instructions](#) are used in conjunction with each other to manage the workflow)

Connectors are added to pages within the TDF/page library. You can have connectors to execute on the display side or process side of a page, such as the following:

## Example

This example illustrates how to add process side connectors to a page in the TDF:

```
<page id="coverage" title="Policy Information" type="dataEntry"><!-- This page's pageElements are omitted for brevity... --> <connectors type="process"><connector type="XARC" id="controllingStateConnector" title="controllingStateConnector"><executeWhen userAction="ContinueSaveAndExit" id="policyAction" /><xarcRules ruleLibraryId="wcValidation" ruleId="controllingStateSupported" id="controlStateRule" /></connector> <connector type="custom" className="com.agencyport.example.DetermineLookupService"></connector> <connector type="custom" className="com.agencyport.example.CallAddressService"></connector> </connectors></page>
```

The most common types of connectors you'll encounter are XARC connectors (XARC is AgencyPortal's built-in rules engine. Refer to the [XARC Rules](#) section for more information) and custom connectors. A custom connector is just a pointer to any Java class that extends the abstract class `com.agencyport.connector.impl.ConnectorManager` [try not to get this confused with the other similarly named classes within the SDK (`IConnector` and `Connector`)]

In a `ConnectorManager` implementation, the method that does the work is the `execute` method. The SDK provides that function with a `ConnectorDataBundle` object, which contains references to a lot of useful information that can be inspected during execution (refer to [How Connectors are Executed](#) for more information).

## Note

There are two other connectortypes (`BUILTIN_FIELD_VALS` and `STANDARD_MESSAGE_COLLATOR`) that you can typically ignore. `BUILTIN_FIELD_VALS` is implicitly added to the process side of your page by default and simply performs all of the serverside validations. `STANDARD_MESSAGE_COLLATOR` is implicitly added to both the display and process side of your page and is responsible for aggregating the messages returned by each prior connector into one `MessageMap` object in preparation for display.

## How Connectors are Executed

Connectors are executed during page transitions (on the display or process side of a page). The unit of work that the framework carries out to process a page submission from the client, or a request to display a page, is loosely referred to as request processing.

Consider that each base processing class (e.g., `servlets.base.CMDBaseProcessDataEntry`) contains the following code:

```
initDataBundle(); ... IConnectorProcessor connectorProcessor = ConnectorProcessorFactory.create(page, dataBundle, false, false, true); boolean allStepsSucceeded = connectorProcessor.run(); if (!allStepsSucceeded) { redisplayPageWithMessages(); return; }
```

## Note

Any user-written class extending the base classes will also inherit this "hook" to the connectors. For more details about the sequence of events between pages, and how to customize those events, check out the [Request Processing](#) section.

This code executes after any data from the page has been pre-loaded into the `APDataCollection` (but not yet persisted to disk).

The first major item of interest in this code is `initDataBundle()`. This method is in `APCommand`. This step rounds up all the data that could possibly be of interest to the connector and dumps it into the aforementioned `ConnectorDataBundle` object.

The data bundle is handed to the `ConnectorProcessor` (in the constructor) who will then give it to the connector when `connectorProcess.run()` is invoked.

## Note

If you customize the request processing flow, you have the ability to use the overridable method `executeCustomPrepareDataBundle()`, should the connector require any other information from the command processor that is not already in the data bundle.

The `ConnectorDataBundle` object has an underlying `java.util.Map`, which contains the following items:

KEY	CONTENTS
APDATACOLLECTION	<code>APDataCollection</code> - AgencyPortal's API over the current data
APSESSION	<code>APSession</code> - wrapper around container's <code>HttpSession</code>
CONTORL_DATA	<code>ControlData</code> - itself a bucket - see below
HTTP_SERVLET_REQUEST	<code>HttpServletRequest</code> - the container's request object
PAGE	Page - the page, containing all the field elements
APCOMMAND	The <code>APCommand</code> instance that handled this request also contains a reference to the <code>DataManager</code> .

You do not have to actually use the key values specified above since the `ConnectorDataBundle` object provides getters for all of these built-in key values (including the `DataManager` reference that is retrieved via the `APCommand` object).

The contents of `CONTROL_DATA` are:

KEY	CONTENTS
WORKITEMID	Integer - the key for this insurance work item
TRANSACTION_NAME	String - the name of the transaction, from the transaction file
PAGE_NAME	String - the name of the current page
USERID	Integer - the current user's Id
USERGROUPID	Integer - the group the user belongs to

The `connectorProcessor.run()` method iterates over the connectors assigned to a page and executes them in the order in which they are specified.

A connector is created by having your class implement the abstract class `ConnectorManager`. There are only two methods that must be implemented and the second may be empty:

```
execute(ConnectorDataBundle dataBundle) postProcess(ConnectorDataBundle dataBundle)
```

The `execute` method is where the `ConnectorManager` does whatever it needs to do (this means whether applying some business logic, invoking a service call, etc). Connectors are required to return an outcome from their `execute()` method. Outcomes are tracked for the purpose of [workflow](#).

The `postProcess` method allows you to do some additional work after the connector's `execute` method has returned. Importantly, if this method returns false, then future connector processing for the page will be skipped.

The base class contains a very important member, which is accessible to the concrete implementer:

```
protected MessageMap messageMap;
```

The `messageMap` is "automatically tied" to the transaction's message area on the UI, which is built into the framework; that is, messages added to the message map will be shown to the end user in the browser.

In simple form, a message is added to the map like this:

```
messageMap.addMessage(IConnectorConstants.MESSAGE_INFO_LITERAL, "The requested liability limit exceeds 500,000. " + "This application cannot be bound on-line. An underwriter will contact you shortly.");
```

#### Note

If the `interpretPresenceOfErrorMessageAsErrorCondition` method returns true (which is the default base implementation), then adding an error message (IConnectorConstants. MESSAGE\_ERROR\_LITERAL) to the MessageMap triggers an error condition. When an error condition occurs, the page re-displays with error messages and no data persistence takes place.

If the connector is one that performs field validation - that is, one that enforces simple data entry rules, such as "zip code must be 5 (or 9) numeric digits" - there is another method available in MessageMap that looks like this:

```
addFieldRelatedMessage(String type, String fieldId, String resultMessage)
```

`fieldId` is the HTML field name of the offending field. The framework will automatically highlight this field when re-displaying the page for correction.

#### Note

In most cases, it is appropriate to use the TDF/Page Library field [validators](#) to validate fields. By default, those validations will be automatically evaluated on the server side by the implicit BUILTIN\_FIELD\_VALS connector.

## Connectors In Detail

### Built-in Field Validation

AgencyPortal's built-in field validation connector supports both 3.x and 4.x paradigms. To allow applications to control if and when SDK field validations run, regardless of whether it pertains to process side or display side connectors, a parameter named `enableValidation` on the connector will govern this. The value will be driven by an enumeration with the following interpretation:

- first - built-in field validations will be run first in the set of sibling connector instances, requiring no additional configuration. This will be the default value for process side connectors. In the following example, the built in field validation connector would be before any of the explicitly configured ones.
- last - built-in field validations will be run last in the set of sibling connector instances, requiring no additional configuration. In the following example, the built in field validation connector would be after all of the explicitly configured ones.
- manual - built-in field validations will be where explicitly configured. This is the default value for display side connectors. Conversely, if the connector is marked as manual and the built-in field validation is not configured, then this is the way you can suppress the execution of the built-in field validation altogether. In the following example, the built-in field validation connector would run second after myOwnConnector.

Running field validations as the first connector:

```
<connectors type="display" enableValidation="first" ><connector id="Connector1" className="myOwnConnector" type="custom" />
<connector id="Connector3" className="myOtherConnector" type="custom" /></connectors>
```

Running field validations as the last connector:

```
<connectors type="display" enableValidation="last" ><connector id="Connector1" className="myOwnConnector" type="custom" />
<connector id="Connector3" className="myOtherConnector" type="custom" /></connectors>
```

### Pre-Validation and TVR

Pre-validation will continue to leverage process side connectors and not display side connectors. Moreover, TVR (refer to the [TVR](#) section for more information) will continue to leverage process side connectors marked as supporting pre-validation. The product's own built-in field validation connector is automatically marked as such. The XARC connector must be explicitly marked as follows to be swept up in a TVR:

```
<connectors type="process" ><connector id="generalInfo_Page" type="XARC" supportsPrevalidation="true" ><executeWhen
userAction="ContinueSave" /> <xarcRules id="X1" ruleLibraryId="generalInfo"/> </connector></connectors>
```

Any custom connectors that are desired to participate in TVR must be marked with the `supportsPrevalidation` flag set to true as well. The underlying custom **Connector Manager implementation must also implement the IPagedataValidator interface for it to participate in TVR**.

If a page is configured to run a TVR, this will run before the display side connectors or pre-validation. The following order prevails on the display side:

- Run TVR against the transaction first
- Run display side connectors for the page next
- Run pre-validation (process side connectors) for the page last

#### Note

Display side connectors on roster pages are not run for each entry in the roster list. They are run once for the page, regardless of the number of roster entries present; whereas, pre-validation is applied to each roster entry.

# XARC Rules

XARC rules is AgencyPortal's built-in rules engine that is used to execute complex data validations and underwriting rules. These rules provide feedback to the user and can alter a work item status for workflow, such as an underwriter referral.

You can create and maintain XARC rules. You must attach rules to pages via [connectors](#) and assign the output messages to one of the following severity types:

- Validation - This type of rule alerts the user, after submitting a page, that they performed an invalid action. The user is not allowed to continue.
- Warning - This type of rule provides an alert to the user, after submitting a page, that data entered in the application requires review; however, the user is allowed to continue.
- Information - This type of rule provides some informational text, after submitting a page, based on an action performed by the user. The user is allowed to continue.

XARC rule messages can contain dynamic values using variables, and are designed around positive phrasing.

## Example

When the results of the rule is true, the following message displays: *If the limit is greater than 100,000, then refer it to underwriting.*

Rule messages can include instructions, such as update the status of a work item or assign a work item to a specific user or role. XARC rules can also:

- hook to SQL tables to fetch data
- depend on other rules for execution

Click [here](#) to view our online XARC Essentials course and learn more about XARC rules.

# Technical Notes for XARC Rules

The following are the steps involved in the execution of XARC rules:

1. Read the rule from the rule file.
2. Determine whether the rule should be executed based on the `dependsOn` condition defined in the rule.
3. Evaluate the variables. If the variable is evaluating data on repeating aggregates, the variables are evaluated for each repeating aggregate.
4. Evaluate each of the conditions for all the aggregates. Remove any aggregates from further evaluation if they don't meet the condition.
5. If there is data that satisfies all the conditions, then process the message text by replacing the substitutable variables.
6. Construct the rule result object and return back to the caller.

## Variables

Variables are used to hold on to data that can be later used in conditions or messages. During the execution of rules, variables are first processed before any conditions are evaluated. Make sure that functions or arithmetic operators are used only when you are sure the field has data.

## Example

You define a variable as the following expression:

```
dateOfBirthPlus10years = PersAutoLineBusiness.PersDriver.DriverInfo.PersonInfo.BirthDt + 10 years
```

This would cause an error if the date of birth field does not have a value in the XML.

## Built-in Variables

Today's Date: Server date in the format YYYY-MM-DD

INDEX: Position of the aggregate under consideration in the ACORD XML. This index is 1 based as XPath uses 1 based index as opposed to APDatacollection, which is 0 based.

So the index of the first aggregate is 1 and not 0.

## Negate Operators

`NOT=`, `IS NOT IN` and `DOES NOT EXIST` are the supported negate operators.

XARC engine performs only positive test and, therefore, when a negate operator is used, it performs a positive test internally and negates the result of the test.

## Example

```
Location/Addr/StateProvCd NOT= 'MA'
```

This is internally evaluated as:

```
NOT (Location/Addr/StateProvCd = 'MA').
```

It's important to note that it is not evaluated as:

```
Location/Addr/StateProvCd != 'MA'
```

# Workflow Management

The framework has the built-in capability to carry out workflow operations that are thought to be quite common place, which include the following:

- Change the status of a work item
- Assign ownership of a work item to a different user or work group, also referred to as an assignee
- Generate an error, warning or informational message to display on the page

Most workflow operations are conditionally executed based on the outcome of a rule or some other business logic. To this end, the workflow feature supports that the execution of workflow operations (aka instructions) based on the outcomes of connectors.

AgencyPortal's workflow management feature and [connectors](#) go hand-in-hand. Workflow operations are configured for a connector set in an entity called an instruction. Instructions, like connectors, can be modified with DTR behaviors.

## Connector Outcome

Instructions can be conditionally executed based on the outcome of one or more connectors. The `IConnectorManager` interface forces all `ConnectorManager` implementations to return an outcome. An outcome can be one of two states represented by the constants.

- `Outcome.CONDITIONS_MET` – The connector wants instructions that depend upon it to be executed.
- `Outcome.CONDITIONS_NOT_MET` – The connector does not want instructions that are dependent upon it to be executed.

Built-in `ConnectorManager` implementations that process XARC rules and server-side validations return outcomes, too. XARC connectors will return an outcome of `CONDITIONS_MET` if any rule executed by the connector has all of its conditions satisfied.

### Note

Many believe that a connector's outcome has something to do with the success or failure of the connector. This is a common misinterpretation; connector outcomes are abstract. They relate only to the execution of instructions that are dependent upon the connector and they have no bearing on the execution of subsequent connectors. If there are no instructions that depend on the connector, the outcome returned by that connector is meaningless and, therefore, ignored by the `ConnectorProcessor`.

### Example

This example illustrates three workflow instructions that are predicated on the outcome of one connector:

```
<connectors type="process"><connector id="generalInfo_Page" type="XARC"><xarcRules id="X1" ruleLibraryId="generalInfo" /></connector><instruction id="I0" type="generateMessage" value="This quote will be referred" severity="warning"><when connector="generalInfo_Page" /></instruction><instruction id="I1" type="statusChange" value="REFER"><when connector="generalInfo_Page" /></instruction><instruction id="I3" type="assign"><when connector="generalInfo_Page" /><target principal="underwriter" type="user" /></instruction></connectors>
```

If the XARC connector were to run and it returned an outcome of `CONDITIONS_MET`, then the following three instructions will run:

- The first will create a referral message that displays on the application
- The second will change the work item status to a referred status
- The third will assign the work item to the user whose login ID is underwriter

A connector's outcome is separate and distinct from whether a connector throws an error condition. Although they can be related, they are treated separately. The presence of an error message in `MessageMap` after a connector execution, along with the `ConnectorManager`'s `postProcess` implementation, governs whether to continue with subsequent connectors. This is independent from the outcome return value emitted by a `ConnectorManager` implementation. The latter is what is used to trigger workflow instructions; the former is used to determine whether subsequent connectors should be executed.

### Note

An instruction that is not dependent on the outcome of any connectors (i.e., the `<instruction>` aggregate in the TDF/page library does not have a `<when connector="xyz_connector"/>` child element defined), the instruction will always be executed.

## Workflow Processing

Processing of workflow operations is handled by a component that implements the interface `com.agencyport.workflow.IWorkflowManager`. AgencyPortal is accompanied with a basic implementation of that interface with the

Java class `com.agencyport.workflow.WorkflowManager` that is engaged with no extra configuration. It supports all of the built-in instruction types of `generateMessage`, `statusChange`, `assign`, `updateServiceData` (refer to the [Timeline](#) topic for more information on this type) and `updateAPData`. The basic implementation for the `assign` operation assumes that the application's ACSI `ISecurityProfileManager` returns an `ISecurityProvider` implementation that can resolve subject and user group lookup requests by login ID and group name respectively.

Custom instructions can be configured, which the base implementation will silently ignore. Applications must extend the base class to support the carrying out of custom instructions. To make your custom workflow manager known to the system, this is done via an attribute on the "connectors" element:

```
<connectors type="process" workflowManagerClassName="test.agencyport.workflow.CustomWorkflowManager"> <connector id="generalInfo_Page" type="XARC" ><xarcRules id="X1" ruleLibraryId="generalInfo" /> </connector> <instruction id="II" type="custom" > <when connector="generalInfo_Page" /> <customParameters><![CDATA[custom configuration goes here]]></customParameters> </instruction> </connectors>
```

```
package test.agencyport.workflow; import org.jdom.Element; import com.agencyport.connector.model.Instruction; import com.agencyport.workflow.WorkflowContext; import com.agencyport.workflow.WorkflowException; import com.agencyport.workflow.WorkflowManager; import com.agencyport.workitem.model.IWorkItem; /** * The CustomWorkflowManager class illustrates how to extend the basic workflow manager * class to support a custom instructions. */ public class CustomWorkflowManager extends WorkflowManager { /* (non-Javadoc) * @see com.agencyport.workflow.WorkflowManager#carryOutCustom(com.agencyport.connector.model.Instruction) */ @Override public void carryOutCustom(Instruction instruction) throws WorkflowException { // We really don't do anything in this example but show how to // get around. Element instructionElement = instruction.getMetaData(); // the sky is the limit in terms of what can be configured on the <instruction> Element customParameters = instructionElement.getChild("customParameters"); String value = customParameters.getText(); // extract custom parameter value WorkflowContext context = this.getContext(); IWorkItem workItem = context.getWorkItem(); // Here we update the work item only and leave the database update to the framework. workItem.setPropertyValue("custom_Property", value); } }
```

## Workflow Management Database Updates

The base workflow manager implementation coordinates database update operations against the work item table(s), the audit trail log and, in the case of `updateAPData` instruction types, the `xmlstore` table. In the case of an error situation, or a display side connector scenario, an instruction may be run because of an outcome. If that is the case, the commit to the database is immediate once all of the instructions within a connector set have finished executing. If an instruction is fired in a non-error situation, the commit to the database is part of the unit of work that controls updates to the `xmlstore`, `workitem` and precondition realms controlled by AgencyPortal's data engine.

For custom persistence, the workflow manager's `persist()` method must be overridden.

### Example

```
public class CustomWorkflowManager extends WorkflowManager { /* (non-Javadoc) * @see com.agencyport.workflow.WorkflowManager#persist() */ @Override public void persist() throws WorkflowException { // We most certainly want the work item table to be updated super.persist(); try { final String updateSQL = "update custom_table set custom_column = ? where work_item_id = ?"; DatabaseResourceAgent dra = getDatabaseResourceAgent(); // Get addressability to the same connection the framework is using. dra.getConnection(false); WorkflowContext context = this.getContext(); IWorkItem workItem = context.getWorkItem(); // The database resource agent will track the prepared statement and will be released by // the framework. The example here is entirely a fabrication. We are taking a work item property and saving to another table PreparedStatement pStmt = dra.getPreparedStatement(updateSQL); // Get some custom value set by the customaction that will be written to this special custom table that the carryOutCustom method initialized. String customValue = context.getControlData().get("MY_CUSTOM_VALUE"); pStmt.setString(1, customValue); pStmt.setInt(2, workItem.getWorkItemId().intValue()); pStmt.executeUpdate(); } catch (Exception exception){ throw new WorkflowException(exception); } }
```

# Views

Views are used to support complex relationships when there is not a one-to-one relationship between a field on the page and a target in the data store. In another way, a view is a construct that allows a separation between how data is stored and how it is presented.

The default SDK data engine is good at processing XML values when there is a one-to-one relationship between what is rendered on the page and the underlying representation in the XML. However, there are several cases where this one-to-one relationship does not fit.

- Dates need to be stored in the ACORD data format (yyyy-MM-dd), but you may need to display and capture them in some other format (e.g., dd/MM/yyyy).
- For coverage limits, combined single limits and splits could be rendered with one field, but can have up to four potential spots in the XML that need processed to render a compliant ACORD structure. In short, if you pick a CSL, then the split limits need to be deleted and vice versa since they are typically treated as mutually exclusive entities.
- For code value, code descriptions are combined as a single item in a select list. The underlying ACORD structure treats them as two discreet element values.
- When looking at the overview section of a roster page, the display lines may need to be rendered in a way that a single column on a roster line is composed of the aggregation of several atomic values delimited with various static characters. For instance, an address is typically composed of an address line, city, state and zip code.
- There are cases where an empty value from the page should actually trigger a delete of an XML element hierarchy rather than just store an empty value in the XML element value. An example of this scenario is ACORD coverages. Typically, there is an option list selection, such as Select One, or an empty edit value that indicates to delete the coverage altogether.
- Other cases where the relationship between the value associated with the visual rendering and the underlying XML representation in the underlying data model is a one-to-many relationship. The use of the Agencyport XML schema typically exacerbates this issue.

You can accomplish the processing of each of these scenarios by writing custom code on both the display and process side. However, the SDK supports a view layer infrastructure on top of the underlying XML framework that accomplishes the same goals; thereby, reducing the number of times a developer must turn to custom code.

## Note

As an alternative to views, consider simplifying your XML schema by using the [ACORD to Simplified technique](#). Theoretically, the schema could be simplified to the degree that there are only one-to-one relationships between the fields on the page and their representations within XML.

After referenced by a field in the TDF, views are automatically engaged by the display and process side of the page. A view performs the following three core operations:

- Read - read the APDataCollection, prepare the data for your field.
- Update - update the APDataCollection, given incoming data from your field after a page submission
- Delete - delete the data from APDataCollection, given no incoming data from your field after a page submission (or an empty value if the deleteMode attribute is set to onEmptyValue)

To resolve the (complex) relationship between your field and the data, views give you the ability to specify what steps the SDK should take when processing the above operations.

## Types of Views

There are three different types of views:

- standard
- display
- mutually exclusive

### Standard

Standard is the most typical view type. It should be used when there is only one way to store the data and when there is a need to have the framework perform read, update and delete operations. You provide one fieldSet element with specific instructions on how to perform the read, update and delete procedures.

### Example

In this example, we have one field in the UI whose intent is to capture a person's name, but we need to store the different parts of the name under discreet elements in the XML. For this example, let's assume that the name will only ever be two parts to a name (first and last) and it will always be entered in a format like "John Smith."

```
<fieldElement type="text" label="Applicant Name" viewId="commonViews.xml:PersonalLineConcatenateEntityName"
id="InsuredOrPrincipal[InsuredOrPrincipalInfo.InsuredOrPrincipalRoleCd='Insured'].GeneralPartyInfo.NameInfo.PersonName"
uniqueId="ApplicantName" required="true" size="30" maxLength="40"> <fieldHelper type="balloon">Enter the Insured's first and last name in
this format "John Smith".</fieldHelper> </fieldElement>
```

The view is attached to the field using the `fieldElement`'s `viewId` attribute. This attribute must be specified in the format `viewFilename:idOfViewInViewFile`. The view file must also be registered as a view artifact under the product within the product database.

Remember, the goal is to take a submitted value, such as "John Smith," and store it in XML like the following:

```
<PersonName> <GivenName>John</GivenName> <Surname>Smith</Surname> </PersonName>
```

The following is the view:

```
<view id="PersonalLineConcatenateEntityName" title="Entity Name" type="standard"><fieldSet> <field
type="id">InsuredOrPrincipal[InsuredOrPrincipalInfo.InsuredOrPrincipalRoleCd='Insured'].GeneralPartyInfo.NameInfo.PersonName.GivenNa
me</field> <field type="formatInfo"></field> <field
type="id">InsuredOrPrincipal[InsuredOrPrincipalInfo.InsuredOrPrincipalRoleCd='Insured'].GeneralPartyInfo.NameInfo.PersonName.Surname<
/field> </fieldSet> </view>
```

For a read operation, the `fieldSet` contains the paths to read along with formatting characters to go in between the values when they are concatenated together for display. For an update, the same `fieldSet` information is used for instructions on how to break apart the input value of "John Smith" into its two parts for storage in two different locations. For a delete operation, we did not provide any `deleteField` instructions, so the framework won't know what to delete. This could be okay under certain circumstances, such as you know the field is required and will never be excluded by behaviors, or if you're relying on data cleanup to delete these fields using group IDs.

## Note

The field's ID attribute is normally what's used to tell the framework where to store the field's data in the XML. However, since we have a view in effect in the above example, the view takes care of data storage and the ID becomes less important. As a best practice, provide an ID attribute that points to some logical and relevant path (like the parent aggregate of the name parts, in this case). This is because the ID could still be used in some cases (it gets passed along to the underlying view object in Java as the `associatedElementPath`).

On roster pages, the ID must begin with the page's roster source to have the view inherit the appropriate index management state. For example:

```
<page id="vehicle" source="PersAutoLineBusiness.PersVeh" title="Vehicles" type="roster" forceViewOnUpload="true" > <pageElement
type="fieldset" legend="Safety Features" id="safetyFeaturesMA"><fieldElement type="selectlist"
id="PersAutoLineBusiness.PersVeh.AntiTheftDeviceInfo[AntiTheftDeviceCd='Y']" label="Anti-Theft Devices"
viewId="personalAutoViews.xml:PersonalAuto.AntiTheftDevice" required="true" defaultValue="" > <optionList reader="xmlreader"
source="codeListRef.xml" target="selectOne"/><optionList reader="xmlreader" source="codeListRef.xml" target="boolean"/></fieldElement>
</pageElement> </page>
```

## Mutually Exclusive

Mutually exclusive views are used in situations where there are two or more different ways to store the data (given different input values). These are almost exclusively used with `OptionLists` under circumstances where different options under the same list need to be stored differently. The classic use case for this feature is the Combined Single Limit field scenario. The following example provides details on this scenario:

## Example

```
<optionList id="liability"> <option value="50000">$50,000 CSL</option> <option value="200000">$200,000 CSL</option> <option
value="20000/40000/10000">$20,000/$40,000/$10,000</option> <option value="25000/50000/25000">$25,000/$50,000/$25,000</option>
<option value="50000/100000/10000">$50,000/$100,000/$10,000</option> <option
value="100000/200000/20000">$100,000/$200,000/$20,000</option> </optionList>
```

The values of the first two options are in a different format than the others. You can infer that the view is going to need a different `fieldSet` for handling the first two values. You'll also need to tell the framework how to identify which `fieldSet` to use given the different input formats that are expected.

```
<view deleteMode="onEmptyValue" type="mutuallyExclusiveFieldSets" id="BOPLineBusiness.LiabilityInfo.CommlCoverage.BIPDXLCSL">
<fieldSet searchId="/"><field
type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='BI'].Limit[0].FormatCurrencyAmt.Amt</field> <field
type="formatInfo"/></field> <field
type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='PD'].Limit[0].FormatCurrencyAmt.Amt</field> <field
type="formatInfo"/></field> <field
type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='XL'].Limit[0].FormatCurrencyAmt.Amt</field>
```

```

<deleteField>BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='BI']</deleteField>
<deleteField>BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='PD']</deleteField>
<deleteField>BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='XL']</deleteField> </fieldSet> <fieldSet searchId="none"><field
type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='CSL'].Limit[0].FormatCurrencyAmt.Amt</field>
<deleteField>BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='CSL']</deleteField> </fieldSet> </view>

```

The `searchId` attribute tells the framework which `fieldSet` to use for a given input value. In the above example, the `searchId` on the first `fieldSet` is a / (forwardslash) character. This tells the framework to use this `fieldSet` when provided with a data value that contains /. In this case, if the user selects the 3rd, 4th, 5th or 6th options in the drop-down, this `fieldSet` is engaged and the split limit value tokenizes into three parts and stores at three different paths in the XML. Otherwise, if there is no /, the second `fieldSet` where `searchId` is set to none is used and the value under a CSL coverage limit path is stored without any tokenization.

You can get more creative with the `searchId`. In the above example, the default `searchId` behavior is used, which is to check the inputted data value (e.g., 20000/40000/10000) for the presence of some token (such as a / character). If you're looking to use more complex patterns as the distinguishing factor, such as the presence of multiple slashes, then you can use the `treatSearchIdAsRegex` flag to have your `searchId` interpreted as a regular expression. Refer to the following example:

### Example

```

<view id="FarmLineBusiness.Structure.Coverage[CoverageCd='STCTR'].Deductible[DeductibleAppliesToCd='WindHail']"
type="mutuallyExclusiveFieldSets" class="basic" deleteMode="onEmptyValue" title="Wind Deductible View"> <description>Storing values less than 100 at the percentage XPATH.</description> <fieldSet searchId="(\d{0,2}|100)" treatSearchIdAsRegex="true"> <field
type="id">FarmLineBusiness.Structure.Coverage[CoverageCd='STCTR'].Deductible[DeductibleAppliesToCd='WindHail'].FormatPct</field>
<deleteField>FarmLineBusiness.Structure.Coverage[CoverageCd='STCTR'].Deductible[DeductibleAppliesToCd='WindHail'].FormatPct</delet
eField> </fieldSet> <fieldSet searchId="none"><field
type="id">FarmLineBusiness.Structure.Coverage[CoverageCd='STCTR'].Deductible[DeductibleAppliesToCd='WindHail'].FormatInteger</field>
>
<deleteField>FarmLineBusiness.Structure.Coverage[CoverageCd='STCTR'].Deductible[DeductibleAppliesToCd='WindHail'].FormatInteger</d
eleteField> </fieldSet> </view>

```

### Display

Display is the same as standard, except there is no update and no delete operation. The primary use case is the roster page overview section where you want your columns to list aggregated, formatted data.

## View Classes

Distinct from view type, there is also the notion of a view class. You can think of a view class as a subtype. For each type of view, the following classes apply:

- Standard
  - Basic
  - AggregateExists Boolean - used with fields not stored in the data store when a Boolean false is selected.
  - Remarks - used to store remarks with a reference to a foreign aggregate.
  - Date Format - used to store a date in the ACORD format (YYYY-MM-DD) and read the date back to the screen format.
  - Custom - coded by a developer when none of the provided views will satisfy a requirement (you must provide a `viewClassName` element).
- Mutually Exclusive - views that require data to be stored in multiple locations in the XML document.
  - Basic
  - Custom - coded by a developer when none of the provided views will satisfy a requirement (you must provide a `viewClassName` element).
- Display
  - Basic
  - Custom - coded by a developer when none of the provided views will satisfy a requirement (you must provide a `viewClassName` element).

## Views in Java

### Programmatically Invoking a View

Views are automatically engaged by the framework during the display and processing of a page on the UI. However, there are some cases when you are working in Java and you need to manually invoke a view. You can do so as follows:

```
IView view = ViewRepository.get().getView(new ViewKey("common Views.xml", "PersonalLineConcatenateEntityName"));
String displayVal = view.read(apData, null, "", "");
```

## Creating a Custom View

Custom view classes must implement the `IView` interface; however, in most cases, subclassing the framework's base view class (`com.agencyport.domXML.view.View`) is the best approach. When you override the base view class, you can focus on customizing the read, update and delete operations with overrides.

### Example 1

In this example, there is a need for a custom view that will adjust the XPath for a field that is based on the select list option from another field.

The `AlternateSearchIdView` actual view in the AgencyPortal 5.2 template core illustrates this scenario. This view takes advantage of a feature in AgencyPortal 5.2 that allows access to the HTML data container, which is something that was not available in previous versions of the framework.

The business scenario under consideration is the limit amount for hired, non-owned coverage on the BOP LOB. The coverage code that qualifies this limit is HRDBD, NOWND or HNA, depending on the coverage type code field that the user selects on the preceding field.

You will notice this looks like your typical mutually exclusive fieldset based view except for the `searchId` custom argument, which identifies the ACORD ID of the field that should be used to select the correct field set in the view's metadata.

Hired and Non-Owned Auto Liability

Coverage Type

Hired/Borrowed

Coverage Not Requested

Hired/Borrowed

Non Owned

Hired/Nonowned

Limit

\$300,000

\$300,000

\$500,000

\$1,000,000

\$2,000,000

This coverage adds an exception to liability exclusion for the activity described below.

The following is the field element and select list configuration for the coverage type code field that this field depends on:

```
<fieldElement type="selectlist" id="HiredNonOwnedAutoLiability.CoverageCd" label="Coverage Type"
viewId="bopViews.xml:BOPLineBusiness.LiabilityInfo.CommlCoverage.HIRED_NON_OWNED.CoverageCode"
uniqueId="hiredNonOwnedAutoLiabilityCoverageCd"> <optionList reader="xmlreader" source="codeListRef.xml"
target="coverageNotRequested" /><optionList reader="xmlreader" source="bopCodeListRef.xml"
target="hiredAndNonOwnedAutoLiabilityCoverageType" /></fieldElement>
```

```
<optionList id="hiredAndNonOwnedAutoLiabilityCoverageType"> <option value="HRDBD">Hired/Borrowed</option> <option
value="NOWND">Non Owned</option> <option value="HNA">Hired/Nonowned</option> </optionList>
```

The following is the field element and view metadata for the coverage limit itself:

```
<fieldElement type="selectlist" id="HiredNonOwnedAutoLiability.Limit" label="Limit"
viewId="bopViews.xml:BOPLineBusiness.LiabilityInfo.CommlCoverage.HIRED_NON_OWNED.Limit"
uniqueId="hiredNonOwnedAutoLiabilityLimit"> <optionList reader="xmlreader" source="codeListRef.xml" target="blank" /><optionList
reader="xmlreader" source="bopCodeListRef.xml" target="hiredAndNonOwnedAutoLiabilityLimit" /></fieldElement>
```

```
<view id="BOPLineBusiness.LiabilityInfo.CommlCoverage.HIRED_NON_OWNED.Limit" title="View for handling hired/non owned
coverage limit" type="mutuallyExclusiveFieldSets" deleteMode="onEmptyValue"> <customArgument name="searchId"
```

```

value="HiredNonOwnedAutoLiability.CoverageCd" /> <fieldSet searchId="HRDBD"> <field
type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='HRDBD'].Limit.FormatInteger</field> <deleteField
type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='HRDBD']</deleteField> </fieldSet> <fieldSet searchId="NOWND">
<field type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='NOWND'].Limit.FormatInteger</field> <deleteField
type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='NOWND']</deleteField> </fieldSet> <fieldSet searchId="HNA">
<field type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='HNA'].Limit.FormatInteger</field> <deleteField
type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='HNA']</deleteField> </fieldSet>
<viewClassName>com.agencyport.shared.custom.AlternateSearchIdView</viewClassName> </view>

```

The following is the custom view source. Note how the custom argument search ID is used:

```

/* * Created on Aug 13, 2014 by nbaker AgencyPort Insurance Services, Inc. */ package com.agencyport.shared.custom; import java.util.Map;
import org.jdom2.Element; import com.agencyport.domXMLAPDataCollection; import com.agencyport.domXML.view.View; import
com.agencyport.shared.APException; import com.agencyport.webshared.HTMLDataContainer; import
com.agencyport.workitem.context.WorkItemContextStore; /** The AlternateSearchIdView class supports a search id for mutually exclusive
field sets that is different from the data value itself. * @since 5.0 */ public class AlternateSearchIdView extends View { /** The
<code>serialVersionUID</code> */ private static final long serialVersionUID = 1377654133640169919L; /** The
<code>SEARCH_ID</code> is a constant for the value 'searchId'. */ private static final String SEARCH_ID = "searchId"; /** The
<code>searchId</code> is the search id value. This is used as a key into the HTML data container. */ private String searchId; /** Constructs an
instance. */ public AlternateSearchIdView() { } /** {@inheritDoc} */ @Override public void init(Element viewElement) throws APException
{ super.init(viewElement); // initialize superMap<String, String> customArguments = CustomViewParser.parseView(viewElement, false,
SEARCH_ID); searchId = customArguments.get(SEARCH_ID); } /** {@inheritDoc} */ @Override public void update(APDataCollection
apData, int[] idArray, String value, String associatedElementPath) { HTMLDataContainer htmlDC =
WorkItemContextStore.get().getHTMLDataContainer(); String searchIdValue = htmlDC.getStringValue(searchId, null); update(apData, idArray,
value, searchIdValue, associatedElementPath); }

```

## Example 2

In this example, there is a need for a custom view that derives the value of one field based on the value of another field multiplied by some factor.

Formula: final XML value = (*dependent field value* \* *a multiplier factor from a selection option*).

The business scenario: The Prds/Cmp Ops liability coverage limit in BOP is computed by taking the liability occurrence limit and multiplying it by a factor (typically either 2x or 3x). In the case that follows, if the user selects the 1,000,000/2,000,000 option for the each occurrence liability limit and select 2x in the Prds/Cmp Ops multiplier, then a value of 2,000,000 (2x1,000,000) should be stored in the Prds/Cmp Ops liability limit field in the resultant XML.

The following is the field, its select list options, view definition, as well as the Java source supporting this advanced data management challenge:

```

<fieldElement type="selectlist" id="PRDCO.Multiplier" label="Prds/Cmp Ops Agg Mult" required="true" size="10" defaultValue="2"
viewId="bopViews.xml:BOPLineBusiness.LiabilityInfo.CommlCoverage.PRDCCO"> <optionList reader="xmlreader" source="codeListRef.xml"
target="selectOne" /> <optionList reader="xmlreader" source="bopCodeListRef.xml" target="prdCOMultiplier" /></fieldElement>

<optionList id="prdCOMultiplier"> <option value="2">2x</option> <option value="3">3x</option> </optionList>

<view id="BOPLineBusiness.LiabilityInfo.CommlCoverage.PRDCCO" title="Multiplier View for PRDCO coverage" type="standard"
deleteMode="onEmptyValue"> <customArgument name="source"
value="BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='EAOCC'].Limit.FormatInteger"/> <fieldSet> <field
type="id">BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='PRDCO'].Limit.FormatInteger</field>
<deleteField>BOPLineBusiness.LiabilityInfo.CommlCoverage[CoverageCd='PIADV']</deleteField> </fieldSet>
<viewClassName>com.agencyport.bop.custom.MultiplierView</viewClassName> </view>

```

```

/**Created on Aug 13, 2014 by nbaker AgencyPort Insurance Services,
Inc.*/ package com.agencyport.bop.custom; import java.util.Map; import org.jdom2.Element; import com.agencyport.domXMLAPDataCollection;
import com.agencyport.domXML.view.Field; import com.agencyport.domXML.view.View; import com.agencyport.fieldvalidation.validators.builtin

```

```

.NumericValidator;importcom.agencyport.shared.APException;importcom.agencyport.shared.custom.CustomViewParser;/**TheMultiplierView
wclassmanagestheupdateanddisplayoffieldswhosevaluesaredeterminedbyamultiplier*factoruponanotherdataentityintheworkitem.Onupdate,
result=multiplierfactor*source.Ondisplay,
*multiplierfactor=result/source.Themultiplierfactoristhevaluewhichispassedtothisclass'supdatemethod*bytheframeworkandthereturnvalueofthiscla
ss'readmethod.*@since5.0replacementforoldSFHsOccurrenceMultiplierHelperandPersAdvertisingInjuryHelper*/publicclassMultiplierViewexten
sView{
/**The<code>serialVersionUID</code>*/privatestaticfinallong serialVersionUID=8299687731656501042L;/**The<code>SOURCE</code>i
sthenameofthecustomattributethatlinkstotherelatedsource.*privatestaticfinalStringSOURCE="source";/**The<code>ZERO</code>isaconstantf
orastringversionofthevalue0.*privatestaticfinalStringZERO="0";/**The<code>sourceId</code>istheidforthescourceentityuponwhichthisand
thefactorismultipliedtoarriveattheresult.*privateStringsourceId;/**The<code>viewElement</code>isareferencetotheXMLviewmetadata.*priv
ateElementviewElement;/**The<code>NON_NUMERIC_ERROR_MSG_TEMPLATE</code>isanerrormsgtemplateusedwhennonnumericoper
andsareencountered.*privatestaticfinalStringNON_NUMERIC_ERROR_MSG_TEMPLATE="The view requires numeric operands. One or
both of the operands were determined not to be numeric: %s %s";/**Constructsinstance.*publicMultiplierView(){ }/**{@inheritDoc
}*/@Overridepublicvoidinit(ElementviewElement)throwsAPException{
super.init(viewElement);//initializesuperthis.viewElement=viewElement;Map<String,
String>customArguments=CustomViewParser.parseView(viewElement, false, SOURCE);sourceId=customArguments.get(SOURCE); }/**{
@inheritDoc }*Weconsideravalueofzeroas"empty"*/@OverridepublicbooleanisEmptyValue(Stringvalue){
returnField.isEmptyValue(value)||ZERO.equals(value); }/**{@inheritDoc }*/@Overridepublicvoidupdate(APDataCollectionapData,
int[]idArray, Stringvalue, StringassociatedElementPath){ StringmultiplierFactorValue=value;if(!EmptyValue(multiplierFactorValue)){
StringsourceValue=apData.getFieldValue(sourceId, idArray, ZERO);if(!EmptyValue(sourceValue)){
if(!NumericValidator.isInteger(sourceValue)||!NumericValidator.isInteger(multiplierFactorValue)){ try{
CustomViewParser.raiseException(String.format(NON_NUMERIC_ERROR_MSG_TEMPLATE, sourceValue, multiplierFactorValue),
viewElement); }catch(APExceptionapException){ thrownewIllegalArgumentException(apException); }
}intmultiplierFactor=Integer.parseInt(multiplierFactorValue);intsource=Integer.parseInt(sourceValue);inttarget=source*multiplierFactor;super.u
pdate(apData, idArray, Integer.toString(target), associatedElementPath);return; } }super.update(apData, idArray, value,
associatedElementPath);/*ThiswilldeletetheentityduetoouroverloadofthemethodisEmptyValue() *//**{@inheritDoc
}*/@OverridepublicStringread(APDataCollectionapData, int[]idArray, StringdefaultValue, StringassociatedElementPath)throwsAPException{
StringtargetValue=super.read(apData, idArray, defaultValue, associatedElementPath);if(!EmptyValue(targetValue)){
StringsourceValue=apData.getFieldValue(sourceId, idArray, ZERO);if(!EmptyValue(sourceValue)){
intsource=Integer.parseInt(sourceValue);inttarget=Integer.parseInt(targetValue);intmultiplier=target/source;returnInteger.toString(multiplier); }
}returnZERO; } }

```

# View File

The following table describes the various elements in a view file:

Artifact	XML Node Type	Parent Element	Description	Validation
Views	Element	None	Root element	None
view	Element	Views	Defines a view instance  Repeats	None
id	Attribute	view	Specifies the view ID	Must be unique within a view XML file.
type	Attribute	view	Specifies the view type	<p>Can be one of these values:</p> <ul style="list-style-type: none"> <li>• standard</li> <li>• mutuallyExclusiveField Sets</li> <li>• display</li> </ul> <p>Standard types can have only one field set; mutually exclusive field sets must have two field sets</p>
deleteMode	Attribute	view	Determines the implied delete action when an empty value is passed in	Can only be onEmptyValue
fieldSet	Element	View	Defines the start of a field set	
searchId	Attribute	fieldSet	Specifies the distinguishing character or string for determining the correct fieldSet to apply in a mutually exclusive field set scenario	<p>Required for mutually exclusive field set situations only.</p> <p>Ignored for standard field sets.</p> <p>Special values:</p> <ul style="list-style-type: none"> <li>• none - means that there is no distinguishing character. This can be used for one of the field sets in a mutually exclusive scenario, but not for both. At least one field set in a mutually exclusive field set situation must be associated with an input value that contains a distinguishing character or string.</li> </ul>

<b>Artifact</b>	<b>XML Node Type</b>	<b>Parent Element</b>	<b>Description</b>	<b>Validation</b>
treatSearchIdAsRegEx	Attribute	fieldSet	Specifies whether the <code>searchId</code> attribute is interpreted as a regular expression	Optional - when provided, it is ignored unless you're in a <code>mutuallyExclusive</code> <code>fieldSet</code> situation.
customArgument	Element	fieldSet	Can be used to provide some custom namevalue arguments, which are interpreted by the custom view class	Optional, repeating - only useful in custom view situation.
name	Attribute	customArgument	The name of the custom argument	Required - type: string.
value	Attribute	customArgument	The value of the custom argument	Required - type: string
field	Element	fieldSet	Specifies a field that is used for read and update purposes	
type	Attribute	field	Specifies what kind of data is contained in the value of the <code>field</code> element	<p>Can be one of the following:</p> <ul style="list-style-type: none"> <li>• <code>id</code> (default if not specified) - means that a valid element path is specified in the value of this field value.</li> <li>• <code>formatInfo</code> - means that the value in this field should be interpreted as static text.</li> </ul> <p>Also note that there must be a field of type <code>formatInfo</code> in between fields of type <code>ID</code>.</p>
deleteField	Element	fieldSet	Specifies an element path that is used for deletes. Typically there is an implied delete in an update situation when an empty value is passed in.	
type	Attribute	deleteField	Specifies what kind of data is contained in the value of the <code>field</code> element	Can only be the value <code>ID</code> (default if not specified), which means that a valid element path is specified in the value of this field value.
viewClassName	Element	view	Specifies a view implementation class other than the implicit default of <code>com.agencyport.domXML.view.View</code>	This class can either inherit from the <code>com.agencyport.domXML.view.View</code> base class or implement the <code>com.agencyport.domXML.view.IView</code> interface from scratch. When this element is missing, the default implementation is the

Artifact	XML Node Type	Parent Element	Description	Validation
				<p>com.agencyport.domXML.view.View implementation.</p> <p>The following are the implementation views available out of the box from the com.agencyport.domXML.view package:</p> <ul style="list-style-type: none"> <li>• View - default implementation that provides the base functionality of many view implementations.</li> <li>• AggregateExistsView - subclasses viewbase class and links the use of a Boolean UI control of Yes/No to the existence of an aggregate (e.g., coverage). Please note that this view inherits from view and NOT the TransformSingleValueView. Because of this, the field ID for this view must come from the view field and not the transaction definition file, and there must be a unique AggregateExistsView file for each unique related aggregate.</li> <li>• RemarksView - subclasses View Base class and provides the "id reffing" update functionality to the APDataCollection.</li> <li>• TransformSingleValueView - fully implements the IView and is targeted toward scenarios where there exists the usual one to one relationship between the UI control and a simple transformation that needs to take place on either the display side or the update side. This view type can also be used to issue smart deletes on empty element values. This uses the element path field ID from the</li> </ul>

Artifact	XML Node Type	Parent Element	Description	Validation
				<p>transaction definition file rather than a field set specification in the view.</p> <ul style="list-style-type: none"> <li>• DateFormatView - subclasses the TransformSingleValueView and supports the notion of differing display and storage formats for the date.</li> <li>• UpperCaseView - subclasses the TransformSingleValueView and supports the notion of upper casing the page value before it is stored in APDataCollection.</li> </ul>

# Example View File

```

>PersAutoLineBusiness.PersVeh[0].Coverage[CoverageCd="UNDSP"].Limit[LimitAppliesToCd='PerPerson'].FormatInteger</field> <field
type="formatInfo"></field> <field type="id"
>PersAutoLineBusiness.PersVeh[0].Coverage[CoverageCd="UNDSP"].Limit[LimitAppliesToCd='PerAcc'].FormatInteger</field> </fieldSet>
</view> <view id="PersonalAuto.UnderinsuredBodilyInjury.MA.Limit.Amt" type="standard" deleteMode="onEmptyValue"> <fieldSet
searchId="/"><field type="id"
>PersAutoLineBusiness.PersVeh.Coverage[CoverageCd="UNDSP"].Limit[LimitAppliesToCd='PerPerson'].FormatInteger</field> <field
type="formatInfo"></field> <field type="id"
>PersAutoLineBusiness.PersVeh.Coverage[CoverageCd="UNDSP"].Limit[LimitAppliesToCd='PerAcc'].FormatInteger</field> <deleteField
type="id">PersAutoLineBusiness.PersVeh.Coverage[CoverageCd="UNDSP"]</deleteField> </fieldSet> </view> <view
id="PersonalAuto.RecoverySystem" type="standard"> <fieldSet> <field
type="id">PersAutoLineBusiness.PersVeh.AntiTheftDeviceInfo[AntiTheftDeviceCd='E']</field> </fieldSet>
<viewClassName>com.agencyport.domXML.view.AggregateExistsBooleanView</viewClassName> </view> <view
id="PersonalAuto.AntiTheftDevice" type="standard"> <fieldSet> <field
type="id">PersAutoLineBusiness.PersVeh.AntiTheftDeviceInfo[AntiTheftDeviceCd='Y']</field> </fieldSet>
<viewClassName>com.agencyport.domXML.view.AggregateExistsBooleanView</viewClassName> </view> <view
id="PersonalAuto.Default.WaiverOfCollisionDeductible" type="standard"> <fieldSet> <field
type="id">PersAutoLineBusiness.PersVeh[0].Coverage[CoverageCd="CWAIV"]</field> </fieldSet>
<viewClassName>com.agencyport.domXML.view.AggregateExistsBooleanView</viewClassName> </view> <view
id="PersonalAuto.PersVeh.WaiverOfCollisionDeductible" type="standard"> <fieldSet> <field
type="id">PersAutoLineBusiness.PersVeh.Coverage[CoverageCd="CWAIV"]</field> </fieldSet>
<viewClassName>com.agencyport.domXML.view.AggregateExistsBooleanView</viewClassName> </view> <view id="Agent.Comments"
type="standard" relatedAggregateId="PersPolicy.@id" index="0" apType="AgentComments" deleteMode="onEmptyValue"> <fieldSet> <field
type="id">RemarkText</field> </fieldSet> <viewClassName>com.agencyport.domXML.view.RemarksView</viewClassName> </view>
<view id="Questionnaire.Remarks" type="standard" relatedAggregateId="PersAutoLineBusiness.@id" index="0"
apType="GeneralQuestionnaire" deleteMode="onEmptyValue"> <fieldSet> <field type="id">RemarkText</field> </fieldSet>
<viewClassName>com.agencyport.domXML.view.RemarksView</viewClassName> </view> </views>

```

# Product Database

Product database is an XML configuration file (productdatabase.xml), which is a means of defining and loading products and artifacts into AgencyPortal.

The path of the productdatabase.xml is defined by the application property `resource_root` and has a default value of "WEB-INF" folder inside the war file.

The product database is loaded into the application as part of the boot services.

`com.agencyport.bootservice.ProductDefinitionsBootService` and  
`com.agencyport.product.ProductDefinitionsManager` are the classes that deals with this.

Products in the productdatabase.xml are typically organized along the same lines as lines of business (LOB codes), but it can also be something like 'shared' or 'commercial'.

The following artifact resource types are support by the product out of the box:

- tdf: Transaction definitions
- pageLibrary: Page libraries consisting of sharable page definitions
- view: Resource type for view files
- optionList: Resource type for select lists
- dynamicListTemplate: Resource type for dynamic list templates
- behavior: Behavior files resource type.
- pdfLayout: layout files used for PDF generation
- xarcRule: resource type for XARC rule files.
- propertyFile: resource type LOB specific property files
- pdf: resource type for ACORD forms (or any other PDF's)
- sfh: Special field helper resource type
- acordWorkItemMap: configuration file for extracting workitem data from ACORD XML
- workItemAssistant: workitem assistant configuration file
- transformer: XSLT transformers
- workListDefinition: configuration for work lists
- workItemActionDefinition: configuration file that defined the actions associated with workitem statuses
- menuDefinition: configuration for menu entries
- workItemStatusDefinition: workitem statuses available

```
<productDatabase><product type="shared" shared="true" version="5.0.0.0" path="shared" title="Shared Product" id="N86"> <artifact type="propertyFile" resourceId="shared.properties" title="Properties" id="N87"/> <artifact type="optionList" resourceId="codeListRef.xml" description="Shared Option Lists" title="Shared Option Lists" id="N88"/> <artifact type="optionList" resourceId="acordCodeListRef.xml" description="ACORD Code Lists" title="ACORD Code Lists" id="N89"/> <artifact type="optionList" resourceId="persLinesCodeListRef.xml" description="Personal Lines Shared Code Lists" title="Personal Lines Shared Code Lists" id="N90"/> <artifact type="view" resourceId="commonViews.xml" title="Views" id="N91"/> <artifact type="pdfLayout" resourceId="POLICY_CHANGE_LAYOUT.xml" title="Policy Change PDF layout" id="N92"/> <artifact type="pdf" resourceId="Acord_Overflow.pdf" title="ACORD_OVERFLOW" description="Overflow PDF" id="N93"/> <artifact type="pdf" resourceId="PolicyChangeRequest.pdf" title="POLICY_CHANGE_FORM" description="Policy Change PDF" id="N94"/> <artifact type="pdf" resourceId="PolicyChangeRequest_Questions.pdf" title="POLICY_CHANGE_QUESTION_FORM" description="Policy Change Questions PDF" id="N95"/> <artifact type="pdf" resourceId="PolicyChangeRequest_Remarks.pdf" title="POLICY_CHANGE_REMARK_FORM" description="Policy Change Remarks PDF" id="N96"/> <artifact type="dynamicListTemplate" resourceId="dynamicListTemplates.xml" description="Shared Dynamic Lists" title="Shared Dynamic Lists" id="N97"/> <artifact type="workItemAssistant" resourceId="workitemassistant.xml" title="Shared work item assistant" description="Shared work item assistant" id="N98"/> <artifact resourceId="shared_transformers.xml" type="transformer" description="" title="ACORD Standard Transformations" id="N99"/> <artifact type="workListDefinition" resourceId="worklist.xml" title="Work Lists" description="Standard Work Lists" id="N100"/> <artifact type="workListDefinition" resourceId="accountWorklist.xml" title="Account Work Lists" description="Account Work Lists" id="N102"/> <artifact type="workItemStatusDefinition" resourceId="workItemStatus.xml" title="Work Item Status Codes" description="Work Item Status Codes" id="N521"/> <artifact type="menuDefinition" resourceId="menu.xml" title="Main Menus" description="Main Menus" id="N103"/> <artifact type="workItemActionDefinition" resourceId="workItemAction.xml" title="Work Item Actions" description="Standard Work Item Actions" id="N520" /></product><product type="commercial" title="Commercial Shared" shared="true" version="5.0.0.0" path="commercial" description="Commercial Lines Product" id="N104"> <artifact type="propertyFile" resourceId="commercial.properties" title="Properties" id="N105"/> <artifact type="optionList" resourceId="appInfoCodeListRef.xml" title="Option Lists" id="N106"/> <artifact type="optionList" resourceId="commPropCodeListRef.xml" description="Commercial Property Option Lists" title="Commercial Property Option Lists" id="N323"/> <artifact type="dynamicListTemplate" resourceId="dynamicListTemplates.xml" description="Dynamic Lists" title="Dynamic Lists" id="N107"/> <artifact type="pageLibrary" resourceId="125Commons.xml" title="125 Commons Page Library" id="N108"/> <artifact type="pageLibrary" resourceId="Comm1Commons.xml" title="Commercial Commons Page Library" id="N109"/> <artifact type="pageLibrary" resourceId="endorse125Commons.xml" title="125 Commons Page Library" id="N110"/> <artifact type="pageLibrary" resourceId="premisesInformationCommons.xml" title="Premises Commons Page Library" id="N111"/> <artifact type="view"
```

```

resourceId="appIInformationViews.xml" title="Applicant Info Views" id="N112"/> <artifact type="view"
resourceId="otherOrPriorPolicyViews.xml" title="Other Prior Policy Views" id="N113"/> <artifact type="view"
resourceId="commonCommViews.xml" title="Common Commercial Views" /> <artifact type="pdfLayout"
resourceId="ACORD_125_LA YOUT.xml" title="Acord 125 PDF layout" id="N114"/> <artifact type="pdf" resourceId="Acord125.pdf"
title="ACORD_125_FORM" description="Acord 125 PDF" id="N115"/> <artifact type="pdfLayout"
resourceId="ACORD_139_LA YOUT.xml" title="Acord 139 PDF layout" id="N116"/> <artifact type="pdf" resourceId="Acord139.pdf"
title="ACORD_139_FORM" description="Acord 139 PDF" id="N117"/> <dependency version="5.0.0.0" productRefType="shared"
id="N118"/> </product> <product type="MROF" description="" title="Management Liability" shared="false" version="5.0.0.0"
path="managementLiability" id="N504"> <artifact type="tdf" resourceId="managementLiability.xml" description="" title="management
Liability" id="N505"/> <artifact type="tdf" resourceId="endorseManageLiability.xml" description="" title="endorse management Liability"
id="N506"/> <artifact type="tdf" resourceId="renewalManageLiability.xml" description="" title="renewal management Liability" id="N507"/>
<artifact type="propertyFile" resourceId="mrof.properties" title="Properties" id="N508"/> <artifact type="acordWorkItemMap"
resourceId="managementLiabilityMap.xml" description="Acord to work item mapping for Liability Carve-Out" title="Acord to work item
mapping for Liability Carve-Out" id="N509"/> <artifact type="behavior" resourceId="miscellaneousManagementLiabilityBehaviors.xml"
description="" title="Misc management Liability Behaviors" id="N510"/> <artifact type="behavior"
resourceId="managementLiabilityBehaviors.xml" description="" title="Misc management Liability Behaviors" id="N511"/> <artifact
type="behavior" resourceId="supplementalExecRecruitingBehavior.xml" description="" title="Supplemental Exec Recruiting Behaviors"
id="N512"/> <artifact type="pageLibrary" resourceId="miscellaneousManagementLiabilityLibrary.xml" title="Miscellaneous management
Liability Library" id="N513"/> <artifact type="pageLibrary" resourceId="managementLiabilityLibrary.xml" title="management Liability
Library" id="N514"/> <artifact type="dynamicListTemplate" resourceId="managementLiabilityDynamicList.xml" description="management
Liability Dynamic Lists" title="management Liability Dynamic Lists" id="N515"/> <artifact type="optionList"
resourceId="managementLiabilityCodeListRef.xml" description="" title="Liability Codelist" id="N516"/> <artifact type="view"
resourceId="managementLiabilityViews.xml" description="Views for Liability Product" id="N517"/> <artifact
resourceId="mrof_transformers.xml" type="transformer" description="" title="Management Liability Transformations" id="N522"/>
<dependency version="5.0.0.0" productRefType="shared" id="N518"/> <dependency version="5.0.0.0" productRefType="commercial"
id="N519"/> </product> </productDatabase>
```

#### Sharable Products

Any product in the product database can be marked as shared. This allows other products to declare dependency on the shared products and hence will be included.

#### Example

In the above example, the “ACCOUNT” product depends on the “Shared” and “Commercial” products.

#### Product Type Filter

A product type filter concept is introduced in AgencyPortal5. This allows application teams to define a semicolon separated list of products that AgencyPortal should load at startup. If defined, only the product types defined are loaded and other products, even though defined in the productdatabase.xml, are ignored. This is one way development team can control which products are released.

#### Example

```
application.product_types_filter=HOME;WORK;AUTOP;AUTOB;
```

#### Import Products

The ImportManager class supports the import from an LOB archive zip file or zip input stream over an existing AgencyPortal based application. It collaborates with the product definitions manager, which negotiates the updates to the product database for resources identified by the systemas product resources. It also writes to the application instance file systemfor resources from the zip file, wh ich is not recognized by the product definitions manager directly to the file systemthat an application instance is based. An audit trail is provided. The current thread or operation must have read/write/delete/create permission/authority over the file system to complete correctly. Synchronization of clustered configurations is not supported.

# Creating a Product Database

The stated prerequisite of LOB isolation is required prior to the creation of a product database. This artifact is located at "\web\WEB-INF\productDatabase.xml" in your implementation project.

Under each LOB isolated folder, you must have a directory structure resembling the following:

```
workerscomp
├─acordtoworkitemmap
├─behaviors
├─codelists
├─definitions
├─pageLibrary
├─pdf
├─pdf_layouts
├─properties
└─views
└─xarcrules
```

## Example

For "\web\WEB\INF\workerscomp\", the above directory lives under \workerscomp.

The workersComp LOB product is definition is defined by the following:

```
<product type="WORK" version="4.0.0.0" path="workerscomp" title="Workers Compensation" description="Workers Compensation"
id="N27"> <artifact type="propertyFile" resourceId="workerscomp.properties" title="Properties" id="N75" /> <artifact type="behavior"
resourceId="workersCompBehavior.xml" title="Dynamic Behaviors" id="N28" /> <artifact type="optionList"
resourceId="workersCompCodeListRef.xml" title="Option Lists" id="N29" /> <artifact type="connector"
resourceId="workerscompConnectorConfigFile.xml" title="Integration points to external services" id="N30" /> <artifact type="tdf"
resourceId="WorkersComp_Endorsement.xml" title="Endorsement Application" id="N31" /> <artifact type="tdf"
resourceId="WorkersComp_NewBusiness.xml" title="New Business Application" id="N32" /> <artifact type="tdf"
resourceId="WorkersComp_QuickQuote.xml" title="Quick Quote Application" id="N33" /> <artifact type="pageLibrary"
resourceId="wcCommons.xml" title="WC Common Page Library" id="N34" /> <artifact type="view" resourceId="workersCompViews.xml"
title="Views" id="N35" /> <artifact type="pdfLayout" resourceId="ACORD_130_LAYOUT.xml" description="Acord 130 PDF layout"
id="N36" /> <artifact type="pdf" resourceId="Acord130.pdf" title="ACORD_130_FORM" description="Acord 130 PDF" id="N76" /> <artifact
type="sfh" resourceId="com.agencyport.workerscomp.custom.PolicyInformationHelper" title="PolicyInformation SFH" id="N77" /> <artifact
type="sfh" resourceId="com.agencyport.workerscomp.custom.RatingClassificationHelper" title="RatingClassification SFH" id="N78" />
<artifact type="acordWorkItemMap" resourceId="workerscomp.xml" title="Acord to work item mapping for workers compensation"
description="Acord to work item mapping for workers compensation" id="N79" /> <dependency productRefType="shared" id="N80" />
<dependency productRefType="commercial" id="N81" /> </product>
```

The "path" at the <product> designates the root of the "workerscomp" LOB. Artifact types () designate a component of the transaction definition environment. Each ID must be unique.

Within the product definition for "workerscomp," notice that two other products are referenced: "shared" and "commercial."

Include special field helpers by designating an artifact with `type="sfh"` and include the package path as the `resourceId`.

Each LOB that is part of your project will follow the same pattern.

# Product Definition Memory Conservation

Prior to AgencyPortal5, all product definitions were held in memory for the entire scope of the application. AgencyPortal 5 leverages the same MRU caching/serialization mechanism that was introduced for optimizing XML schemas for selected product definitions. The folder location for the serialized product definition files defaults to <application context>/WEB-INF/cache, which is fine for testing and production environments. Certain development environments under Eclipse assume that the application server runtime context is a carbon copy of the source project folder structure in Eclipse and may delete the <application context>/WEB-INF/cache folder, dynamically causing runtime issues during development.

It is recommended that you configure both your production definition cache and XML schema cache directories outside of the application context. This can be configured using application properties. The product definition cache folder's name and location can be altered by using a `cache_directory` application property while the XML schema cache folder's name or location can be altered by using an application property called `schema_cache_directory`.

## Note

Both the product definition and XML schema cache folders are rebuilt at application bootstrap time. The contents of either folder are not carried from one application session to the next.

## Example

The following is an example of overriding the product definition and XML schema cache folders:

```
output_dir=/runtime/AgencyPortal/
cache_directory=${output_dir}cache
schema_cache_directory=${output_dir}schema_cache
```

# Field Level Messaging

When a form is uploaded from Turnstile (or an agency management system), AgencyPortal applies logic at the field level to maximize the accuracy of the uploaded data. When an issue or a change from the original value is detected, specific field level messaging and specific icons can display to grab the user's attention. The messaging centered around this field level upload functionality is customizable.

As a reference, you will need to modify the `core_prompts.properties` resource bundle if you want to make any changes to this feature's out of the box functionality (refer to the [Field Level Upload Messages](#) topic in the User Workflow section for details on how the out of the box functionality of this feature works).

## Simple vs Detailed Messaging

You can change messages for two different levels of messaging: simple and detailed. Toggling between the two can be done by changing the `action.UploadMessage.SubcategoriesEnabled` property to true (detailed) or false (simple).

### Simple Messaging

Simple messaging takes into account the following basic scenarios:

- Upload message with attention/action required from the user. This message can be customized by using the `action.UploadMessage.AttentionRequired` property.
- Upload message without attention/action required from the user. This message can be customized by using the `action.UploadMessage.AttentionNotRequired` property.
- Upload message with revert to previous 'value' available. This message can be customized by using the `action.UploadMessage.RevertMessage` property.

### Detailed Messaging

After the `action.UploadMessage.SubcategoriesEnabled` property is set to true, more detailed messages are enabled. All detailed scenarios simply include additional information and more detailed versions of the simple messages described above. Detailed messaging scenarios are:

- Revert (same as simple messaging)
- Field was corrected
- Field was corrected and is a format mask (e.g., auto-truncated to a valid value)
- Field is a format mask and value upload is invalid
- Field is a select list and is required; value uploaded did not match the available options
- Field is a select list and is not required; value uploaded did not match the available options
- Field had invalid data uploaded (and is not a select list or format mask)

## Inserting the Value into a Message

When a popover message dialog is populated by the `core_prompts.properties` resource bundle, you can put the original value into any custom message. The string ' `${0}`' will be replaced by the original value at display time.

## Changing the Icon

If you want to change the icon for any scenario, change the scenario's associated `.icon` property.

### Example

You want to change the corrected icon to be the  icon. To do this, set the `.icon` property to the following:

```
action.UploadMessage.Corrected.icon=fa-ban
```

## Tooltips

Tooltips are disabled by default. You can enable tooltips by setting the following property in the resource bundle:

```
action.UploadMessage.tooltipsEnabled=true
```

You can configure tooltips on a per-scenario basis by adding `.tooltip` to the end of the scenario property in the resource bundle.

### **Example**

```
action.UploadMessage.AttentionNotRequired.tooltip=There Was a Problem Uploading this Field.
action.UploadMessage.AttentionRequired.tooltip=There Was a Bigger Problem Uploading this Field.
You need to fix this before you can continue.
```

### **Note**

We do not recommend that you enable tooltips since they cannot be viewed by touch-screen devices (they are triggered by on-hover).

# Work Items

Work items are a single, unique submission and are the primary unit of work in AgencyPortal. A work item can be typically thought of as an individual insurance application. Many work items may exist for a given customer if they have multiple active quotes or have applications in progress for a variety of lines of business. Examples of work items include an individual quick quote, new business or endorsement submission.

This section includes detailed information on how to set up, configure and work with the various work item functionality within the AgencyPortal application.

# Work Item Management

The work item management infrastructure is supported by two basic interfaces:

- `com.agencyport.workitem.model.IWorkItem` - models the basic contents of a work item
- `com.agencyport.workitem.model.IWorkItemManager` - models the mechanism for reading and writing the work item to the database

The application properties used to identify the implementations to the factory are `workitem.classname` and `workitem.manager_classname`. The product's implementations of those interfaces are `com.agencyport.workitem.impl.WorkItem` and `com.agencyport.workitem.impl.WorkItemManager`. These are engaged by the default if the properties are not explicitly configured otherwise.

The rules and guidelines for extension by the application depend heavily on whether the application is migrating from an older version of AgencyPortal with a desire to retain work item information or is a brand new application.

# ACORD to Work Item Mapping

The product's data mapping engine automatically shreds data entities in the ACORD XML document over to the work item instance. To engage this feature, a transaction definition file (TDF) refers to a data map that contains the rules on which fields to copy from the XML to the work item instance. The relationship between the TDF and the data map is made via an attribute called `mappingId`.

```
<transaction title="Workers Compensation" id="workersComp" target="WorkCompPolicyQuoteInqRq" lob="WORK"
summaryPageId="policySummary" mappingId="workersComp" prevalidate="true" >
```

The mapping ID refers to an XML file with a structure documented in the XML schema (refer to [acordtoworkitemmap.xsd](#)).

## Example

The following is an illustrated example of a mapping file:

```
<?xml version="1.0" encoding="UTF-8"?> <!-- id is the transaction.@mappingId --> <acordtoworkitemmap
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ns1="http://reference.agencyport.com/schemas/4.0/acordtoworkitemmap/acordtoworkitemmap.xsd"
id="personalAuto"><mapEntry> <xpath viewId="commonViews.xml:PersonalLineConcatenateEntityName" valueType="string" />
<workitemproperty>ENTITY_NAME</workitemproperty> </mapEntry> <mapEntry> <xpath
valueType="date">PersPolicy.ContractTerm.EffectiveDt</xpath> <workitemproperty>EFFECTIVE_DATE</workitemproperty> </mapEntry>
<mapEntry> <xpath valueType="date">PersPolicy.ContractTerm.ExpirationDt</xpath>
<workitemproperty>EXPIRATION_DATE</workitemproperty> </mapEntry> <mapEntry> <xpath
valueType="string">PersPolicy.LOBCode</xpath> <workitemproperty>LOB</workitemproperty> </mapEntry> </acordtoworkitemmap>
```

Any time one of the values configured in the mapping file changes in the ACORD XML document changes, it automatically synchronizes with that corresponding column in the work item by using the `IWorkItem.setPropertyValue()` method.

# Work Item Management Example

New applications should start with a clean slate regarding the work item data model and align with the single work item table model. It is customary to add one or more columns to the work item for the purposes of enhancing an application's work list and/or workflow. The example shown on this page adds two columns to the work item table and illustrates how you can leverage existing core functionality without replicating implementation.

## Extending the DDL

In this example, two columns are added to the work item: a work item description field and a `copy_source_id` field, which tracks the work item ID from the current work item cloned in a copy work item scenario.

### Best Practice

Extend the `work_item` table's DDL shipped with the product (if necessary) to meet the needs of the application.

Append the new custom columns to the end of the existing product's version of the DDL. Do NOT insert columns.

Extend the DDL for the work item table as follows:

```
CREATE TABLE work_item (work_item_id integer NOT NULL, user_group_id integer NOT NULL, creator_id integer NOT NULL, owner_id integer NOT NULL, owner_group_id integer NOT NULL, status integer NOT NULL, policy_number varchar(25), LOB varchar(10) NOT NULL, effective_date ${db_current_date}, expiration_date ${db_current_date}, creation_time ${db_current_time}, entity_name varchar(80), premium decimal(13,2), transaction_id varchar(25) NOT NULL, upload_flag smallint NOT NULL, last_update_by varchar(${login_id_size}) NOT NULL, last_update_time ${db_current_time} NOT NULL, commit_flag smallint NOT NULL, /* Append custom columns to end per best practice */description varchar(128), copy_source_id integer);
```

## Extending the WorkItem Class

### Best Practice

Extend the concrete `com.agencyport.workitem.impl.WorkItem` class by adding set/get methods for the fields not currently modeled by the base class. Leverage the property map internal to the base class by use of the `setProperty`/`getProperty` by name methods. Using the `setProperty` method manages the dirty flag state of the work item correctly requiring no special code.

```
/* * Created on May 21, 2009 by nbaker AgencyPort Insurance Services, Inc. */ package test.agencyport.workitem.impl; import com.agencyport.id.Id; import com.agencyport.workitem.impl.WorkItem; /** * The CustomWorkItem class illustrates how to extend the product's work item class * to add custom properties. other properties */ public class CustomWorkItem extends WorkItem { /** * The <code>COPY_SOURCE_ID</code> is the key for the copy source id data value. */ public static final String COPY_SOURCE_ID = "COPY_SOURCE_ID"; /** * The <code>DESCRIPTION</code> is the key for the description of the work item. */ public static final String DESCRIPTION = "DESCRIPTION"; /** * Return the source from which this work item was originally based on. */ @return the source from which this work item was originally based on. */ public Id getCopySourceId(){ return (Id) getProperty(COPY_SOURCE_ID); } /** * Sets the work item id for the work item this work item was copied from. * @param copySourceId is the work item id of the work item this one was copied from. */ public void setCopySourceId(Id copySourceId){ setProperty(COPY_SOURCE_ID, copySourceId); } /** * Returns the description this work item. * @return the description this work item. */ public String getDescription(){ return (String) getProperty(DESCRIPTION); } /** * Sets the description of this work item. * @param description is the description of the work item to set. */ public void setDescription(String description){ setProperty(DESCRIPTION, description); } }
```

## Extending the WorkItemManager Class

### Best Practice

Extend the concrete `com.agencyport.workitem.impl.WorkItemManager` class and only override the following specific methods:

- `getAdditionalCustomColumnNameList()`
- `customInitializeWorkItem()`
- `customBindDataToStatement()`
- `customMapResultsSetToWorkItem()`

## Tip

In 4.5, there was a change to some of the method signatures that render the following code samples, making them slightly out of date. If your method signatures are out of date, substitute `IWorkItem` for `ILOBWorkItem`.

### Example

```
customBindDataToStatement(ILOBWorkItem workItem, PreparedStatement pStmt, int
ordinalOfNextBoundParameter)

/* * Created on May 21, 2009 by nbaker AgencyPort Insurance Services, Inc. */ package test.agencyport.workitem.impl; import
java.sql.PreparedStatement; import java.sql.ResultSet; import java.sql.SQLException; import java.util.Map; import
com.agencyport.domXML.APDataCollection; import com.agencyport.id.Id; import com.agencyport.security.profile.ISecurityProfile; import
com.agencyport.trandef.Transaction; import com.agencyport.workitem.impl.WorkItemException; import
com.agencyport.workitem.impl.WorkItemManager; import com.agencyport.workitem.model.IWorkItem; import
com.agencyport.workitem.model.IWorkItemStatusManager; /*** The CustomWorkItemManager class illustrates how the product's work item
manager class can be * extended to support the scenario where the application needs to add one or more columns to the work_item table with
additional * columns appended therein. The restrictions are as follows: * All of the columns in the physical work_item table must match the
current product's version of the DDL up to the last column which it defines.. * The application can only append new columns at the end
and not insert new columns. * If the application have additional tables beyond the work_item table which comprises the notion * of a
work item then this technique will not work. */ public class CustomWorkItemManager extends WorkItemManager { private static final
String description = "description"; private static final String copy_source_id = "copy_source_id"; /*** The
<code><nonKeyColumnNameList</code> contains all of the non primary * key column names in the work item table. */ private static final
String[] additionalNonKeyColumnNameList = { description, copy_source_id }; /*(non-Javadoc) * @see
com.agencyport.workitem.impl.WorkItemManager#addAdditionalCustomColumnNameList() */ @Override protected String[]
getAdditionalCustomColumnNameList() { return additionalNonKeyColumnNameList; } /*(non-Javadoc) * @see
com.agencyport.workitem.impl.WorkItemManager#customInitializeWorkItem(com.agencyport.workitem.model.IWorkItem,
com.agencyport.domXML.APDataCollection, com.agencyport.trandef.Transaction, com.agencyport.id.Id, com.agencyport.id.Id,
java.lang.Boolean, java.lang.Boolean, com.agencyport.security.profile.ISecurityProfile, java.util.Map) */ @Override protected void
customInitializeWorkItem(IWorkItem newWorkItem, APDataCollection apdata, Transaction transaction, Id userId, Id userGroupId, Boolean
isUpload, Boolean commitFlag, ISecurityProfile securityProfile, Map<?, ?> customAttributes) throws WorkItemException { if (
customAttributes == null) return; CustomWorkItem customWorkItem = (CustomWorkItem) newWorkItem;
customWorkItem.setCopySourceId((Id) customAttributes.get(CustomWorkItem.COPY_SOURCE_ID));
customWorkItem.setDescription((String) customAttributes.get(CustomWorkItem.DESCRIPTION)); } /*(non-Javadoc) * @see
com.agencyport.workitem.impl.WorkItemManager#customBindDataToStatement(com.agencyport.workitem.model.IWorkItem,
java.sql.PreparedStatement, int) */ @Override protected void customBindDataToStatement(IWorkItem workItem, PreparedStatement pStmt, int
ordinalOfNextBoundParameter) throws SQLException { pStmt.setString(ordinalOfNextBoundParameter,
((CustomWorkItem)workItem).getDescription()); pStmt.setInt(ordinalOfNextBoundParameter + 1,
((CustomWorkItem)workItem).getCopySourceId().intValue()); } /*(non-Javadoc) * @see
com.agencyport.workitem.impl.WorkItemManager#customMapResultSetToWorkItem(com.agencyport.workitem.model.IWorkItem,
java.sql.ResultSet, com.agencyport.workitem.model.IWorkItemStatusManager) */ @Override protected void
customMapResultSetToWorkItem(IWorkItem workItem, ResultSet resultSet, IWorkItemStatusManager workItemStatusManager) throws
WorkItemException, SQLException { ((CustomWorkItem)workItem).setDescription(resultSet.getString(description));
((CustomWorkItem)workItem).setCopySourceId(new Id(resultSet.getInt(copy_source_id))); } }
```

# Work Item Status Management

The model for work item status management class instantiation is similar to ACSI since a factory design pattern is adopted. Applications should no longer have to engage the class hijacking technique that was used in previous version of AgencyPortal. The work item status management infrastructure is supported by two basic interfaces:

- `com.agencyport.workItem.model.IWorkItemStatus` - models the basic contents of a work item status
- `com.agencyport.workItem.model.IWorkItemStatusManager` - models the mechanisms for retrieving work item status instances

The application properties used to identify the implementations to the factory are `workitemstatus.classname` and `workitemstatus.manager_classname`. The product's implementations of those interfaces are `com.agencyport.workitem.impl.WorkItemStatus` and `com.agencyport.workitem.impl.WorkItemStatusManager`. They are engaged by the factory if the properties are not explicitly configured otherwise.

The AgencyPortal implementation of the work item status and manager interfaces uses a simple database table as the repository for status code values and their associated mnemonic abbreviation and various title states. The following is the data model for that table:

Entity	Description	Data Type
status code value	Numeric value - this is the value that is carried on the work item table	Integer
instruction mnemonic	Mnemonic used in declarative instructions as in <code>page.connectors.instruction[@type='statusChange'].@value</code>	String
Before Change Status Title	User friendly eye catcher describing the status before it is applied to a work item	String
During Change Status Title	User friendly eye catcher describing the status during the time when it is applied to a work item	String
After Change Status Title	User friendly eye catcher describing the status after it is applied to a work item	String

The following is the inventory of status codes that accompany the product:

15	INPROGRESS	In Progress	In Progress	In Progress
20	REFER	Refer	referring	Referred
21	APPROVE	Approve	approving	Approved
22	BIND	Bind	binding	Bound
30	REJECT	Reject	rejecting	Rejected
35	DECLINE	Decline	declining	Declined
40	OFFLINE	Offline	offline	Offline
55	DELETE	Delete	deleting	Deleted

## Best Practice

Applications are urged to use the above table structure for their own status values and associated mnemonics and titles. Applications that choose to adopt this database table approach can expect to leverage the work item status infrastructure to its fullest utilization. This is considered a superior technique to implementing the `IWorkItemStatusManager` interface.

## Best Practice

Don't evaluate status codes by using a hard coded value comparison.

### Example

Replace this technique:

```
void dontDoThis(IWorkItem status) { if (status.getStatusCodeValue() == 15) { //Is in progress } }
```

With this approach:

```
void butInsteadDoThis(IWorkItem status) { if (IWorkStatus.IN_PROGRESS.equals(status.getMnemonic())) { // Is in progress } }
```

## Best Practice

Put all custom work item status mnemonic constant values in an extended class of the AgencyPortal's work item status class.

### Example

```
public class CustomWorkItemStatus extends WorkItemStatus { public static final String MY_SPECIAL_WORK_ITEM_STATUS = "MY_SPECIAL_WORK_ITEM_STATUS"; }
```

### Tip

#### Requirement:

When using the `statusChange` instruction workflow feature, you must use the status mnemonic and NOT the status code value as illustrated in the following:

```
<instruction id="II" type="statusChange" value="REFER"> <when connector="generalInfo_Page" /> </instruction>
```

Where the following is recorded in the status code database table:

```
20 REFER Refer referring Referred
```

# Work Item Record Locking

AgencyPortal comes with built-in work item record locking. This means only one user can edit a work item at a time. As long as one user is editing a work item, the work item is read-only for all other users.

The `client_update_interval` application property controls how often the client sends periodic updates to the back end to check work item locking. This property defaults to 15 seconds. The minimum value allowed is 15 seconds.

A user may stop working on a work item for awhile and leave the session idle. In this case, to reduce the network traffic, and to release connection resources and work item locking, the period update is terminated. The `client_update_timeout` property controls the length of time the client can be idle before the periodic updates are stopped. The default is 600 seconds.

The `work_item_record_lock_time_in_seconds` property is used to determine how long a resource is locked:

Property	Description	Default Value	Recommended Setting (Optional)
<code>work_item_record_lock_time_in_seconds</code>	Number of seconds a work item is locked after a user is no longer making changes to a work item. If a user closes their browser window while in a work item, the work item is locked from the last update, plus number of seconds configured using this property. The default value is 600 (10 minutes) and blank; 0 (zero) or a negative number turns work item locking off, which is not recommended.	600	

## Note

If the user closes their browser without logging out first, they will lock out all other users for the work item up to the configured lock time. However, if the user's session times out due to inactivity, the record lock on the work item is released to allow another user to access and maintain the work item. Refer to the Session Timeout section in Miscellaneous Properties for more information.

# Autosave

Data entered on a work item transaction page is automatically saved at a configurable time interval (the default is 60 seconds). This time interval is configurable via the `auto_save_interval` property in the `framework.properties` file. This property defaults to 60, which indicates the autosave feature is on and set to 60 seconds. If you don't want to use the autosave feature, set the `auto_save_interval` property to 0.

Property	Description	Default Value	Recommended Setting (if any)
<code>auto_save_interval</code>	The amount of time in seconds between checks to autosave data. 0 or less turns auto saving off. The value defaults to 60 if no property is provided.	60	

When data is automatically saved, the user is notified and the time and date displays on the page. If a user leaves the browser open and becomes inactive, which causes the browser to time out, they are brought back to the page where the data was automatically saved the next time they access the work item. Refer to the [Autosave](#) topic in the User Workflow section for more information on how this feature works.

Autosaved data is stored and processed differently than other work item data. When a user continues off of a page or adds/edits a roster item, a formal commit process is engaged (including data validations, business rules, workflow operations and persistence to the `APDataCollection's xmlstore` table) for most work item data. However, these commit processes do not take place for autosave; instead, the field's current values are taken from the page verbatim and stored in the database separately from the `APDataCollection's xmlstore`. No validation takes place on autosaved data since autosaved data is transitory (i.e., the user hasn't engaged a formal commit to the data since they haven't clicked the Continue, Add or Save button).

## Note

Encrypted fields that are configured to contain sensitive data, such as tax ID, are not autosaved in the `auto_save_store` table. The autosave feature does not autosave encrypted field data. Encrypted field data is only saved when progressing through the regular transaction workflow.

A user can also use the Save or Save and Exit option to save data without passing validations. This is configurable in the `user_auto_save_on_save` property in the `framework.properties` file. This property defaults to `false`. If this feature is turned on (property is set to `true`), a user is brought back directly to the page where they last left off when they access the work item again. Since the data entries are not validated, the data is stored temporarily until the user manually saves the data. The data is then saved in the XML when the user saves it.

# Enforcing Account Creation

AgencyPortal allows users to create work items without having to create an account first, but there are times when a carrier requires an account to be created to comply with a back end system. We provide the ability to enforce account assignment at any point within the application workflow.

When a work item is created without first creating an account, a rule or validation connector can be used to stop the user from proceeding further at an appropriate stage within the transaction workflow. The following is an example of a simple validation connector. When this connector is used on any page, the user is stopped from proceeding beyond that page without first linking the work item to an account:

```
<connector type="custom" className="com.agencyport.shared.commercial.connectors.AccountValidationManager"
id="AccountValidationManager"> <executeWhen userAction="Continue"/> </connector>
```

# Percent Complete

The Percent Complete feature provides users with information on the progress of a work item. This information displays in the form of a percent on the work list and within the work item transaction workflow. By default, the algorithm of this percentage is measured by the number of pages completed (user has manually saved the page) out of the number of pages within the transaction. Each page is weighted identically, even if some pages have more fields than others. However, read-only pages, such as summary pages, and autosaved pages are not included in the calculation.

## Note

When a work item is uploaded into the application, it goes through a series of transaction validations. The percent complete in this scenario is determined by the number of pages visited by the total page count. The work item is considered complete if no validation errors are detected.

## Customizing the Calculation

If you do not want to include a page within the percent complete calculation, set the <LOBCODE>.progress.page.ignored property as a semicolon-separated list.

### Example

```
AUTOP.progress.page.ignored=GaragingLocationsPage;VehiclesPage
```

In this example, the Garaging Locations and Vehicles pages are excluded from the calculation for Personal Auto transactions.

Customizing the calculation is usually used to exclude certain pages from the calculation during testing or to exclude read-only pages.

## Overriding the Default Calculation

Alternatively, you can also override the default calculation entirely by creating a new custom class and setting a property to handle the calculation. This class must implement the IProgressBarUtil interface, which currently has getTransactionProgress() as a single method.

### Example

```
progress_bar_util=com.agencyport.shared.utils.MyCustomProgressBarUtil
```

This example tells the system to use the getTransactionProgress() method in MyCustomProgressBarUtil to calculate the percentage complete.

## Disabling the Progress Bar

The percent complete calculation displays as a progress bar on each individual transaction page. You can turn off the progress bar by setting the show\_progress\_bar property to false.

When you restart the application, the progress bar no longer displays on transaction pages.

## Removing the Calculated Percent

The percent complete calculation displays on both the card and classic views of the work list. The information and columns that display within the work list is customizable. Refer to the [Work List](#) topic for more information on how to remove a column from the work list.

# Improved Access to Work Item/Account Data

AgencyPortal 5 introduced a new mechanism for getting access to the current `IWorkItem` instance and its associated `APDataCollection` instance, as well as to some other key entities. This mechanism eliminates redundant fetches of the work item in flight from the database by the framework and the application. The same thread local storage technique that several other facets of the framework uses is leveraged, which makes the work item easily accessible anywhere without the need to pass explicit references everywhere. In most cases, the framework will have loaded the work item in flight by the time any application logic is engaged.

The following items are made available by the `WorkItemContext` class:

1. `IWorkItem` instance for the work item that is in flight.
2. `Transaction` instance for the transaction that is in flight.
3. `APDataCollection` instance for the work item XML that is in flight.
4. `HTMLDataContainer` instance for the HTML data container that is in flight. This is only available on the process side during the time the framework data engine is active [while `CollectionProcess.updateCollection()` is running]. In fact, this facility allows custom view (implementations of `IView`) logic to gain access to current request data.

The `WorkItemContextStore` class is the framework class that manages work item context on thread local storage. All public methods are static; the following are salient:

<code>isPresent</code>	Determines whether a work item context has been associated with the current thread.
<code>get</code>	Returns the work item context instance.
<code>clearWorkItemContext</code>	Clears the work item context instance from thread.
<code>load</code>	Checks first to see if there is a work item context. If so, verifies the work item ID matches the one passed in. If either of these tests fail, the work item and the <code>APDataCollection</code> are fetched from the database and loaded into the current work item context and set on the thread.

## Example

The following is an example of its usage:

```
IWorkItem currentWorkItem = WorkItemContextStore.get().getWorkItem();

APDataCollection currentXML = currentWorkItem.getDataCollection(); OR WorkItemContextStore.get().getDataCollection();

Transaction currentTransaction = WorkItemContextStore.get().getTransaction();
```

A custom view (snipped from `OtherThanCollisionCoverageView` from the 5 AUTOB Template) using access to `HTMLDataContainer` to determine which field set to apply to the `APDataCollection` for the in flight update operation.

```
/** * {@inheritDoc} */ @Override public void update(APDataCollection apData, int[] indices, String value, String associatedElementPath){ if (associatedElementPath.contains("Deductible")){ handleDeductibleAmountUpdate(apData, indices, value); } else { super.update(apData, indices, value, associatedElementPath); } } /** * Updates the deductible amount. * @param apData is the data collection in context. * @param indices are the indices to apply to the data collection. * @param value is the value to apply to the data collection. */ private void handleDeductibleAmountUpdate(APDataCollection apData, int[] indices, String value){ if (StringUtilities.isEmptyOnTrim(value)){ return; } // deletion of parent aggregate assumed to be handled by the call into this view for the coverage code and not the delete element } // Look up the coverage code value from the HTML data container HTMLDataContainer htmlDataContainer = WorkItemContextStore.get().getHTMLDataContainer() ; if (htmlDataContainer == null){ throw new IllegalStateException("HTML data container missing from work item context store but is needed by this view."); } String htmlCoverageCodeValue = htmlDataContainer.getStringValue(htmlCoverageCodeFieldId , null); if (StringUtilities.isEmptyOnTrim(htmlCoverageCodeValue)){ return; } // no update needed } int fieldSetIndex = this.determineFieldSetFromInputValue(htmlCoverageCodeValue); deductibleFields[fieldSetIndex].update(apData, indices, value); }
```

# Account Management

Account management allows you to create an account before or after creating a work item. Refer to [Turnstile Integration](#) for more information on how accounts and work items are created using Turnstile within AgencyPortal 5.2.

## Configuring Account Management

Set the following properties in the framework.properties file to start using the account management feature:

Property	Description	Default Value	Recommended Setting (if any)
application.account_management	Boolean flag that governs whether account management is on or off at the application scope.	false	
application.account.management.support_merge	Boolean flag that governs whether account merge is supported.	none	
application.account.management.usergroup.filter	Boolean flag that governs whether accounts can only be accessed through user group permissions.	none	
account.manager_classname	Java class name that implements the com.agencyport.account.model.IAccountManager interface.	com.agencyport.account.impl.AccountManager	
application.workitem_save_exit_to_account	Boolean flag that governs whether a save and exit action on a work item with a linked account will lead the user to the account save_exit page (by default, the summary page).	false	

Set the application.account\_management\_exit\_next\_page to /DisplayWorkInProgress?WORKITEMID=\${ACCOUNTID}&action=View&WorkListType=AccountsView when using the template support for account management. When a user clicks Save and Exit, they are taken to the Account Detail view of the account associated with the work item.

## Account Management Infrastructure

The account management infrastructure is supported by two basic interfaces:

- one that models the basic contents of an account
- one that models the mechanism for reading and writing the account to the database

These interfaces are:

- com.agencyport.account.model.IAccount
- com.agencyport.workItem.model.IWorkItemManager

The application properties used to identify the implementations to the factory are:

- account.classname
- account.manager\_classname

The product's canonical implementations of these interfaces are the following:

- com.agencyport.workitem.impl.Account
- com.agencyport.account.impl.AccountManager

These are engaged by the default if the properties are not otherwise explicitly configured.

## Personal and Commercial Account Templates

Two account templates are provided:

- Personal Account
- Commercial Account

These templates provide the basic fields a user needs to fill out to set up either a personal line or commercial line account. You need at least one of these templates to get account management up and running. Contact Agencyport for access to these templates.

## Account Type Removal

If you do not need one of the account types added to your application, you can remove the reference of the account type on any account page.

Perform the following remove the personal account type:

1. Update the account TDF located in \WEB-INF\account\pagelibrary\accountPages.xml to use hidden fields instead of radio fields for the account type and to code the account type as commercial:

```
Page: accountInformation <fieldElement type="hidden" id="BroadLOBCd" label="Account Type" uniqueId="AccountHolderType" value="C" /> <!-- <fieldElement type="radio" id="BroadLOBCd" label="Account Type" uniqueId="AccountHolderType" required="true"> <optionList source="accountCodeListRef.xml" target="accountTypeCd" /></fieldElement> --> Page: People <fieldElement type="hidden" id="MiscParty.GeneralPartyInfo.NameInfo.NameTypeCd" label="Type" uniqueId="AccountContactType" value="C" /> <!-- <fieldElement type="radio" id="MiscParty.GeneralPartyInfo.NameInfo.NameTypeCd" Label="Type" uniqueId="AccountContactType" required="true"> <optionList source="accountCodeListRef.xml" target="accountTypeCd" /></fieldElement> -->
```

2. Remove all personal account related fields from the TDF:

- /accountInformation/AccountInformationSection/AccountHolderFirstName
- /accountInformation/AccountInformationSection/AccountHolderLastName
- /accountInformation/AccountInformationSection/taxIdSSN
- /accountInformation/AccountInformationSection/ApplicantHomePhoneNumber
- /accountInformation/AccountInformationSection/ApplicantWorkPhoneNumber
- /accountInformation/AccountInformationSection/ApplicantMobilePhoneNumber
- /people/ContactDetailSection/ContactFirstName
- /people/ContactDetailSection/ContactLastName
- /people/ContactDetailSection/contactSSN
- /people/ContactDetailSection/ContactHomePhoneNumber
- /people/ContactDetailSection/ContactWorkPhoneNumber
- /people/ContactDetailSection/ContactMobilePhoneNumber

3. Remove all of the exclude/include behaviors related to the account type in WEB-INF\account\behaviors\accountBehavior.xml.

4. Remove the /people/ContactDetailSection/AccountContactType hot field.

5. Remove the newId[1], newId[2], newId[3], newId[4], newId[5], newId[8], newId[9] and newId[12] behaviors.

6. Modify worklist/partials/account-addnewcard.tpl.jsp

```
<div class="col-xs-12 col-sm-6 col-md-4" id="add_new_workitem"> <div class="card add" id="add-button" ng-show="addNewState == 'none'"> <div ng-click="createWorkItem(href:addlinks[0].value + '&account_type=<%=AccountType.C.name()%>')" class="add-button"> <i class="fa"></i> <a><%= accountRB.getString("action.AddNewAccount") %> </div> </div> </div>
```

7. Remove the following from worklist/partials/tabularview.jsp:

```
<li ng-if="tblCtrl.isAccountsView()"> <a ng-href="" ng-click="tblCtrl.createWorkItem(tblCtrl.workListModel.metaData.newWorkItemLinks[0].value + '&account_type=<%=AccountType.P.name()%>') "><%= accountRB.getString("label.Account.Personal") %>
```

8. Open the My Work page in the UI.

9. Click the Add New Account option and verify that a new commercial account launches (there is no option to select from personal or commercial).

10. Remove the account type from the My Accounts work list page:

1. Remove the account\_type column in account work list queries (\shared\worklist\solrAccontWorklist.xml) so that it does not display on the classic view version of the My Account page.
2. Remove the account type so that the account type does not display on the card view version of the My Account page:
  - Remove line 18 in Web-ROOT\worklist\partials\account-card.tpl.jsp:

```
<li class="type">{lookupValue("ACCOUNT_TYPE",workitemData.account_type)}
```

- Remove line 39 in WEB-ROOT\account\accountSummary.jsp

```
<dd class="account-type"><span data-toggle="tooltip" data-placement="right"
title="<%=(IAccount.AccountType.isPersonal(details.getAccountType()))?accountRB.getString("account.management.label.person")%>"><%=(IAccount.AccountType.isPersonal(details.getAccountType()))?accountRB.getString("account.management.label.company")%>"><%=(IAccount.AccountType.isPersonal(details.getAccountType()))?accountRB.getString("account.management.label.person")?accountRB.getString("account.management.label.company")%></dd>
```

Perform the following remove the commercial account type:

1. Update the account TDF located in \WEB-INF\account\pagelibrary\accountPages.xml to use hidden fields instead of radio fields for the account type so you can code the account type as personal:

```
Page: accountInformation <fieldElement type="hidden" id="BroadLOBCd" label="Account Type" uniqueId="AccountHolderType" value="P" /> <!-- <fieldElement type="radio" id="BroadLOBCd" label="Account Type" uniqueId="AccountHolderType" required="true"> <optionList source="accountCodeListRef.xml" target="accountTypeCd" /></fieldElement> --> Page: People
<fieldElement type="hidden" id="MiscParty.GeneralPartyInfo.NameInfo.NameTypeCd" label="Type" uniqueId="AccountContactType" value="P" /> <!-- <fieldElement type="radio" id="MiscParty.GeneralPartyInfo.NameInfo.NameTypeCd" Label="Type" uniqueId="AccountContactType" required="true"> <optionList source="accountCodeListRef.xml" target="accountTypeCd" /></fieldElement> -->
```

2. Remove all commercial account related fields from the TDF.
  - /accountInformation/AccountInformationSection/AccountHolderCompanyName
  - /accountInformation/AccountInformationSection/DoingBusinessAs
  - /accountInformation/AccountInformationSection/taxIdFEIN
  - /accountInformation/AccountInformationSection/ApplicantBusinessPhoneNumber
  - /accountInformation/AccountInformationSection/legalEntity
  - /people/ContactDetailSection/ContactCompanyName
  - /people/ContactDetailSection/contactFEIN
  - /people/ContactDetailSection/ContactBusinessPhoneNumber
  - /people/RolesSection/contactRoleLaborClient
  - /people/RolesSection/contactRoleLaborContractor
3. Remove all of the exclude/include behaviors related to the account type in WEB-INF\account\behaviors\accountBehavior.xml.
4. Remove the /people/ContactDetailSection/AccountContactType hot field.
5. Remove the newId[1], newId[2], newId[3], newId[4], newId[5], newId[8], newId[9] and newId[12] behaviors.
6. Modify worklist/partials/account-addnewcard.tpl.jsp:

```
<div class="col-xs-12 col-sm-6 col-md-4" id="add_new_workitem"> <div class="card add" id="add-button" ng-show="addNewState == 'none'"> <div ng-click="createWorkItem({href:addlinks[0].value + '&account_type=<%=AccountType.P.name()%>'})" class="add-button"><i class="fa"></i> <a><%= accountRB.getString("action.AddNewAccount")%> </div> </div> </div>
```

7. Remove the following from worklist/partials/tabularview.jsp:

```
<li ng-if="tblCtrl.isAccountsView()"> <a ng-href="" ng-click="tblCtrl.createWorkItem(tblCtrl.workListModel.metaData.newWorkItemLinks[0].value + '&account_type=<%=AccountType.C.name()%>')"><%= accountRB.getString("label.Account.Commercial") %>
```

8. Open the My Work page in the UI.
9. Click the Add New Account option and verify that a new personal account launches (there is no option to select from personal or commercial).
10. Remove the account type from the My Accounts work list page:
  1. Remove the account\_type column in account work list queries (\shared\worklist\solrAccontWorklist.xml) so that it does not display on the classic view version of the My Account page.
  2. Remove the account type so that the account type does not display on the card view version of the My Account page:
    - Remove line 18 in Web-ROOT\worklist\partials\account-card.tpl.jsp:

```
<li class="type">{{lookupValue("ACCOUNT_TYPE",workitemData.account_type)}}
```
    - Remove line 39 in WEB-ROOT\account\accountSummary.jsp

```
<dd class="account-type"><span data-toggle="tooltip" data-placement="right"
title="<%=(IAccount.AccountType.isPersonal(details.getAccountType()))?accountRB.getString("account.management.label.person")%>"><%=(IAccount.AccountType.isPersonal(details.getAccountType()))?accountRB.getString("account.management.label.company")%>"><%=(IAccount.AccountType.isPersonal(details.getAccountType()))?accountRB.getString("account.management.label.person")?accountRB.getString("account.management.label.company")%></dd>
```

## Enforcing Account Creation

AgencyPortal includes the ability to enforce account assignment at any point within the application workflow, which provides compliance for those applications that interface with a back end system that requires an account. The implementation of this feature is customizable via the [Enforcing Account Creation](#) topic in the Developer Guide section.

When account creation is enforced, a message displays to the user at the determined point within the transaction workflow. This message is based on the XARC rules established.

# Solr Configuration

Solr makes it easy to add the capability to search your application entities through the following steps:

1. Define a schema. The schema tells Solr about the contents of documents it will be indexing. Solr's schema is powerful and flexible, and allows you to tailor Solr's behavior to your application.
2. Deploy Solr as part of your application.
3. Feed Solr the document for which your users will search.
4. Expose search functionality in your application.
5. Applications interact with Solr using the RESTful queries, which means that a query is a simple HTTP request URL and the response is a structured document (mainly XML, but can also be JSON, CSV or some other format). This allows a wide variety of clients to use Solr from other web applications to browser clients, rich client applications and mobile devices. Any platform capable of HTTP can talk to Solr.

The AgencyPortal product ships with a pre-configured Solr schema and the components necessary to populate and query the Solr index. This section is intended for those looking to customize and extend the out of the box Solr implementation or those looking to simply gain a better understanding of it.

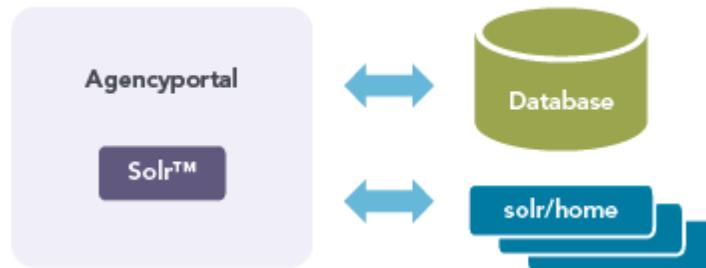
## Note

For more information regarding the deployment of Solr and the environment configuration that goes along with a deployment, refer to the [Solr Setup](#) topic in the Getting Started section.

## Solr Architecture

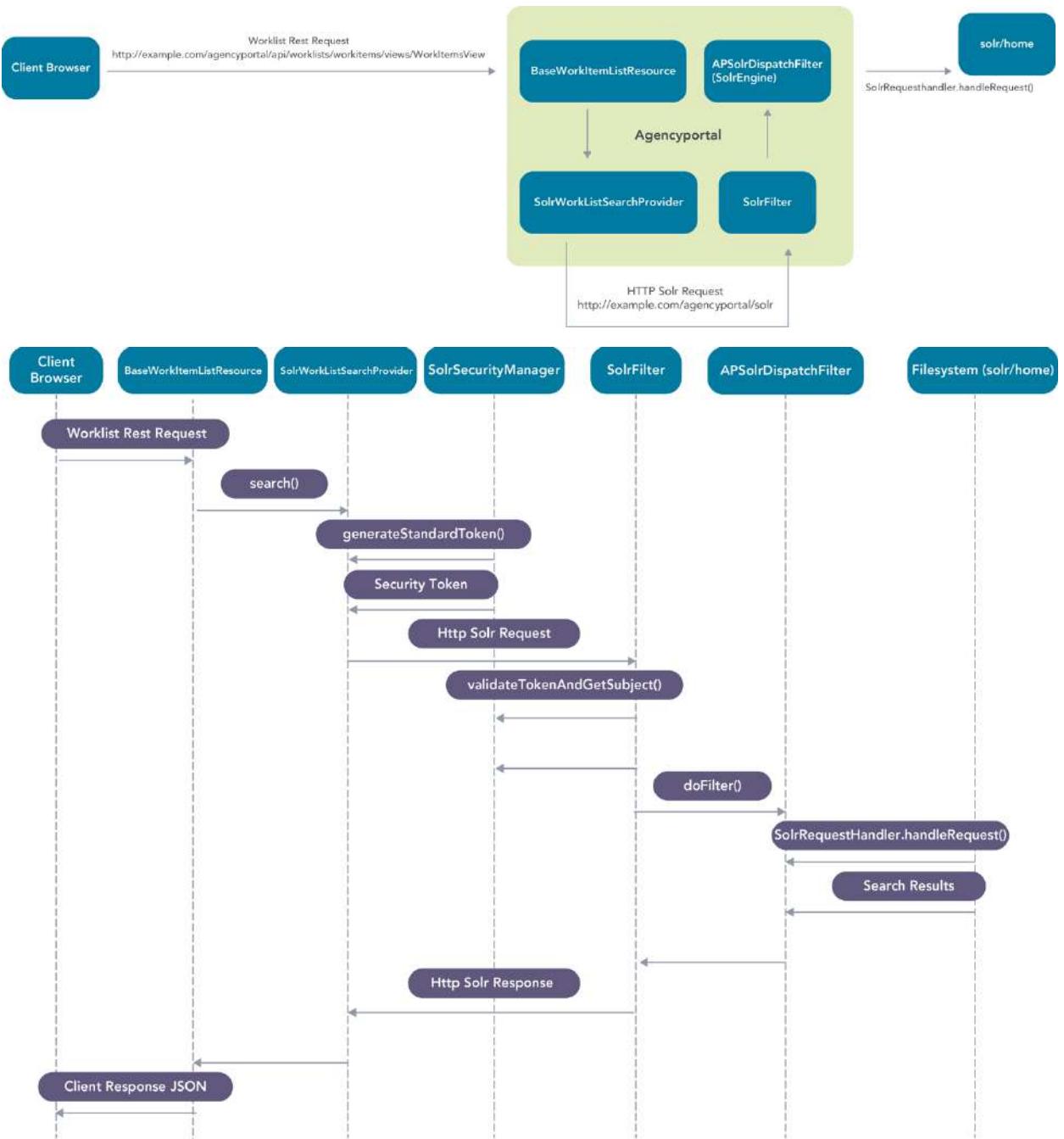
The following diagram illustrates the layered architecture and the various components that constitute each tier when using Solr for search and filter within AgencyPortal:

### Portal Runtime



As illustrated above, the Solr engine is embedded within an AgencyPortal war. The Solr runtime libraries come with the AgencyPortal SDK artifacts, and the template web.xml file also exposes an APSolrDispatchFilter (a subclass of Solr's SolrDispatchFilter), which is responsible for processing Solr queries. Solr's data index and configuration files are located outside of the deployed WAR file in "solr home" (illustrated in the diagram above as "solr/home"). The pre-configured Solr home directory structure ships with the SDK and template project artifacts under the solr/SOLR\_HOME path. Out of the box, this SOLR\_HOME directory is ready to be deployed to a server along with an AgencyPortal war file. For information on deployment, refer to the [Solr Setup](#) topic in the Getting Started section. For information on the makeup of the Solr home directory and the nature of the configuration files within it, refer to the [Solr Configuration Files](#) section below.

The following diagrams below attempt to illustrate the components of AgencyPortal at work as it handles an incoming Solr request for a client browser:



## Client Browser

Search requests into the Solr engine are typically triggered by AgencyPortal's client user interface in the form of a work list REST API call, which indicates which work list "view\_name" is required. Even though the embedded Solr engine itself exposes a REST API, this endpoint has been protected with a security token so that the client browser can never call Solr directly. As a result, all requests to Solr must originate from the server side.

## BaseWorkItemListResource

The **BaseWorkItemListResource**, implemented by either the **WorkItemListResource** or the **AccountListResource** subclass, loads the appropriate **WorkListView** object based on the "view\_name" provided by the client and utilizes the **SolrWorkListSearchProvider** to generate a search request into Solr.

## Note

The `WorkListView` object referred to above is effectively the runtime representation of a `workListViewDefinition` product definition XML artifact that has been registered to `productDatabase.xml`. Refer to the [Work List Architecture](#) and [Server Side Work List Configuration](#) topics for more information.

## SolrManager

The implementation of the `ISolrManager` interface is responsible for providing a simplified interface for interacting with the Solr server to search, index and delete. Requests to Solr from the `SolrManager` utilize the `ISolrSecurityManager` implementation to generate a security token, which will ultimately be inspected by the `SolrFilter`. For more information, refer to the [SolrManager](#) topic.

## SolrSecurityManager

The implementation of the `ISolrSecurityManager` interface is responsible for generating and subsequently validating security tokens that will be added to Solr requests as request headers. The `APSolrHttpRequestInterceptor` intercepts all Solr requests bound for Solr and uses the `SolrSecurityManager` to generate a security token, and then attaches it to the request. The `SolrFilter` will then use the `SolrSecurityManager` to validate all incoming Solr requests. Refer to the [Solr Security](#) topic for more information.

## SolrFilter

The `SolrFilter` sits in front of inbound HTTP requests that are headed for the embedded Solr engine (`SolrDispatchFilter`). Its sole responsibility is to validate the Solr security tokens on such requests; thereby, insuring that they originate from our own `SolrManager` or `SolrJ` internal clients and not a bad actor. The `SolrFilter` delegates to the `SolrSecurityManager` implementation to validate request's security token. Refer to the [Solr Security](#) topic for more information.

## APSolrDispatchFilter

The `APSolrDispatchFilter` is AgencyPortal's own subclass of Solr's "SolrDispatchFilter." The `SolrDispatchFilter` is responsible for processing all Solr queries into the Solr engine. The `SolrDispatchFilter` is initialized during application bootstrap, after the `ServletContextListeners` defined in `web.xml` have been processed. The `APSolrDispatchFilter` was created so that AgencyPortal's application framework properties can be used in the derivation of initialization parameters for the Solr engine. This approach is taken to provide a more convenient means of configuring Solr in both non-clustered and clustered SolrCloud environments (since environment specific framework properties are easier to work with than `JVM` arguments). For more information, refer to the [Setting Up Solr in AgencyPortal](#) topic in the Getting Started section.

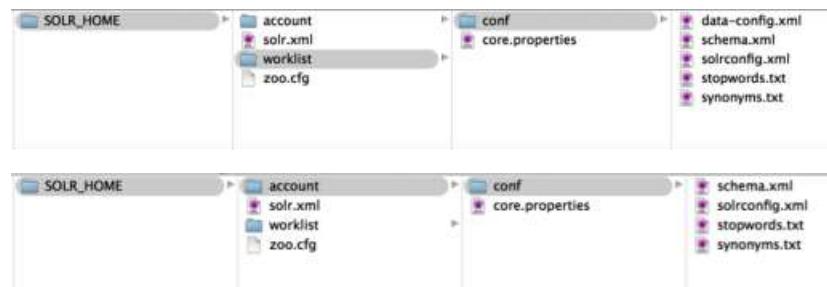
## Solr Configuration Files

This section describes the configuration files that drive the Solr functionality. Refer to the [Solr documentation](#) on the Apache site for in-depth details.

### Solr Home Folder Structure

When Solr runs in an application server, it needs access to a home directory. The home directory contains important configuration information and is the place where Solr stores its index.

The following displays a typical home directory structure:



Configuration files `solr.xml`, `solrconfig.xml` and `schema.xml` determine how Solr works.

### Schema.xml Configuration

The `schema.xml` configuration file defines:

- field types for fields in a document
- unique/primary key fields
- field characteristic, such as required, indexed, stored or multiValued
- how fields are indexed and searched

## Example

```
<schema name="account" version="1.1"> <types> <fieldtype name="string" class="solr.StrField" sortMissingLast="true" omitNorms="true"/>
<!-- A text field that only splits on whitespace for exact matching of words --> <fieldType name="text_ws" class="solr.TextField"
positionIncrementGap="100"> <analyzer> <tokenizer class="solrWhitespaceTokenizerFactory"/></analyzer> </fieldType> <!-- Partial match ->
<fieldType name="partial_search_type" class="solr.TextField"> <analyzer type="index"> <tokenizer
class="solrWhitespaceTokenizerFactory"/><filter class="solr.NGramFilterFactory" minGramSize="3" maxGramSize="100" /><filter
class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" enablePositionIncrements="true"/><filter
class="solrLowerCaseFilterFactory"/> <filter class="solr.TrimFilterFactory" /><filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
</analyzer> <analyzer type="query"> <tokenizer class="solr.StandardTokenizerFactory"/><filter class="solr.WordDelimiterFilterFactory"
generateWordParts="1" generateNumberParts="1" catenateWords="0" catenateNumbers="0" catenateAll="0" splitOnCaseChange="1"/><filter
class="solrLowerCaseFilterFactory"/> <filter class="solr.TrimFilterFactory" /><filter class="solr.SynonymFilterFactory"
synonyms="synonyms.txt" ignoreCase="true" expand="true"/></analyzer> </fieldType> </types> <fields> <!-- field in the document --> <field
name="id" type="string" indexed="true" stored="true" multiValued="false" required="true"/> <field name="_version_" type="long"
indexed="true" stored="true"/> <field name="type" type="string" indexed="true" stored="true" multiValued="false"/> <field name="firstName"
type="text_ws" indexed="false" stored="true" multiValued="false"/> <field name="lastName" type="text_ws" indexed="false" stored="true"
multiValued="false"/> <field name="companyName" type="text_ws" indexed="false" stored="true" multiValued="false"/> <field
name="normalText" type="text_ws" indexed="true" stored="true" multiValued="true"/> <copyField source="firstName" dest="normalText"/>
<copyField source="lastName" dest="normalText"/> <copyField source="companyName" dest="normalText"/> <field
name="partial_search_text" type="partial_search_type" indexed="true" stored="true" multiValued="true"/> <copyField source="firstName"
dest="partial_search_text"/><copyField source="lastName" dest="partial_search_text"/><copyField source="companyName"
dest="partial_search_text"/></fields> <!-- field to use to determine and enforce document uniqueness. --> <uniqueKey>id</uniqueKey> <!--
field for the QueryParser to use when an explicit fieldname is absent --> <defaultSearchField>partial_search_text</defaultSearchField> <!--
SolrQueryParser configuration: defaultOperator="AND/OR" --> <solrQueryParser defaultOperator="OR"/> </schema>
```

## solrConfig.xml Configuration

The solrConfig.xml file defines:

- the data directory location
- request handlers
- search components
- cache parameters

## Example

```
<config> <luceneMatchVersion>LUCENE_43</luceneMatchVersion> <directoryFactory name="DirectoryFactory"
class="${solr.directoryFactory:solr.StandardDirectoryFactory}"/> <dataDir>${solr.account.data.dir:</dataDir> <schemaFactory
class="ClassicIndexSchemaFactory"/> <updateHandler class="solr.DirectUpdateHandler2"> <autoSoftCommit> <maxTime>1000</maxTime>
</autoSoftCommit> <autoCommit> <maxDocs>100</maxDocs> <maxTime>1800000</maxTime> <openSearcher>false</openSearcher>
</autoCommit> <updateLog> <str name="dir">${solr.account.data.dir:</str>} </updateLog> </updateHandler> <requestHandler
name="standard" class="solr.SearchHandler"> <lst name="defaults"> <str name="defType">dismax</str> <str
name="echoParams">explicit</str> <str name="tie">0.01</str> <str name="qf">partial_search_text^1.0 normalText^3.0</str> <str
name="pf">partial_search_text^1.0 normalText^3.0</str> <str name="mm">3<90%</str> <int name="ps">100</int> <str
name="q.alt">*:*</str> </lst> <arr name="last-components"> <str>spellcheck</str> </arr> </requestHandler> <requestHandler
name="/analysis/field" startup="lazy" class="solr.FieldAnalysisRequestHandler"/> <requestHandler name="/update"
class="solr.UpdateRequestHandler"/> <requestHandler name="/admin/" class="org.apache.solr.handler.admin.AdminHandlers"/>
<requestHandler name="/admin/ping" class="solr.PingRequestHandler"> <lst name="invariants"> <str name="q">solrpingquery</str> </lst>
<lst name="defaults"> <str name="echoParams">all</str> </lst> </requestHandler> <searchComponent name="spellcheck"
class="solr.SpellCheckComponent"> <str name="queryAnalyzerFieldType">partial_search_text</str> <!-- a spellchecker built from a field of
the main index --> <lst name="spellchecker"> <str name="name">default</str> <str name="field">partial_search_text</str> <str
name="className">solr.DirectSolrSpellChecker</str> <!-- the spellcheck distance measure used, the default is the internal levenshtein --> <str
name="distanceMeasure">internal</str> <!-- minimum accuracy needed to be considered a valid spellcheck suggestion --> <float
name="accuracy">0.5</float> <!-- the maximum # edits we consider when enumerating terms: can be 1 or 2 --> <int name="maxEdits">2</int>
<!-- the minimum shared prefix when enumerating terms --> <int name="minPrefix">1</int> <!-- maximum number of inspections per result. -->
<int name="maxInspections">5</int> <!-- minimum length of a query term to be considered for correction --> <int
name="minQueryLength">4</int> <!-- maximum threshold of documents a query term can appear to be considered for correction --> <float
name="maxQueryFrequency">0.01</float> <!-- uncomment this to require suggestions to occur in 1% of the documents <float
name="thresholdTokenFrequency">.01</float> --> </lst> </searchComponent> <!-- config for the admin interface --> <admin>
<defaultQuery>*:*</defaultQuery> </admin> </config>
```

## Solr.xml Configuration

The solr.xml file defines one or more Solr cores. Multiple cores allow you to have a single Solr instance with separate configurations and indexes with their own configuration and schema for very different applications. Core discovery defines some configuration parameters in solr.xml, but no cores are defined in this file. Instead, the Solr home directory is recursively walked until a core.properties file is encountered.

This file is presumed to be at the root of a core, and many of the options placed in the <core> tag in legacy Solr are defined here as a simple properties (i.e., a file with entries, such as 'name=core1,' 'schema=myschema.xml', etc).

### Example

```
<solr> <solrcloud> <str name="host">${host:10.21.39.153}</str> <int name="hostPort">${portal.port:8080}</int> <str name="hostContext">${hostContext:agencyportal/solr}</str> <int name="zkClientTimeout">${zkClientTimeout:15000}</int> <bool name="genericCoreNodeNames">${genericCoreNodeNames:true}</bool> </solrcloud> <shardHandlerFactory name="shardHandlerFactory" class="HttpShardHandlerFactory"><int name="socketTimeout">${socketTimeout:0}</int> <int name="connTimeout">${connTimeout:0}</int> </shardHandlerFactory> </solr>
```

### Note

The presence of the <solrcloud> element does not mean that the Solr instance is running in SolrCloud mode. Unless the -DzkHost or -DzkRun are specified at startup time, this section is ignored.

# SolrManager

The `ISolrManager` interface and `SolrManager` concrete implementation provide a simplified interface for interacting with the Solr server to search, index and delete. It also provides hook points for tasks, such as modifying JSON responses back to the client and automatically applying ACSI security constraints.

## Using SolrManager

Use `SolrManager` to do the following:

- [index](#)
- [delete](#)
- [search](#)

### Indexing

To index, there is an `index()` method that accepts a list of `SolrInputDocument` instances. These are simple collections of fields with a string ID and string value that model the index. The collection is sent as a whole to Solr for indexing.

### Deleting

To delete, provide a list of strings that represent the IDs of the indexed documents to delete. For example, the worklist uses the work item ID as its ID; therefore, to delete work items 1001, 1002 and 1003, provide a list with strings “1001,” “1002” and “1003.”

### Searching

#### Determining which Index to Use

Each call to Solr in `SolrManager` accepts an `indexType` parameter. This parameter is used with the configured `ISolrResolver` instance to produce a URL that points to a Solr index. By default, mappings to account and worklist URLs are supported. For projects that want to use Solr for other features, you must implement `ISolrUrlResolver` or extend `SolrUrlResolver` to provide additional mappings to suit their purposes.

#### Subclassing SolrManager or Implementing ISolrManager on a Custom Classes

You do not need to subclass `SolrManager` or implement `ISolrManager` on one of the custom classes, at least not for interacting with Solr. These classes provide fairly generic methods for making calls to Solr; parameters should accommodate any index type.

- The `index` method accepts a list of `SolrInputDocuments`, which can contain any fields and are not domain-specific.
- The `search` method takes a `SolrQuery` object, which can also contain any fields and is not domain-specific.
- The `delete` method takes a list of string IDs. Since all indexes are configured with an `index` field, this is also not domain-specific.

You may need to extend ACSI integration. This takes place in the `applySecurityFilter` method, which accepts a `SolrQuery` parameter that you can modify.

#### Note

If an implementation project decides to override the `index/search/delete` methods, you must call the following after acquiring the URL for the desired index:

```
SOLR_Server.setBaseUrl(baseUrl);
```

This is performed because the `HttpSolrServer` is instantiated with a single URL and you want to use it to communicate with multiple URLs. You can:

- Remove the singleton instance and initialize a new `HttpSolrServer` instance for each call, or
- Reset the base URL that is used with each call. It is more efficient to reset the URL.

## Search/Index/Delete Calls

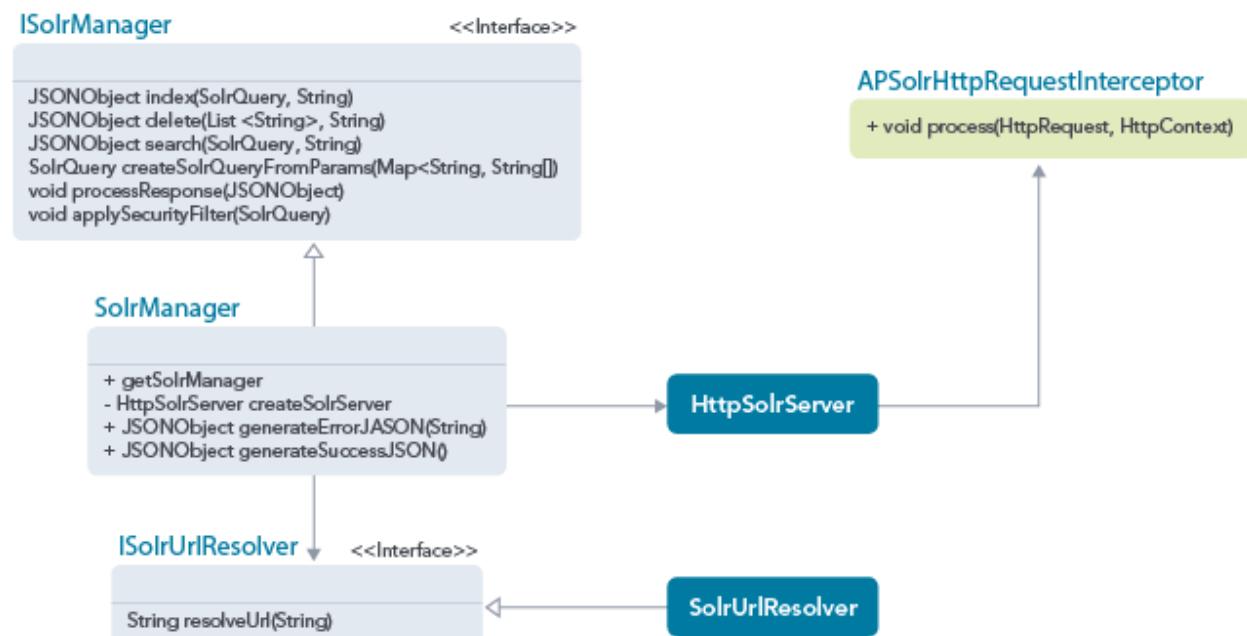
When the `ISolrManager` instance class loads, an instance of `HttpSolrServer` (of the Apache SolrJ library class that actually does the HTTP communication to the Solr server) is created and configured with the appropriate Solr URL. Each `HttpSolrServer` instance contains an `HttpClient` instance.

An `HttpRequestInterceptor` implementation called `APSolrHttpRequestInterceptor`. class is invoked prior to each HTTP request made by `HttpClient` and attaches the `X-Agencyport-Token` and `X-Agencyport-Token-Type` security header onto the request. The presence of this token satisfies the `SolrFilter` and lets the request pass through to Solr.

### framework.properties Attributes

Be aware of the following `framework.properties` attributes:

- `solr_manager` - The fully qualified package name for the concrete instance of `ISolrManager` used in the application. This defaults to `com.agencyport.solr.manager.SolrManager`.
- `solr_url_resolver` - The fully qualified package name for the concrete instance of `ISolrUrlResolver` used in the application. This defaults to `com.agencyport.solr.manager.SolrUrlResolver`.



# SearchManager

The search interface (search-manager.js) is driven by [Twitter's typeahead library](#), which consists of two parts:

- Bloodhound – a suggestion engine
- Typeahead – handles UI related tasks

Bloodhound is used for the purpose of interacting with Solr. When `SearchManager` is initialized, an options hash is passed, which contains, among other attributes, a `solrUrl`. By default, this is configured as the URL for the `SolrSecurityServlet` in `WorkListMiscDataPopulator`. We strongly suggest that you do not change this value.

There is also an option to manually construct the options that Bloodhound is initialized with. This gives you the flexibility to specify the function called to construct the Solr query. Place this function in `bloodhoundOpts.remote.replace`.

The rendering component exists as a function called `renderSolrData()` within `search-manager.js`. This takes the JSON response from Solr and parses it out into an array of JSON objects, keyed by value.

## Example

```
[{value: "Item 1"}, {value: "Item 2"}, ..., {value: "Item n"}]
```

This is handed off to the typeahead component for HTML rendering.

# Solr Batch Indexing

Batch indexing allows AgencyPortal to completely sync an index with its data source (i.e., the worklist index with the `work_item` database table or the account index with the `account` database table). This is a useful tool for developers when a data source or Solr schema is constantly changing or for when an existing project with certain data-like work items or accounts is upgraded to AgencyPortal 5.2.

This feature allows a user (configured out-of-the-box for an admin role) to use a web interface for easy re-indexing of configured Solr indexes. The indexes are presented in a table with the index label and an Update Index button. When you click the Update Index button, Solr is prompted to delete the current index entirely and re-fetch all entities in the configured data source to re-index.

By default, this feature is available via a menu entry for configured users. Refer to the [Accessing Batch Indexers](#) topic for more information.

## Solr Batch Index Processing

A Solr batch index process involves first sending a "delete all" requests to Solr. We then lift up every applicable record (accounts or work items in our case) from the database and send them over to Solr via HTTP. When we're dealing with production scale, there are two concerns that need to be accommodated:

1. If you load all of the database records into memory with the intention of sending one massive Solr index request, then you will surely run out of memory and crash your system.
2. Conversely, sending database records over to Solr one at a time, as you process the SQL query results, would likely expose a bottleneck at the network layer (since one million database records would result in one million HTTP requests).

The best compromise is to send the document over to Solr in appropriately sized batches. This is how AgencyPortal's Solr batch index processor works. Out of the box, AgencyPortal will read and send batch of 10000 documents to Solr at a time. This can be configured on a per index basis by specifying the following application property in `framework.properties`:

```
<index type>.batch_index_size_threshold=20000
```

(replace `<index type>` with one of the specific indexes you're trying to configure (i.e., "account", "worklist" or "some\_custom\_index").

# Working with Solr Batch Indexers

Out of the box, the SDK ships with two `ISolrBatchIndexer` implementations: one for the account search index (`AccountSolrBatchIndexer`) and one for the regular work item (a.k.a. “worklist” search index `index`) (`WorkListSolrBatchIndexer`). These two classes are responsible for querying the database for all records associated with a search index, and then creating corresponding `SolrInputDocument` objects for each record. Each `SolrInputDocument` object is provided to the given `ISolrDocumentIndexer` implementation, which is responsible for sending the `SolrInputDocument` objects to Solr over HTTP, in appropriately sized batches.

The [Index Mapping](#) configuration files, which are registered to the application, are responsible for mapping the JDBC `ResultSet` from a database query to `SolrInputDocument` objects. The `AbstractSolrBatchIndexer`, which is the base class of the aforementioned `ISolrBatchIndexer` implementations, takes care of processing the database query and utilizing the Index Mapping configuration available to create `SolrInputDocument` objects. In essence, this means that the leaf `AccountSolrBatchIndexer` and `WorkListSolrBatchIndexer` classes are only responsible for providing an appropriate JDBC `PreparedStatement` object, which will return the records corresponding to the search index.

## Example

The following is an example of what the `WorklistSolrBatchIndexer` class looks like:

```

/**
 * SolrBatchIndexer responsible for creating Solr XML for batch updates on work
items.
 */
public class WorkListSolrBatchIndexer extends AbstractSolrBatchIndexer {

 /**
 * The <code>QUERY_STATEMENT</code> is the select statement that will gather
all of the work items to apply to the SOLR index.
 */
 private static final String QUERY_STATEMENT =
DatabaseAgent.databaseNormalizeSQLStatement("select * from
${db_table_prefix}work_item w "
 + "left outer join ${db_table_prefix}account_member am on
w.work_item_id = am.work_item_id "
 + "left outer join ${db_table_prefix}external_relation er on
w.work_item_id = er.internal_id and er.relation_type = 'W'");

 /**
 * Default constructor.
 */
 public WorkListSolrBatchIndexer() {
 super(IndexNames.WORKITEM_SOLR_INDEX_NAME,
WorkItemType.REGULAR_LOB_WORK_ITEM_TYPE);
 }

 /**
 * {@inheritDoc}
 */
 @Override
 protected PreparedStatement getPreparedStatement(DatabaseResourceAgent dra)
throws SQLException {
 return dra.getPreparedStatement(QUERY_STATEMENT);
 }
}

```

If there is a need to modify the database query used for work item or account batch indexing, then a custom ISolrBatchIndexer can be used in place of the out of the box implementations referenced above. Modify the solr\_batch\_indexers property specified in framework.properties to nominate your custom implementation class. The format for this property is "<index\_name>|<batch\_indexer\_classname>" (tokenized).

### Best Practice

The new class should extend the SDK's AbstractSolrBatchIndexer class. A subclass of the AbstractSolrBatchIndexer class only needs to provide the JDBC PreparedStatement, which will pull the appropriate records from the database. The AbstractSolrBatchIndexer uses the Solr [Index Mappings](#) registered to the application to take care of mapping the ResultSet from the query to fields in the SolrInputDocuments that will be sent to Solr.

# Accessing Batch Indexers

A UI tool configured at /SolrBatchUpdate provides a table of configured batch indexers, along with a button to initiate re-indexing. This table is driven based on the `solr_batch_indexers` property and is only available to administrator roles by default.

The page is rendered by Web Root/solrbatchupdate/solrbatchupdate.jsp if a project needs to edit the UI.

JSON is returned for batch update calls. The format is:

```
{ responseHeader: { status:<0 for success, 1 for error>, description: "some error text" // only exists if status=1 } response: { <data from Solr> }
```

## framework.properties

The `solr_batch_indexers` property lists the indexers available to AgencyPortal. The format for each entry is:

```
<index type>|<class name>
```

Each entry is separated by a semicolon.

### Example

The setup for the default account and worklist indexers display as follows:

```
solr_batch_indexers=worklist|com.agencyport.worklist.solr.WorkListSolrBatchIndexer;account|com.agencyport.account.solr.AccountSolrBatchIndexer
```

## General Setup for Access

The menu entry is configured in `menu.xml` with the following line:

```
<menuMember name="Solr Admin" icon="fa fa-cogs" description="Solr Administration" url="SolrBatchUpdate" permission="accessMenu_Solr_Admin"/>
```

This provides access for any user configured with the `accessMenu_Solr_Admin` permission.

# Index Mapping

In AgencyPortal 5, the shredding of work item and account data over to their respective Solr indices was accomplished completely with Java code (both in the framework and at the application level). 5.1 introduced a configuration based approach to updating Solr indices.

## Index Mapping Schema

An index mapping schema defines the structure for conveying how (Solr) index fields map to their AP counterparts. Multiple indices are supported, although it is believed that most applications will only need two at most: one for the work item table and one for the account table. Each index mapping instance has an index name that refers to its physical search index. It also has zero or more field definitions that each contain the following properties. Each field has one or more content elements that will provide the localization information specific to a locale/field combination.

Property Name	Data Type	Required	Default Value	Comments
id	xs:ID	True	N/A	Provides the unique identity of this field. The value must be unique within an index mapping instance.
name	xs:string	False	<id>value	Links to the column name of the corresponding AP table of record.
indexFieldName	xs:string	False	<name>value	Links to the column name of the corresponding search index.
indexSearchFieldName	xs:string	False	None	Designates an alternate search name, such as entity_name_search. This is used instead of the index field name value when supplied.
workItemPropertyName	xs:string	False	Upper case version of <name> value	Maps to the IWorkItem property name.
isSearchable	xs:boolean	False	True	Determines whether this field is searchable or not. There may be some fields whose values are needed to constitute search index values that are a concatenation/derivation of one or more AP data values.

## Resource Type

The index mapping resource type can be configured to Solr index field mapping. The index mapping resource type has its own unique product resource provider and launches during application bootstrap time. The resource provider processes the XML artifacts registered in the application's product database for the type into its own specific compiled in-memory representation. This representation is used at runtime rather than its raw JDOM elements.

The following illustrates the index mapping resource type, including the resource providers and compiled runtime representations:



## Compiled Runtime Representations

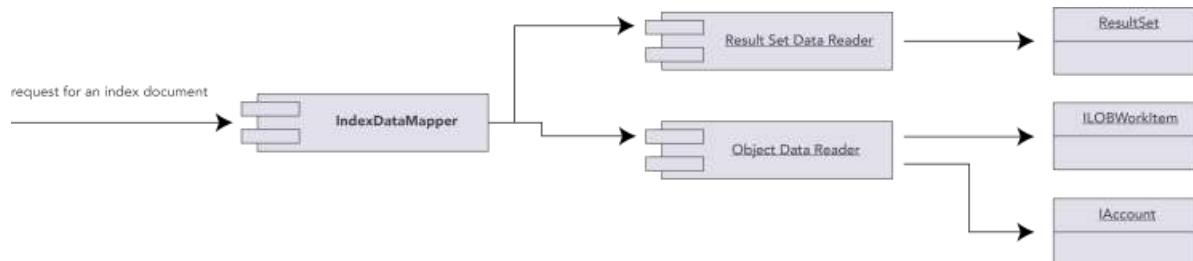
### Compiled Index Mapping

Compiled index mapping objects serve the following purposes:

- Serves as the compilation runtime unit for index mapping structures conforming to the XML schema. Refer to the [Index Mapping Schema](#) for more information.
- Provides the mapping between AP system of record data stores (account and work\_item tables) and their respective in memory data objects (IAccount and ILOBWorkItem) with their respective Solr search counterparts. These mappings come into play when work items and accounts are updated real-time, as well as when the Solr indices are rebuilt from scratch.
- Supplies the field definitions referred to by the various work list view definition configuration files.
- Provides the data types for each field.
- Provides locale information for each field, including their localized titles and formatting information. This information is then leveraged by the UI.

### Index Data Mapper (Synchronization of Search Indices)

The synchronization of search indices with their respective AP table of record are engaged with each commit to the AP table of record. Batch synchronization mechanisms are also implemented to fully update a search index to support search index schema changes during development, as well as those that may be encountered in production. The index data mapper provides a generic mechanism to create an "index document" that contains the data values from either the AP table of record (account or work\_item) or object of record (IAccount or ILOBWorkItem). An interface called `IDataReader` provides a generalization for accessing a JDBC result set or an AP object of record in a canonical manner.



## Communication and Invocation

### Salient APIs

This section outlines the salient provider interfaces that provide the bulk of this functionality.

<b>com.agencyport.searchindex.ISearchIndexMapping</b>	<p>This interface defines the basic contract for providing access to field instances for a given index mapping field instance. This is an abstraction around this concept, allowing extension or full interface implementation where necessary. Instances are brought to life by the <code>IndexMappingResourceProvider</code> class. For applications to instantiate their own custom implementations of this interface, they must create their own custom resource provider. By default, instances of <code>CompiledIndexMapping</code> type are created. The following are notable methods. Note the support for sequence iteration of fields.</p>
-------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<ul style="list-style-type: none"> <li>• <code>getField()</code> - This method gets a field instance for the given name for the current locale.</li> <li>• <code>getLocalizedContent()</code> - This method gets the localized content for a given field.</li> </ul> <p>Refer to the <a href="#">JavaDoc</a> for more information.</p>
<b>com.agencyport.searchindex.IDataReader</b>	<p>This interface generalizes the reading of data values from disparate data sources. It comes into play when an update to the search index is deemed necessary. There are currently four implementations of this interface: one for the <code>IAccount</code> object, one for an account table result set, one for an <code>ILOBWorkItem</code> and the last for the <code>work_item</code> table result set. The <code>com.agencyport.searchindex.SearchIndexFactory</code> factory has two supporting factory methods: one for the JDBC result set data reader (<code>account</code> or <code>work_item</code>) and one for the AP data object (<code>IAccount</code> or <code>ILOBWorkItem</code>). Applications may need to extend one or more of these implementations if there are situations when there is not a straight forward, one-to-one relationship between AP fields and their corresponding index field.</p>

# Solr Security

The `com.agencyport.solr.filter.SolrFilter` class sits in front of requests to `/solr/*` and is responsible for determining whether requests to Solr's REST interface adhere to the project's security policy. There are two categories of requests that the `SolrFilter` anticipates. Those requests, which are triggered by or on behalf of a user and have a subject associated with them in some way, are considered standard requests. Those requests, which are systemlevel, non-usercentric and have no associated subject, are considered to be internal (e.g., SolrCloud node to node requests). Standard and internal requests are secured by decorating them with different types of security tokens. The main difference is that when validating standard requests, you must determine who is the associated subject. Requests evaluated by the `SolrFilter` must contain two request headers:

- The "X-Agencyport-Token" contains the encrypted Solr security token (Hex encoded). For standard requests, this token is constructed as a string concatenation of <an internal Solr key><pipe character><login ID><pipe character><timestamp in MM-dd-yyyy HH:mm:ss format>. For internal requests, the token's makeup is similar, but there is no login (since internal requests are non-usercentric). Its format is a string concatenation of <an internal Solr key><pipe character><timestamp in MM-dd-yyyy HH:mm:ss format>.
- The "X-Agencyport-Token-Type" request header tells the `SolrFilter` whether this is an internal or standard request. Effectively, this is a processing instruction that determines how to interpret the "X-Agencyport-Token" header.

Additionally, to make the locale-specific error messages available, include the `local` request parameter with the string representation of the locale as its value.

## Note

The internal Solr key referred to above is a secret key that is read from the `solr.key.keystore` file. To use your own key or keystore file, along with its associated password, you can specify the `solr_keystore_filename` and the `solr_keystore_password_filename` application properties (refer to the Keystore Files section in the [Encryption](#) topic for more information about secure password files). Otherwise, you can create a custom `ISolrSecurityManager` implementation and register it with the `solr_security_manager` application property.

## Token Validity

To determine whether the token is valid, the `SolrSecurityManager`'s `validateInternalToken` method for internal requests or `validateTokenAndGetSubject` method for standard requests is called by the `SolrFilter` for each incoming request.

- If the token provided is either null or the timestamp cannot successfully parse out of the token (after decryption), or the internal Solr key provided is incorrect, the `SecurityException` displays with the following `ResourceBundle` message: `message.error_token_invalid`.
- If the timestamp is successfully parsed out, then a comparison is done to ensure the token was sent within the last five seconds (based on the server's clock). A failed check displays a security exception with the following `ResourceBundle` message: `message.error_token_expired`.
- For standard requests, a check takes place to ensure that the login ID is valid by using it to inflate a subject.

If all checks pass, the validation method returns true to the caller.

## Token Generation

For both internal and standard requests, the token generates by the `generateStandardToken` (`ISecurityProfile`) and `generateInternalToken()` methods in the default `SolrSecurityManager` implementation of the `ISolrSecurityManager` interface. If your delivery team needs to customize the token generation process, provide your own `ISolrSecurityManager` implementation by specifying your custom class name in framework properties with the `solr_security_manager` attribute.

## Attaching the Token to an HTTP Request

Within the application code, use an implementation of `ISolrManager` to interact with Solr (the product ships with a default implementation called `SolrManager`). Security tokens are automatically generated and attached to the request to Solr by the `APSolrHttpRequestInterceptor` class, which is invoked by `HttpClient`. However, if there is a need to attach additional security headers (or basic authentication information) to the outbound request to Solr, then you can override the default `APSolrHttpRequestInterceptor` by nominating your own custom extension class with the following application property:

```
solr_http_request_interceptor=com.prestige.solr.security.PrestigeSolrHttpRequestInterceptor
```

For a client side Solr interaction, point the request to the endpoint configured for `SolrSecurityServlet` (refer to [Solr Security Servlet](#)).

## SolrSecurityServlet

The SolrSecurityServlet class acts as a proxy for calls to Solr from client-side AJAX calls. This allows those calls to go through the appropriate security measures, such as extracting or constructing a security profile and using AgencyPortal's API for Solr interaction, and provide a controlled mechanism for making Solr calls when those calls don't originate from Java code.

Since the calls go through the `ISolrManager` instance, there's no need to worry about generating a security token.

For all client-side requests attempting to reach the Solr REST interface, target the URL configured for the SolrSecurityServlet.

### Request Requirements (GETs and POSTs)

For standard requests, such as the following, a logon ID is necessary to satisfy the SolrFilter that guards the Solr REST interface. The servlet attempts to extract a security profile from the incoming request and, if one is not found, it looks for a `logon_id` parameter.

- If either of these is found, the logon ID is acquired and processing continues.
- If both of these are not found, a JSON error message generates and is sent back in the response to the client.

If the logon ID extraction is successful, execution is handed off to an instance of `ISolrManager`, which is responsible for token generation and sends the search or index query to Solr.

### ACSI Decoration for Solr Queries

ACSI decoration for Solr queries takes place inside the `ISolrManager` implementation class within the `applySecurityFilter()` method. The client-side does not need to be aware of usergroups when building search queries. When a search request comes into the `SolrManager` from the client (via the `SolrSecurityServlet`), the `applySecurityFilter()` method modifies the search query by adding additional usergroup constraints. This ensures that the client side does not see Solr results that the logged in user does not have access to.

### Response Messages

Responses throughout the Solr security infrastructure are in the form of JSON:

```
{ responseHeader: { status:[0 or 1] // 0 on success, 1 on error description: "text description for errors"//Will only exist if error occurs. }response:
{ // Stuff added by Solr on successful query. } }
```

Results or exceptions propagated from `SolrFilter` and `SolrManager` are then transformed in `SolrSecurityServlet` into JSON with the `SolrManager.generateErrorJSON` method.

### Custom Client Requests

You should target requests coming from external clients (AJAX, external Java applications, etc) to the configured SolrSecurityServlet URL. Depending on whether there is access to the authentication cookie (AJAX requests – yes; external Java `HttpURLConnection` – no), you may need to include a URL parameter of `logon_id` for authentication purposes. The `SolrSecurityServlet` has no ability to trigger an "internal" Solr request; therefore, all requests bound for the `SolrSecurityServlet` must be in some way attached to a user (even if the user is some sort of administrator or systemuser).

Configure clients to properly handle success or error responses based on what is detailed in the [Response Messages](#) section.

If you're writing custom Java code to access Solr from within the AgencyPortal application, we recommend you go through the `SolrManager` API directly instead of sending HTTP requests to the `SolrSecurityServlet`. Refer to the [SolrManager](#) topic for more information.

# Adding a New Field to an Existing Solr Core

A batch indexer is an Agencyport mechanism to provide easy re-indexing of an entire data source with just one click. An implementation of `ISolrBatchIndexer` is used by the batch indexing framework to clear out an entire index, query for all data from a configured data source and re-index all data. Refer to the [Solr Setup](#) topic in the Getting Started section for more information.

Perform the following to add a new field to an existing Solr core:

1. Modify the desired `conf/schema.xml` file in the appropriate Solr core to include the new field.
2. Modify the desired index mapping file for the Solr index to include the new field. If there exists a one to one mapping between the AgencyPortal data store and the Solr index, **then nothing additional is necessary**. If, however, the new Solr field is the result of the concatenation of two or more fields on the Agencyport side, then you will need to create two custom data mappers by extending the framework's default data mappers and overriding the `fetchValue` method to support the new derived field.

There is one data mapper that deals with JDBC result set oriented data formats and another that deals with `IWorkItem` oriented in memory data formats. There are four default framework data mappers in play: one for JDBC account data, one for IAcount data, one for JDBC work item data and one for ILOBWorkItem data formats. The data reader class names and their respective factory oriented application properties are:

Data Format	Class Name	Application Property
IAccount	<code>com.agencyport.account.impl.reader.AccountDataReader</code>	<code>account.data_reader_classname</code>
JDBC Account	<code>com.agencyport.account.impl.reader.AccountResultSetDataReader</code>	<code>account.result_set_data_reader_classname</code>
ILOBWorkItem	<code>com.agencyport.workitem.impl.reader.WorkItemDataReader</code>	<code>workitem.data_reader_classname</code>
JDBC ILOBWorkItem	<code>com.agencyport.workitem.impl.reader.WorkItemResultSetDataReader</code>	<code>workitem.result_set_data_reader_classname</code>

3. Run the associated batch indexer, if appropriate.

# Adding a New Solr Core (Index)

Solr cores are used to store all configuration files and indexed data for a particular index. Take the following approach if you want to index a completely new type of data (i.e., class codes), instead of simply adding more (related) data to an existing index:

1. In the application's configured SOLR\_HOME directory, create a new directory to house index data and configuration files:
  - conf (directory)
    - Solrconfig.xml
    - Schema.xml
  - core.properties  
(can contain a single line "name=XXX")

## Example

If you want to add an index for class codes, perform the following:

- a. Create a classcodes directory within SOLR\_HOME.
- b. Create a conf directory.
- c. Add solrconfig.xml and schema.xml files and edit, as necessary, to support the needs of this index. Refer to the following for appropriate edits:

[Comparison of worklist solrconfig.xml vs. classcodes solrconfig.xml](#)

Class Codes

```
<fields> <!-- general --> <field name="_version_" type="long" indexed="true" stored="true" multiValued="false"/> <field name="type" type="string" indexed="true" stored="true" multiValued="false"/> <field name="id" type="string" indexed="true" stored="true" required="true" multiValued="false" /> <field name="state_id" type="string" indexed="true" stored="true" required="true" multiValued="false" /> <field name="class_code_id" type="string" indexed="true" stored="true" required="true" multiValued="false" /> <field name="description" type="string" indexed="true" stored="true" required="true" multiValued="false" /></fields>
```

Worklist

```
<fields> <!-- general --> <field name="_version_" type="long" indexed="true" stored="true" multiValued="false"/> <field name="type" type="string" indexed="true" stored="true" multiValued="false"/> <field name="id" type="string" indexed="true" stored="true" required="true" multiValued="false" /> <field name="effective_date" type="date" indexed="true" stored="true" required="false" multiValued="false" /> <field name="lob" type="text_general" indexed="true" stored="true" required="true" multiValued="false" /> <field name="entity_name" type="string" indexed="true" stored="true" required="false" multiValued="false" /> <field name="status" type="int" indexed="true" stored="false" required="false" multiValued="false" /> <field name="workitem_status" type="text_general" indexed="true" stored="true" required="false" multiValued="false" /> <field name="transaction_id" type="text_general" indexed="true" stored="true" required="false" multiValued="false" /> <field name="transaction_type" type="text_general" indexed="true" stored="true" required="false" multiValued="false" /> <field name="lastupdate_time" type="date" indexed="true" stored="true" required="false" multiValued="false" /> <field name="usergroupid" type="int" indexed="true" stored="true" required="false" multiValued="false" /> <field name="creator_id" type="int" indexed="true" stored="true" required="false" multiValued="false" /> <field name="owner_id" type="int" indexed="true" stored="true" required="false" multiValued="false" /> <field name="owner_group_id" type="int" indexed="true" stored="true" required="false" multiValued="false" /> <field name="premium" type="float" indexed="true" stored="true" required="false" multiValued="false" /> <field name="complete_percent" type="int" indexed="true" stored="true" required="false" multiValued="false" /> <field name="commit_flag" type="int" indexed="true" stored="true" required="false" multiValued="false" /> <field name="account_id" type="int" indexed="true" stored="true" required="false" multiValued="false" /> <field name="entity_name_search" type="partial_search_type" indexed="true" stored="true" required="false" multiValued="false" /> <copyField source="entity_name" dest="entity_name_search"/><!-- catchall field, containing all other searchable text fields (implemented via copyField further on in this schema --> <field name="text" type="text_general" indexed="true" stored="false" required="true" multiValued="true" /></fields>
```

[Comparison of worklist schema.xml vs. classcodes schema.xml](#)

Worklist

```
<dataDir>${solr.worklist.data.dir:}</dataDir>
```

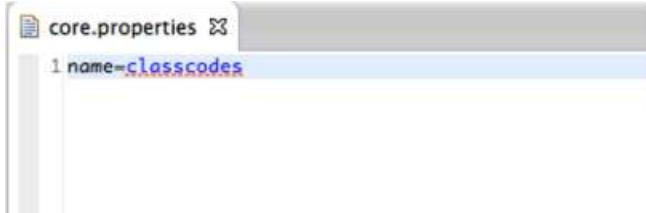
Class Codes

```
<dataDir>${solr.classcodes.data.dir:}</dataDir>
```

## Note

These changes are to simply provide a place for JVM system argument overrides for the data directory. We are currently not using the overrides.

- d. Create a core.properties file and edit core.properties to contain the text name=classcodes.



After starting the application, the index is reachable for querying and document manipulation. However, add batch indexing capabilities if there is existing data to immediately add to the index. Refer to [Adding a New Field to an Existing Solr Core](#) for the new index.

# Work Item Assistant

Work Item Assistant was developed in response to the increasing demand for collaboration technology within the portal. We have learned firsthand from agents that a high degree of collaboration between agents and a variety of parties at an insurance company are often required to issue new insurance policies. Numerous email, fax and telephone exchanges are required for business to take place between parties, increasing the operating expenses associated with the process and negatively impacting the purchasing experience for the insurance buyer.

Work Item Assistant helps alleviate much of the back and forth faxing, emailing and telephone calls by allowing increased communication within the portal. The assistant allows agents and insurance carrier employees to communicate real-time within a quote, upload file attachments at any point within the transaction and receive email notifications for specific events. This feature also gives users the ability to add comments and see comments added by other users.

## Work Item Assistant Configuration

You can enable or disable the Work Item Assistant at the system level. The following parameters are available via the framework.properties file:

Property	Description	Default Value	Recommended Setting (if any)
workitem_assistant_supported	Boolean flag that governs whether the work item assistant feature is active.	none	
max_file_attachment_size	The size limit for files uploaded via the work item assistant feature.	10240000	

Set this property to "true" to enable Work Item Assistant; otherwise, Work Item Assistant is not supported. The property is set to "true" by default.

## Server

Server support for the Work Item Assistant feature is implemented with REST service (refer to the [Client-Side Development](#) topic for more information on REST). com.agencyport.api.workitemassistant.WorkitemAssistantResource is the resource class that handles all processing. This class must be registered as one of the application.rest\_resource\_classes in framework.properties, such as the following:

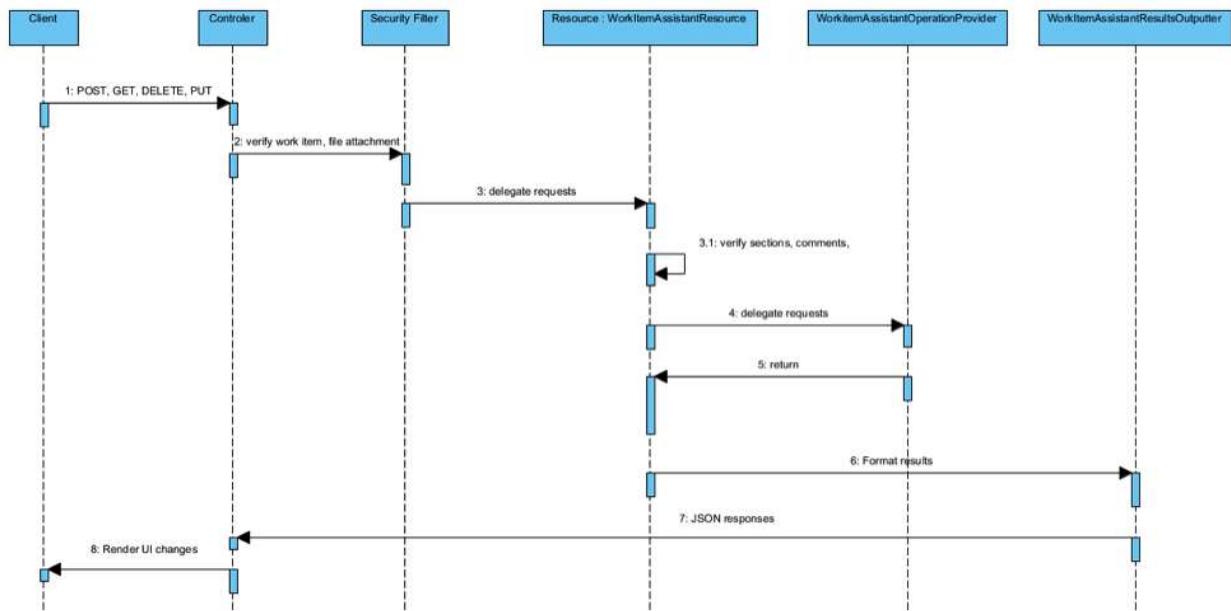
```
#####
REST API resource class registration.
#
application.rest_resource_classes=com.agencyport.api.product.ProductResource;\com.agencyport.api.product.RawProductResource;\com.agencyport.api.worklist.WorkItemListResource;\com.agencyport.api.worklist.AccountListResource;\com.agencyport.api.workitemassistant.WorkitemAssistantResource;
```

The following are the URIs for each of the Work Item Assistant operations:

- Comments (DELETE): /api/workitemassistants/pages/{{page id}}/sections/{{section id}}/comments
- Comments(POST): /api/workitemassistants/pages/{{page id}}/sections/{{section id}}/comments/contents
- Attached files (DELETE): /api/workitemassistants/pages/{{page id}}/sections/{{section id}}/fileattachments
- Attached file rename (PUT): /api/workitemassistants/pages/{{page id}}/sections/{{section id}}/fileattachments/filenames

The file deletion operation is permission based. A user with the canDeleteWorkItemAssistantAttachedFile permission can delete attached files.

The following sequence diagram describes the interactions between the controllers and models:



## Client

The implementation of the Work Item Assistant feature at the client level separates presentation, data and logic components. AngularJS is used on the client side (refer to the [Client-Side Development](#) topic for more information on AngularJS).

### Periodic Updates

The client issues periodic requests to synchronize the comments and file attachments. Templates are rendered in plain HTML according to data contained in a `scope` defined in the model. The `$scope` service in Angular detects changes to the model section and modifies HTML expressions in the view via a controller. AngularJS detects changes in models by comparing the current values with values stored earlier via a process called dirty-checking. Likewise, any alterations to the view are reflected in the model. This circumvents the need to actively manipulate the DOM.

The different sections in Work Item Assistant are periodically updated to show all of the latest comments, file attachments and other active users. The `client_update_interval` property controls how often the client sends these periodic updates to the back end to get the latest entries, or to check work item locking. The default is 15 seconds; the minimum value allowed is also 15 seconds.

A user may stop working on a work item for a while and leave the session idle. In this case, to reduce network traffic and to release connection resources and work item locking, the periodic update is terminated. The `client_update_timeout` property controls the length of time that the client can be idle before the periodic updates are stopped. The default is 600 seconds.

### Work Item Assistant Sections

Templates are structured to include the comments, file attachments, file upload and active users sections of the Work Item Assistant.

The screenshot displays the Work Item Assistant interface with four main sections:

- Comments**: Shows a comment from "Great Underwriter" stating "need more info on this policy" at "04-23-2015 21:52:05". A text input field and a "Comment" button are below.
- Documents**: Shows a file attachment "FL Heron Pest Control.pdf" (Agings of Receivables/Payables, 444 kb, 04-17-2015 12:43:26). An "Upload A File" section with a "Drag file here" area and an "Add Files" button follows.
- Upload A File**: Contains a "Drag file here" area with a document icon, an "Add Files" button, and a "Type" dropdown menu with "Select One" selected. "Upload" and "Cancel" buttons are also present.
- Users**: Shows the user "Great Underwriter".

The AngularJS controller, `workitemAssistantCtrl`, manages the data binding for comments, file attachments and active users. A child controller, `workitemAssistantUploadCtrl`, manages the file queue and uploads.

The content and functionality of the Work Item Assistant sections are controlled by the `workitemassistant.xml` artifact located under `shared/workitemassistant/`. You must add the artifact to `productDatabase.xml`.

### Example

```
<artifact type="workItemAssistant" resourceId="workitemassistant.xml" title="Shared work item assistant" description="Shared work item assistant"/>
```

You must consider the following when configuring each section of the Work Item Assistant. An example configuration is listed at the end of this topic.

- A section must have a type. `fileAttachment`, `comments` and `activeUsers` are supported.
- You can define multiple sections of the same type.
- A section must have an ID. The ID must be unique among all sections.
- A section must have a label. The label displays as the section's title in the Work Item Assistant user interface.
- The order of the sections in the Work Item Assistant is determined by the order of sections in `workitemassistant.xml`.
- The visibility of each section is controlled by user roles. If no roles are defined in a section, the section is visible to everyone. To add a role to `workitemassistant.xml`, the role name must be used.

## Example

If you want only underwriters to have the ability to view a comments section, define the sections as:

```
<section type="comments" id="uwcomments" label="UnderWriter Comments"> <role>underwriter</role> </section>
```

## File Attachments

You can set the size limit for the file attachments at the systemlevel using the `max_file_attachment_size` property. The default is 10MB.

## Transaction Definition File Reference

You must reference Work Item Assistant in the transaction definition file (TDF) for Work Item Assistant to be visible in the transaction. The `workItemAssistantId` list below must match the ID in `workitemassistant.xml`.

```
<?xml version="1.0" encoding="UTF-8"?> <transaction id="bop" title="Business Owners Application" target="BOPPolicyQuoteInqRq" lob="BOP" summaryPageId="policySummary" type="new_business" mappingId="bop" workItemAssistantId="workitemassistant" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://www.agencyport.com/schemas/4.2/definition/transactionDefinitionFile.xsd">
```

## Work Item Assistant and Product Create

If you create new products within the toolkit, you want to make sure the Work Item Assistant is available with those new products. To accomplish this, the following XML needs added to `productDatabase.xml` before using the Work Item Assistant. We recommend you add this to the shared product.

```
<artifact type="workItemAssistant" resourceId="workitemassistant.xml" title="work item assistant" description="work item assistant"/>
```

You must restart the application after making this change.

## Configuration Example

The following is an example of a configuration:

```
<?xml version="1.0" encoding="UTF-8"?> <workitemassistant id="workitemassistant" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://reference.agencyport.com/schemas/4.1/workitemassistant/workitemassistant.xsd"> <section type="fileAttachment" id="internalattachments" label="Internal Files"> <role>admin</role> </section> <section type="comments" id="uwcomments" label="UnderWriter Comments"> <role>underwriter</role> </section> <section type="comments" id="internalcomments" label="Internal Comments"> <role>admin</role> <role>underwriter</role> </section> <section type="fileAttachment" id="attachments" label="File Attachments"> <role>underwriter</role> </section> <section type="activeUsers" id="activeusers" label="Other Active Users"> <role>underwriter</role> </section> </workitemassistant>
```

## Email Notifications

If you want your Work Item Assistant's workflow to include email notifications, you can configure the following:

- the events that trigger an email message
- the recipients of that email

The email notifications feature is turned off by default. You must modify the `email_notifications.properties` file to use this feature. You can turn email notifications on at the systemlevel by setting `email.send_notifications=true`. Then, configure email notifications to send for selected events (work item status changes, comments added, file attachments added) and to selected users (creator, owner). The creator is the user who submitted/created the work item. The owner is the user the work item is assigned to.

## Example

A properties file where notifications are turned on for all events and users looks like the following:

```
email.send_notifications=true email.smtpHost=exchange03.corp.ap email.from_address=msanderson@agencyport.com
email.work_item_owner=true email.work_item_creator=true email.send_status_change_email=true email.send_file_attachment_email=true
email.send_comments_email=true
```

The `email.from_address` property determines the email address that displays in the From field of the email notification.

## Email Notification Configuration

Perform the following to set up email notifications:

1. Open the following properties file:  
`AgencyPortal/WEB-INF/email_notifications.properties`
2. Set  
`email.send_notifications=true`  
  
This turns on notifications.
3. Set the following property to your email server name:  
`email.smtpHost=`
4. Set the following property to the sender's email address:  
`email.from_address=`
5. Set the following properties for the events and users you want included:  
`email.work_item_owner=true`  
`email.work_item_creator=true`  
`email.send_status_change_email=true`  
`email.send_file_attachment_email=true`  
`email.send_comments_email=true`

### Tip

When testing in a local environment, configure the following property in `framework.properties`:

```
my_portal_app=9999
```

### Example

```
my_portal_app_protocol=http my_portal_app_domain=localhost my_portal_app_port=9999
```

### Tip

The email notifications are sent to the user's email address stored in your project's security model.

### Example

The following is a sample script you can run to update the user's email address:

```
UPDATE subject SET email_address='chrism@chrism.com' WHERE login_id='agent'
```

# Timeline

Timeline is a feature that provides an in-depth look into the lifetime of a work item, capturing important events and milestones from work item creation to business submission. This feature gives users the visibility to see who's changed work item data, what rules have been broken and when status changes have occurred.

Timeline tracks the following events:

- data modification
- rule triggering
- status change
- service call
- work item creations

## Note

Rule outcomes, service call and data modifications are tied to specific pages.

Work item creation events are tracked when the event occurs. Snapshots of ACORD XML is collected right after the work item is created for upload, endorsement and renewal. For new business work items, snapshots of ACORD XML is collected right after the status of the work item is changed.

If a status change occurs outside of the SDK, application developers must ensure these events are tracked. In addition, for custom pages, custom mechanism and implementation are required to detect data. Refer to the [Timeline Technical Overview](#) section for more information.

You have the choice to either turn on or turn off the Timeline feature. This is controlled at the `timeline_supported` global level property. If you change the control property value, you must also restart the server to use the new control parameter value. Refer to [Timeline Configuration](#) for more information.

It is assumed that the policy data change comparisons is based on a single transaction definition. If a quick quote application is converted into a new business application, the data or quote changes are processed based on the transaction definition of the event.

# Timeline Configuration

You must configure the following to use the Timeline feature in AgencyPortal:

1. Set the global-level property `timeline_supported` controls.

Property	Description	Default Value	Recommended Setting (if any)
<code>timeline_supported</code>	Indicates whether you want to turn on the Timeline feature.	false	

These controls indicate whether you want to turn on the Timeline feature and make it available to your users. If the value of the control property is changed, you must restart the server to allow the new control parameter value to be used. To enable Timeline, set the following in `framework.properties`: `timeline_supported=true`.

2. If `timeline_supported=true`, the event data and snapshot of policy XML for Work Item Created, Rule Triggered and Status Changed is immediately collected and stored in the database. The event data and snapshot of policy XML data modification is then collected after the first status change. Service call data is collected when service calls are configured.
3. Set the Transaction ID attribute `autoMaintainIdAttributes` to true. If this attribute is not true, an exception will be thrown.
4. Configure service data. The service data events are configured as follows with a workers compensation LOB as an example. You can set these configurations via Toolkit:

1. Create a connector to collect service data.

```
public class SampleServiceCallConnector extends ConnectorManager { public SampleServiceCallConnector () {} public Outcome execute(ConnectorDataBundle dataBundle) throws APException { APDataCollection apData = dataBundle.getDataCollection(); apData.setFieldValue("WorkCompLineBusiness.CommlCoverage[CoverageCd='W CEL'].CurrentTermAmt", "222310.00"); apData.setFieldValue("WorkCompLineBusiness.CommlCoverage[CoverageCd='ANN'].CurrentTermAmt", "2111301.00"); return Outcome.CONDITIONS_MET; } public boolean postProcess(ConnectorDataBundle dataBundle) { return true; } }
```

2. Add a service data instruction to the connector. The snapshot of ACORD XML is collected via instructions associated with the connector.

```
<connector type="custom" id="ServiceDataManager" className="com.agencyport.workerscomp.custom.SampleServiceCallConnector" title="ServiceDataManager"><executeWhen userAction="Continue"/></connector><instruction type="updateServiceData" title="Quote Data" id="ratingServiceData"> <when connector="ServiceDataManager" /></instruction>
```

3. Add service data to service data fieldset on a page. Fields will not display as change if you do not complete this step. Put these fields on a page in a service data fieldset.

```
<pageElement type="serviceData" legend="Service Call Data"> <fieldElement type="text" id="WorkCompLineBusiness.CommlCoverage[CoverageCd='W CEL'].CurrentTermAmt" /> <fieldElement type="text" id="WorkCompLineBusiness.CommlCoverage[CoverageCd='ANN'].CurrentTermAmt" /></pageElement>
```

4. Ensure the XPaths of the two added fields match the fields that are set in the sample connector.
5. Implement `executeCustomRuleMessagePreparation()` to ensure that users can view permitted rule messages since the SDK does not provide role-based message filtering. For rule triggering events, if the message are filtered with role-based message filter classes, use `CMDBaseProcessTimeline`.

# Timeline Technical Overview

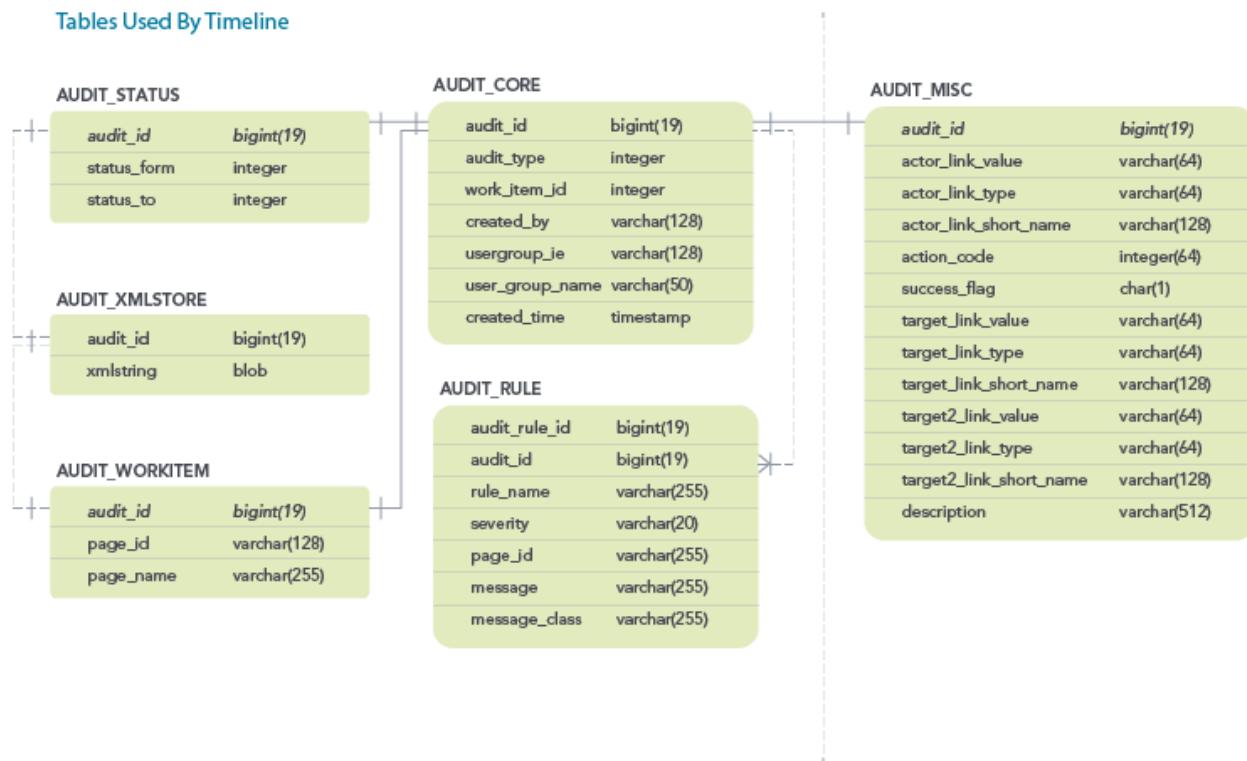
This section provides a technical overview of the components that make up the Timeline feature and are available to application developers setting up the Timeline feature in AgencyPortal.

## Data Model

The events and results of status changes and rule outcomes are audited and stored in AUDITLOG. The AUDITLOG table logs all information from all domains in the system, including user access, rule execution, toolkit operation, work item tracking and user's administration. There is required data in the timeline, but not recorded in the AUDITLOG table.

The following diagram details how to model all events and allow flexible control and easier data retrieval. The auditing data is stored in AUDIT\_CORE and the AUDIT\_RULE, AUDIT\_MISC, AUDIT\_STATUS, AUDIT\_WORKITEM and AUDIT\_XMLSTORE domain specific tables. The AUDITLOG table is still supported for backward compatibility.

### Tables Used By Timeline



### AUDIT\_CORE

The AUDIT\_CORE table can be referenced with the audit\_id column with all the domain specific audit tables. This table tracks whom and when the auditing event is created. The audit\_type column stores the types of auditing events that are used to link domain specific auditing tables, including the following:

Audit Type	Audit Table
STATUS_CHANGE	AUDIT_STATUS
RULE	AUDIT_RULE
WORKITEM_ADD	AUDIT_WORKITEM
DATA_MODIFIED	AUDIT_WORKITEM

Audit Type	Audit Table
SERVICE_DATA	AUDIT_WORKITEM
SECURITY_ADMIN	AUDIT_WORKITEM
SECURITY_ADMIN	AUDIT_MISC
SETTINGS_ADMIN	AUDIT_MISC
LOGIN	AUDIT_MISC
FRONT_DOOR	AUDIT_MISC
ASSIGN	AUDIT_MISC
TOOLKIT_SAVE	AUDIT_MISC
TOOLKIT_IMPORT	AUDIT_MISC
TOOLKIT_DELETE	AUDIT_MISC
TOOLKIT_EXPORT	AUDIT_MISC
TOOLKIT_CREATE	AUDIT_MISC

## AUDIT\_STATUS

The AUDIT\_STATUS table tracks a status change, including prior and current statuses. If the Timeline feature is supported, the ACORD XML of the work item is stored in AUDIT\_XMLSTORE when a status change occurs.

## AUDIT\_WORKITEM

The AUDIT\_WORKITEM table tracks a work item creation, service call data and data modification. If the Timeline feature is supported, the ACORD XML of the work item is stored in AUDIT\_XMLSTORE when a work item is created or when service call data is collected. Data modifications are tracked only after the status change of the work item is registered.

## AUDIT\_RULE

The AUDIT\_RULE table stores a rule firing outcome, including the page where the rule is attached, rule name, message, severity and message filter class. In some circumstances, an agent can trigger a rule and that same rule has an underwriter equivalent. However, the resulting messages can be different. You want to make sure that the underwriter rule is also stored so that when the underwriter comes in, he or she sees that specific rule message.

## AUDIT\_XMLSTORE

The AUDIT\_XMLSTORE table stores the snapshots of ACORD XML, if needed, for status change, work item creation, data modifications and service call data.

## Backward Compatibility

All auditing data was previously stored in the AUDITLOG table in prior releases. It is also likely that custom auditing is involved in some projects.

To ensure backward compatibility, the existing APIs and data structure for auditing is supported. The auditing data for work items, statuses and rule outcomes are persisted into their specific tables. The remaining auditing data, including admin, security tracking and toolkit (and possible custom data) is stored in the AUDIT\_MISC table. If timeline\_supported=true, snapshots of ACORD XML may be stored in AUDIT\_XMLSTORE.

## Event Collection

Auditing events occur under various workflows and scenarios. The following discusses how and where these events are controlled, tracked and created.

### Note

Any events that are not under these scenarios may require custom coding to track these events.

### Example

The status and data of a work item can be modified through direct database and ACORD XML manipulations.

### Control of Data Collection

The control of Timeline events occurs at a global level with the `timeline_supported` property, which is usually located in `frameworks.properties`. If `timeline_supported=true`, snapshots of ACORD XML is collected and stored in the `AUDIT_XMLSTORE` table.

### Rule Outcome

Rule outcome data is collected via `XarcConnectorManager`. To collect the rule severity and source of the rule execution, a new record is inserted into the `AUDIT_CORE` and `AUDIT_RULE` tables. If a rule contains multiple messages, each message is logged separately. If rules are executed under TVR, all rule outcome messages are associated with the same auditing event and display accordingly.

The following are some sample data for rule events:

The screenshot shows a database interface with two tables displayed side-by-side. The top table is titled 'Messages' and has columns: audit\_id, audit\_type, workitem\_id, create\_time, created\_by, user\_group\_id, and user\_group\_name. It contains three rows with data: (1, RULE, 1469, 2011-12-11 13:35:26.573, agent, 200, Example Agency), (2, RULE, 1469, 2011-12-12 10:29:38.667, agent, 200, Example Agency), and (3, RULE, 1469, 2011-12-12 10:30:04.020, agent, 200, Example Agency). The bottom table is titled 'Audit\_Rule' and has columns: audit\_rule\_id, audit\_id, rule\_name, severity, page\_id, page\_name, message, and message\_class. It contains six rows with data: (1, 22, 69, testDuplicate, error, generalInfo, Agent message, Agent), (2, 23, 69, testDuplicate, warning, generalInfo, Underwriter message, Underwriter), (3, 27, 74, testDuplicate, error, generalInfo, Agent message, Agent), (4, 28, 74, testDuplicate, warning, generalInfo, Underwriter message, Underwriter), (5, 29, 75, testDuplicate, error, generalInfo, Agent message, Agent), and (6, 30, 75, testDuplicate, warning, generalInfo, Underwriter message, Underwriter).

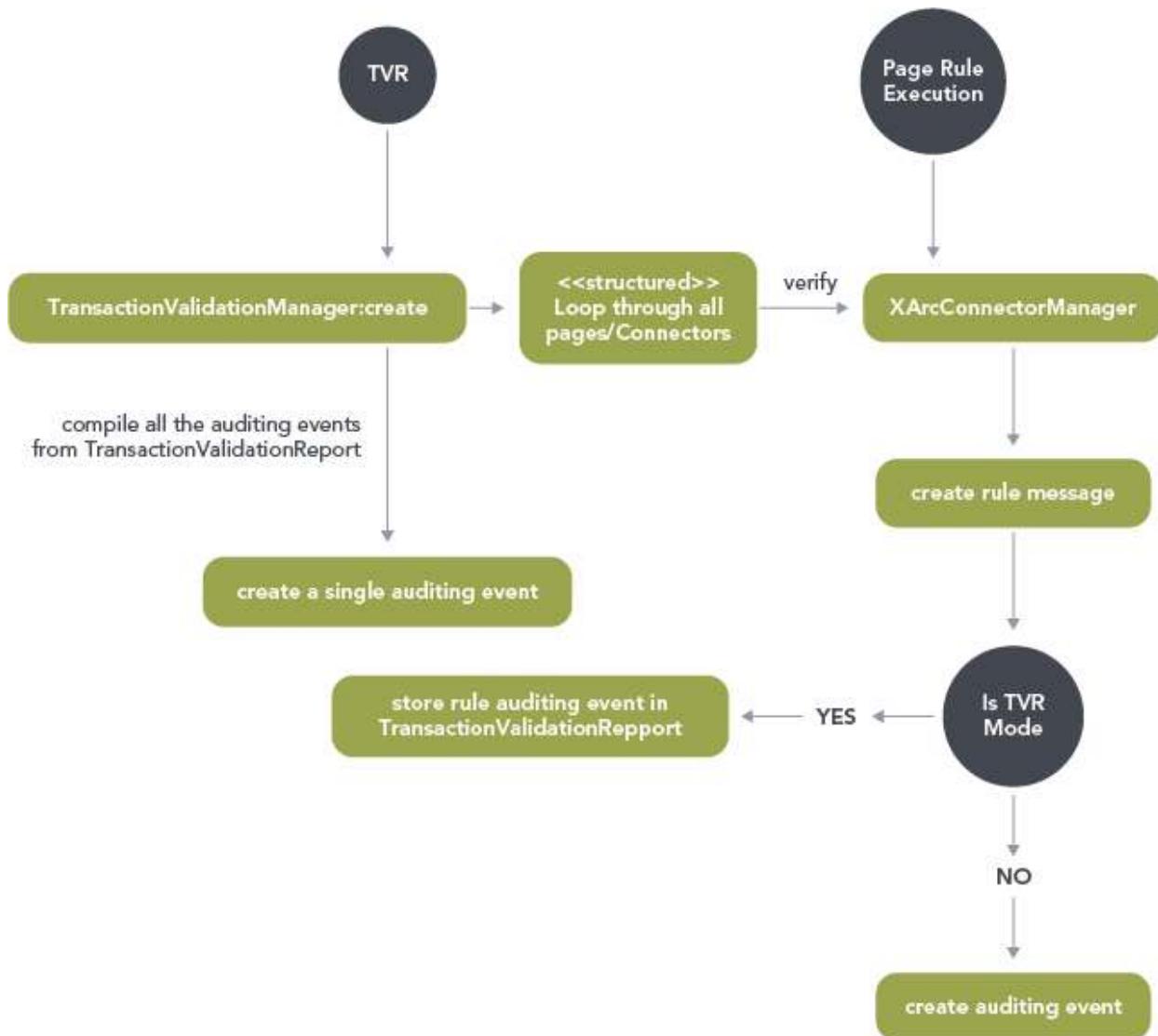
	audit_id	audit_type	workitem_id	create_time	created_by	user_group_id	user_group_name
1	69	RULE	1469	2011-12-11 13:35:26.573	agent	200	Example Agency
2	74	RULE	1469	2011-12-12 10:29:38.667	agent	200	Example Agency
3	75	RULE	1469	2011-12-12 10:30:04.020	agent	200	Example Agency

	audit_rule_id	audit_id	rule_name	severity	page_id	page_name	message	message_class
1	22	69	testDuplicate	error	generalInfo	Agency/Applicant Information	Agent message	Agent
2	23	69	testDuplicate	warning	generalInfo	Agency/Applicant Information	Underwriter message	Underwriter
3	27	74	testDuplicate	error	generalInfo	Agency/Applicant Information	Agent message	Agent
4	28	74	testDuplicate	warning	generalInfo	Agency/Applicant Information	Underwriter message	Underwriter
5	29	75	testDuplicate	error	generalInfo	Agency/Applicant Information	Agent message	Agent
6	30	75	testDuplicate	warning	generalInfo	Agency/Applicant Information	Underwriter message	Underwriter

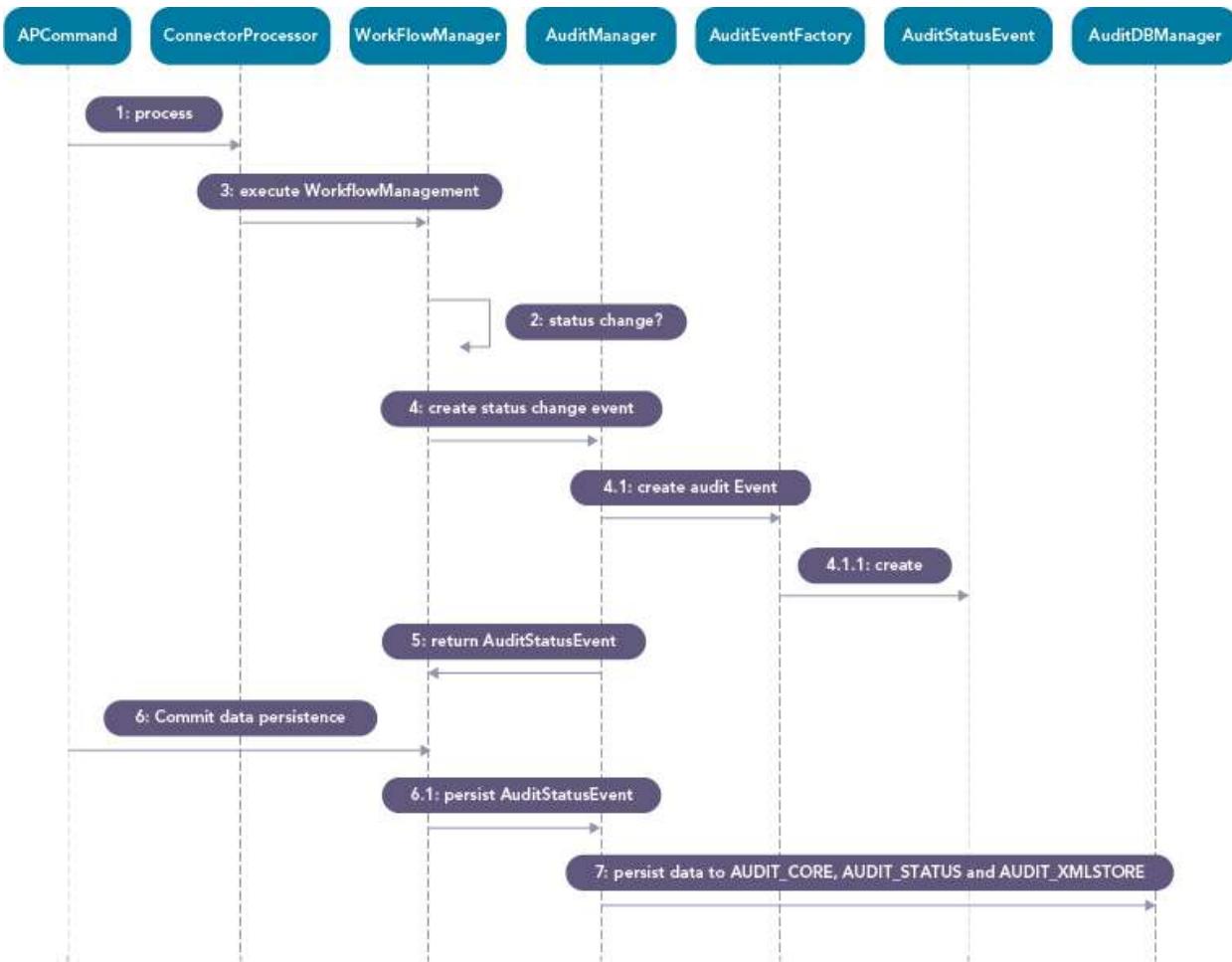
If a rule is executed under a non-TVR scenario, auditing events are created as soon as the rule messages are generated. Under TVR, all rule messages are cached until all rules are expected. `TransactionValidationManager` then collects all the messages for all the rules on all pages and creates one auditing event in the `AUDIT_CORE` table.

The following activity diagram demonstrates the workflow under TVR and non-TVR scenarios:



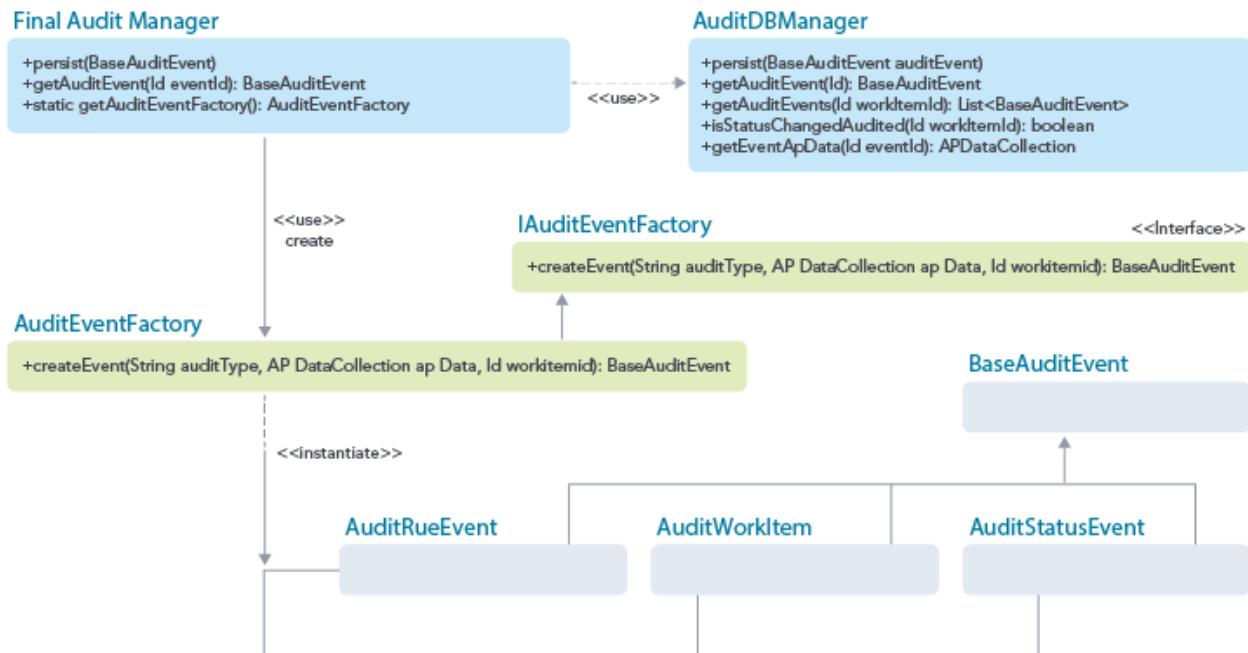
## Status Changes

Status changes are executed through **WorkFlowManager**. The following sequence diagram shows how status change events are tracked. If the work item status is changed, an auditing event for status change is created:



Work item status changes can be performed with custom codes in applications. Under these circumstances, the auditing records must also be created through custom codes.

Events and snapshots of ACORD XML for status changes, service call data and data modification are managed through **AuditManager**, as shown in the following class diagram:



The following are some sample status change records:

Results						
	audit_id	audit_type	workitem_id	create_time	created_by	user_group_id
1	92	STATUS	1469	2011-12-13 10:49:55.517	agent	200
2	95	STATUS	1469	2011-12-13 10:51:46.220	agent	200
3	100	STATUS	1469	2011-12-13 12:37:02.093	agent	200
4	158	STATUS	1469	2011-12-28 07:56:26.603	agent	200
5	163	STATUS	1469	2011-12-28 07:56:54.773	agent	200
6	167	STATUS	1469	2011-12-28 07:58:22.167	agent	200
7	300	STATUS	1469	2012-01-12 12:15:25.490	agent	200

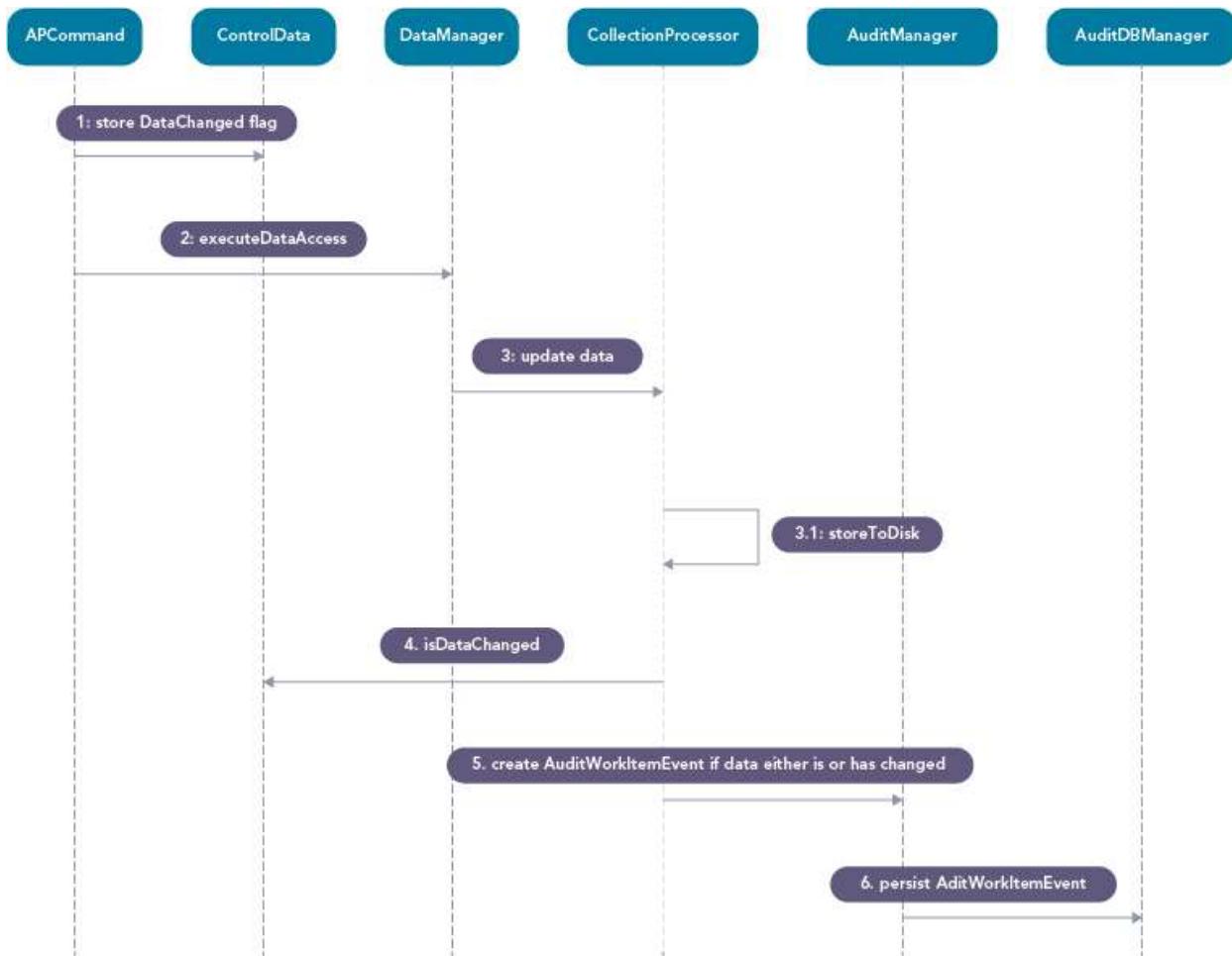
  

Messages		
	audit_id	old_status
1	92	15
2	95	15
3	100	15
4	158	15
5	163	15
6	167	15
7	300	15

## Data Modifications

A hidden form field, **DATA\_CHANGED**, is included in **dynamicape.jsp** to carry a flag to indicate whether there are any data changes on the page. When data on a page is submitted, **JavaScript Page.hasUnsavedChanges()** is invoked to determine the data changes. The service side uses this flag to evaluate the data modification.

- If **DATA\_CHANGED=1**, data on the page is modified. Thus, a data modification event is registered and the ACORD XML is persisted into **AUDIT\_XMLSTORE** if the Timeline feature is supported.
- If pages are custom, application developers can write custom scripts to determine the data change and set the corresponding value for **DATA\_CHANGED**.
- If data modifications are not detected, data modification events are not created. The following is the sequence diagram for the data modification event:



The following are some sample data modification records:

	Results	Messages
1	audit_id	audit_type
1	77	DATA_MODIFIED
2	89	DATA_MODIFIED
3	91	DATA_MODIFIED
4	93	DATA_MODIFIED
5	98	DATA_MODIFIED
6	156	DATA_MODIFIED
7	161	DATA_MODIFIED
8	165	DATA_MODIFIED
	workitem_id	
1	1469	1469
2	1469	1469
3	1469	1469
4	1469	1469
5	1469	1469
6	1469	1469
7	1469	1469
8	1469	1469
	create_time	
1	2011-12-12 10:30:36.107	2011-12-12 10:30:36.107
2	2011-12-12 10:39:59.137	2011-12-12 10:39:59.137
3	2011-12-13 10:49:55.007	2011-12-13 10:49:55.007
4	2011-12-13 10:51:46.107	2011-12-13 10:51:46.107
5	2011-12-13 12:37:01.867	2011-12-13 12:37:01.867
6	2011-12-28 07:56:26.543	2011-12-28 07:56:26.543
7	2011-12-28 07:56:54.757	2011-12-28 07:56:54.757
8	2011-12-28 07:58:21.500	2011-12-28 07:58:21.500
	created_by	
1	agent	agent
2	agent	agent
3	agent	agent
4	agent	agent
5	agent	agent
6	agent	agent
7	agent	agent
8	agent	agent
	user_group_id	
1	200	Example Agency
2	200	Example Agency
3	200	Example Agency
4	200	Example Agency
5	200	Example Agency
6	200	Example Agency
7	200	Example Agency
8	200	Example Agency
	user_group_name	
1	Example Agency	Example Agency
2	Example Agency	Example Agency
3	Example Agency	Example Agency
4	Example Agency	Example Agency
5	Example Agency	Example Agency
6	Example Agency	Example Agency
7	Example Agency	Example Agency
8	Example Agency	Example Agency

	audit_id	page_id	page_name
1	77	generalInfo	Agency/Applicant Information
2	89	generalInfo	Agency/Applicant Information
3	91	generalInfo	Agency/Applicant Information
4	93	generalInfo	Agency/Applicant Information
5	98	generalInfo	Agency/Applicant Information
6	156	generalInfo	Agency/Applicant Information
7	161	generalInfo	Agency/Applicant Information
8	165	generalInfo	Agency/Applicant Information

## WorkItem Creation Events

For new business, renewal, endorsement and upload work items, an auditing event is created right after the work item is created in `WorkItemManager.createWorkItem()`. The corresponding ACORD XML is also stored in the `AUDIT_XMLSTORE` table if the Timeline feature is supported.

## Service Call Data

The service call data, such as rating data, is requested for collection. The service call data is usually collected via connectors. The rating data can be stored in ACORD XML.

The `updateServiceData` connector instruction is added for service call auditing events. If a rating connector runs and a processing condition is met, the instruction for the service call can be processed.

### Example

```
<connector type="custom" id="ServiceDataManager" className="com.agencyport.shared.connector.CommAutoConnectorManager" title="ServiceDataManager"> <executeWhen userAction="Continue"/></connector><instruction type=" updateServiceData" title="Quote Data" id="ratingServiceData"> <when connector="ServiceDataManager" /> </instruction>
```

The mechanism of policy change summarizer is used to differentiate the service data change; however, service data is usually not defined in the transaction definitions and, therefore, is not summarized by the policy change summarizer. To use the existing policy change mechanism, the following changes are implemented:

1. Expand the transaction definition schema for page element to add a new `serviceData` type:

```
<xs:attribute name="type" use="required"><xs:simpleType> <xs:restriction base="xs:string"><xs:enumeration value="fieldset"/><xs:enumeration value="questionnaire"/><xs:enumeration value="roster"/><xs:enumeration value="script"/><xs:enumeration value="tips"/><xs:enumeration value="worksheet"/><xs:enumeration value="serviceData"/><xs:enumeration value="index"/></xs:enumeration> </xs:restriction> </xs:simpleType> </xs:attribute>
```

Define the service call data into `serviceData` page elements. Service call data can now be stored into `serviceData` page elements. The `ServiceDataPageElement` object is created for the `serviceData` page element.

2. Create a page element for the new type of `serviceData` page element so that the field elements within the `serviceData` page elements do not print on pages:

```
public class ServiceDataPageElement extends FieldsetPageElement { public ServiceDataPageElement(Element pageElementMetaData, Page owningPage){ super(pageElementMetaData, owningPage); interestLevel = TransactionDefinitionProvider.INTEREST_LEVEL_EXCLUDE; } public String printHTML(){ return ""; //do not print any } }
```

3. Change `BasePageElement.supportedByJavaScriptBasedFieldSet()` so that no JavaScript print out on pages:

```
public boolean supportedByJavaScriptBasedFieldSet(){ if (getUniqueId() == null) return false; boolean isNotSupported = (this instanceof PageScriptPageElement || this instanceof RosterPageElement || this instanceof IndexPageElement || this instanceof ServiceDataPageElement); return !isNotSupported; }
```

Since the elements and related JavaScript within `serviceData` page elements are not rendered on a page, the service call data in ACORD XML should remain intact when data is cleaned up during intrapage processing.

4. Modify `PolicyChangeSummarizer` to include `ServiceData`:

```
PolicyChangeSummarizer.getFieldLevelChangeDetails(){ List<FieldLevelChangeDetail> modInfoDetailsList = new LinkedList<FieldLevelChangeDetail>(); List<BaseElement> fieldList = page.getAllFields(FIELD_FILTER); }
```

`Page.getAllFields(IFieldFilter)` is a method; `IFieldFilter` is an interface.

```
public List<BaseElement> getAllFields(IFieldFilter fieldFilter) { List<BaseElement> fields = new ArrayList<BaseElement>(); for (BasePageElement basePageElement: pageContent) { if (null != fieldFilter && !fieldFilter.acceptPageElement(basePageElement.getType())){ continue; } List<BaseElement> baseElements = basePageElement.getContent(); for (int j = 0; j < baseElements.size(); j++) { BaseElement baseElement = baseElements.get(j); if (null != fieldFilter && !fieldFilter.acceptFieldElement(baseElement)){ continue; } fields.add(baseElement); } } return fields; }
```

`PolicyChangeSummarizer` summarizes all fields on pages, including service call data.

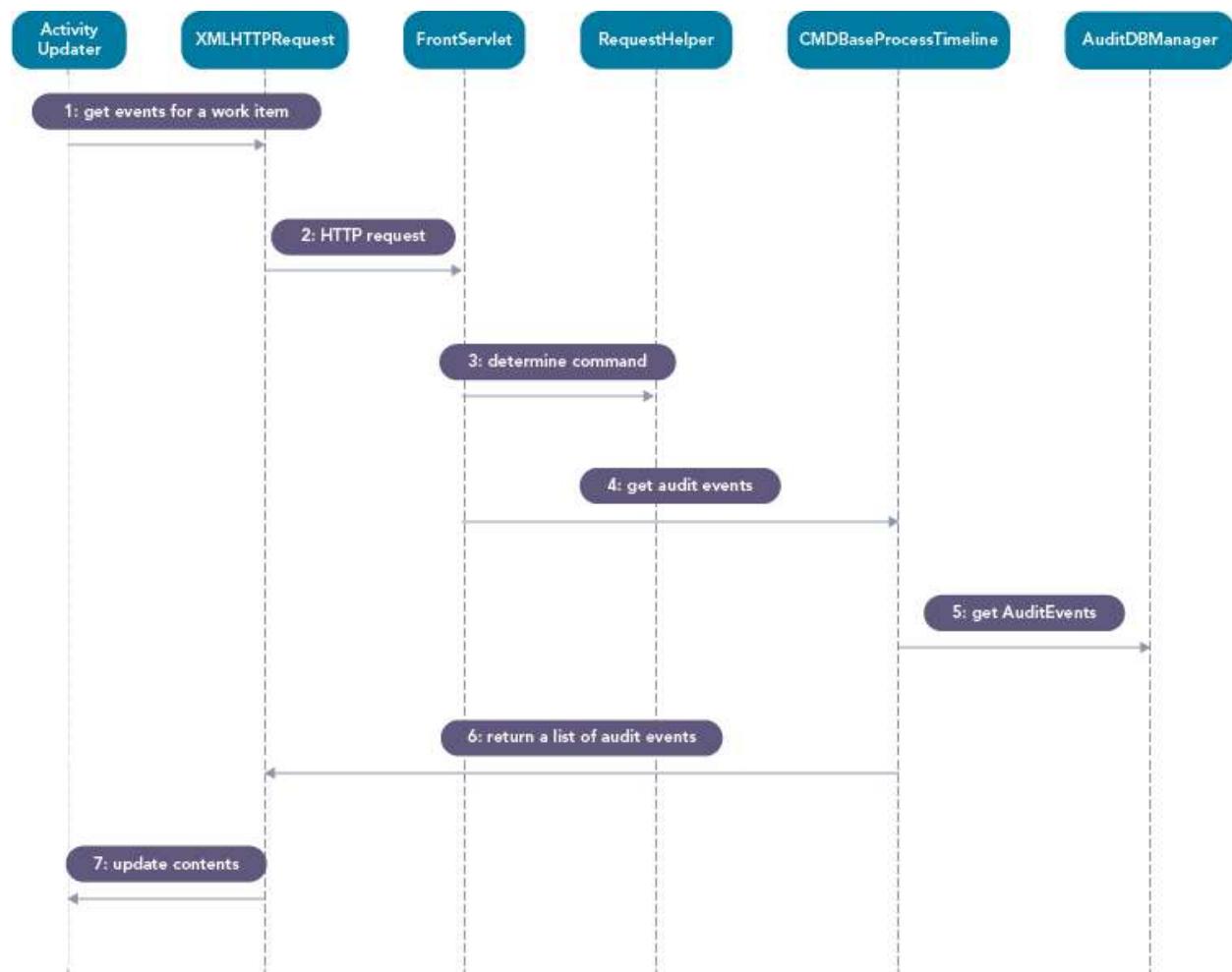
## Work Item Timeline View

The work item timeline view is updated via Ajax requests.

### Event List

When Timeline view displays, a list of audit events for the work item is retrieved and presented on a page via Ajax request.

FrontServlet is the entry point for any calls coming in from the client. RequestHelper uses the Ajax request parameters to determine the command class to call by using the `specialRecognizedActionsMap` entry created Work Item Timeline process command, as shown in the following sequence diagram:

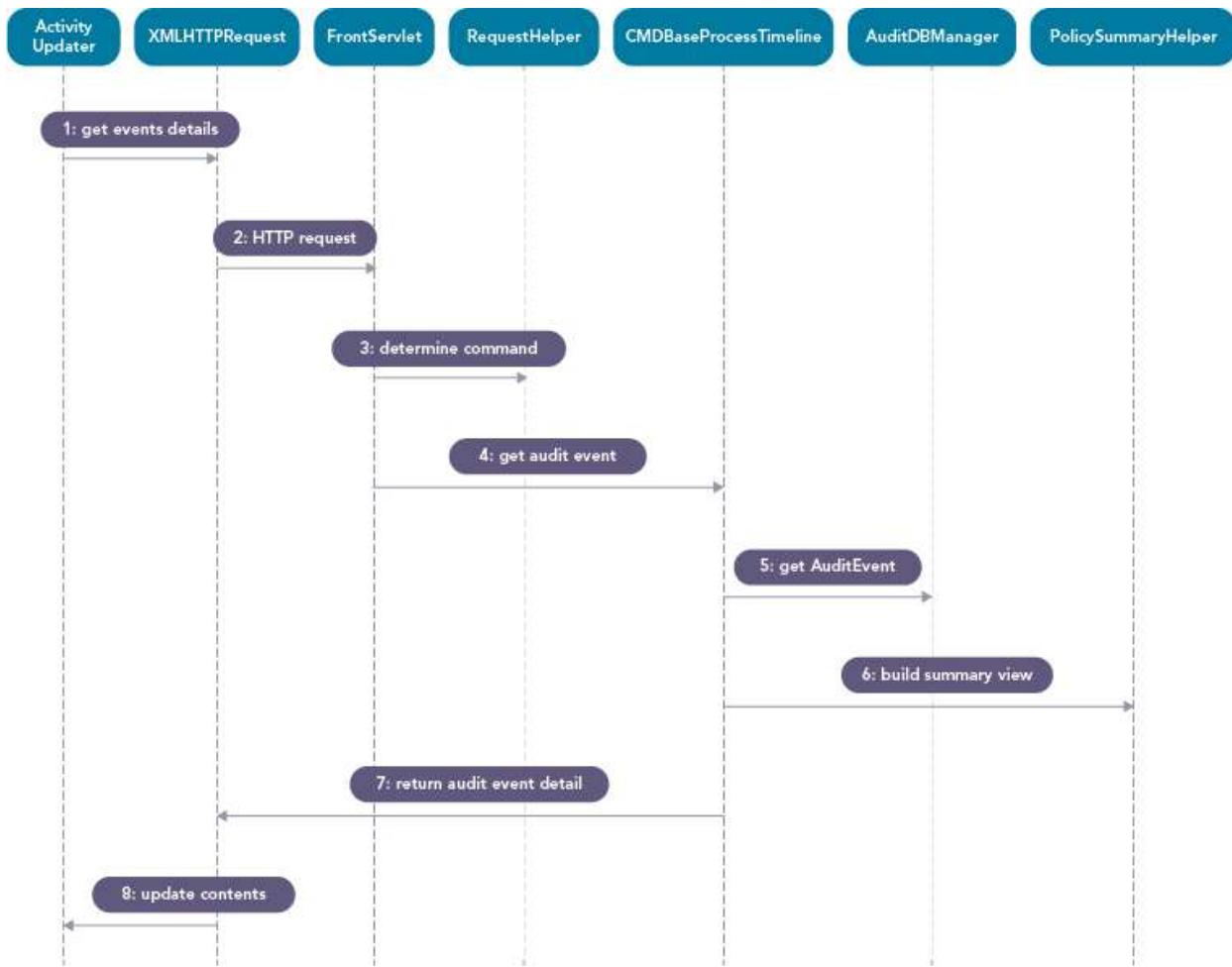


CMDBaseProcessTimeline relies on AuditDBManager to query the database to build a list of audit events. CMDBaseProcessTimeline is defined as follows:

CMDBaseProcessTimeline
<pre>+process() #prepareEventList() #prepareAuditEvent() #prepareSummaryReview() #prepareDifferenceView() #executeCustomRuleMessagePreparation(List RuleEventData) #executeCustomSummaryViewPreparation(List SummaryData)</pre>

## Audit Event Detail Report

When users select an event, the details of the event are updated.



After an audit event is retrieved, the event-specific details are formulated.

#### Rule Event

A list of RuleEventData objects are instantiated and rendered onto a JSP, then send the DOM element back to an Ajax request. RuleEventData controls all the rule triggering messages. If a user does not have permission to view all of the messages, these messages can be removed by implementing `CMDBaseProcessTimeline.executeCustomRuleMessagePreparation()`:

```
/** * Allow custom processing of the rule message. Applications can override this to determine what rule messages should be displayed. *
@param ruleData a list of RuleEventData */ protected void executeCustomRuleMessagePreparation(List<RuleEventData> ruleData) { return; }
```

The following is a sample of a rule triggering event modal:

The screenshot shows a 'Work Item History' page with a 'Event Summary' section. It details three events:

- 1 Work Item Created** on 09-05-2014 09:12:50
- 2 Rule Triggered** on 09-05-2014 09:13:03
- 3 Rule Triggered** on 09-05-2014 09:14:03

The 'Event Summary' for the second event states: "A rule was triggered by the values in this work item at the time". Below this, it shows the event details: When: 09-05-2014 09:13:53, By: R.G. A table then lists the rule message and severity for the third event.

Page	Rule Message	Severity
Policy Information	Only MA,RUL, MD and FL are currently supported. Please select a supported state.	error

## Work Item Creation

A list of `SummaryData` objects are built with the ACORD XML associated with the work item creation event. These objects are rendered onto a JSP before the response is sent back to the Ajax request.

`CDMBaseProcessTimeline.executeCustomSummaryViewPreparation()` is added to allow custom manipulation of the policy summary data before they are presented to users:

```
/** * Allow custom processing of the summary data. Applications can override this to determine what summary data *should be shown.*
@param summaryData a list of SummaryData */protected void executeCustomSummaryViewPreparation(List<SummaryData> summaryData)
{ return; }
```

The following are sample work item creation event modals:

### Manual

The screenshot shows a 'Work Item History' page with a 'Event Summary' section. It details one event:

- 1 Work Item Created** on 09-09-2014 08:19:18

The 'Event Summary' for this event states: "The work item was created". Below this, it shows the event details: When: 09-09-2014 08:19:16, By: Jami Della.

### Upload

The screenshot shows a 'Work Item History' page with a 'Event Summary' section. It details one event:

- 1 Work Item Created** on 09-09-2014 03:42:49

The 'Event Summary' for this event states: "About this Upload". Below this, it shows the event details: When: 09-09-2014 03:42:49, By: Great Salesman. A table then lists the system, version, and vendor information.

System	Version	Vendor
Agencyport Software Turnstile	1.0	Agencyport Software

The work item was created

## Data Modification

A difference summary between the ACORD XML associated with the current audit event and the ACORD XML associated with an audit event right before the current event is built via `PolicySummaryHelper`. The difference summary data is rendered onto a JSP and sent back to Ajax request. The summary data can also be further filtered out with `CDMBaseProcessTimeline.executeCustomSummaryViewPreparation()`.

## Status Change

If the status change is the very first status change for the work item, the ACORD XML associated with the status change is used to build the policy summary data.

If the status change is not the very first status change for the work item, a difference summary between the ACORD XML associated with the current status event and the ACORD XML associated with an audit status change event right before the current event is built via `PolicySummaryHelper`.

The following is a sample status change event modal:

The screenshot shows a modal window titled "Work Item History". It contains two entries in a table format:

	Event Summary	Snapshot
1	Work Item Created 09-09-2014 10:24:45	
2	Status Changed by Great Salesman 09-09-2014 10:26:43	

For the second entry (Status Changed), there are two rows of details:

Previous Status	New Status
INPROGRESS	REFER

## Service Call Data

Policy summary data is processed using the same algorithm as data modification.

## Access to Work Item Timeline

All users and roles have permission to access the Timeline feature. As an option, you can enforce the Timeline feature during permission/role/group implementation.

# My Defaults

The My Defaults feature allows users to save common transaction information to reuse when creating similar transactions. Out of the box, this feature is turned on for Workers Comp and Commercial Auto line of business transactions.

To turn this feature on for any other transaction, set `supportsDefaults=true` in the TDF.

## Example

The following is an example of the My Defaults feature turned on for a New Business Commercial Auto transaction.

```
<transaction
 title ="Commercial Auto"
 id ="commlAuto"
 target ="CommlAutoPolicyQuoteInqRq"
 lob ="AUTOB"
 type ="new_business"
 allowEmptyNode = "false"
 enableAllMenuEntries = "false"
 fieldValidations = "enabled"
 summaryPageId ="policySummary"
 prevalidate = "true"
 mappingId ="autob"
 workItemAssistantId = "workitemassistant"
 supportsIPDTRDynamicContent="true"
 supportsDefaults = "true"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://reference.agencyport.com/schemas/5.1/definition/transactionDefinitionFile.xsd"
 autoMaintainIdAttributes="true">
 <description>Multi state New Business transaction for Commercial Auto Line of Business</description>
 <!-- General Information -->
 <importPage id="AUTOB.generalInfo" fileName="commlAutoCommons.xml"/>
```

Once the `supportsDefaults` parameter is set to true, the My Defaults drop-down and Save to Defaults action option displays on all of the transaction's pages. Refer to the [My Defaults](#) topic in the User Workflow section for more information.

## Save to Defaults

When the Save to Defaults option is selected, the system is instructed to take the current work item in context (in its current state) and save it as a prototype document (ACORD XML).

## Apply Defaults

The Apply Defaults option allows users to select one of the prototype documents (ACORD XML) saved via the Save to Defaults option and apply it to the current work item in context.

## Available Fields

The fields available to use with the My Defaults feature are established in the TDF. To add a field to the list of available fields, you must add a `useSubstitute` correction type to the field. Refer to the [Transaction Definition/Page Library](#) topic for more information on the components of TDFs.

## Example

```
<fieldElement type="selectlist" id="Producer.GeneralPartyInfo.Addr.StateProvCd" label="State" required="true" uniqueId="producerState">
 <optionList reader="xmlreader" source="codeListRef.xml" target="selectOne" />
 <optionList reader="xmlreader" source="codeListRef.xml" target="stateCd" />
 <correction type="useSubstitute" condition="missing" source="prototype" message="A substitute value was copied from user defaults" />
 <correction type="bestMatch" condition="incorrect" liberty="moderate" message="This value was changed to match an item in the list." />
</fieldElement>
```

All fields configured as `useSubstitute` corrections in the TDF display when the Available Fields option is selected for that particular transaction.

### Fields for which Default Values Can Be Set

X

Page	Field
Agency/Applicant Information	State
Agency/Applicant Information	Tax Id Type
Agency/Applicant Information	Legal Entity
Agency/Applicant Information	Billing Method
Agency/Applicant Information	Direct Bill Pay Plan
Agency/Applicant Information	Auditable?
Agency/Applicant Information	State
Locations	State
Locations	Located
Rating Classifications	Location
Rating Classifications	Class Code
Questions & TextArea	Billing Method
General Information	Business in Trust

# Work List

A work list is used to display a large amount of data to a user in a tabular format. Optionally, you can configure the work list to allow a user to take action on member items within the list. In AgencyPortal5.2, there are two ways to view this data: card or classic view.

The following is an example of the card view work list:

The screenshot shows a card-based work list interface. At the top, there is a header bar with buttons for '81 Work Item(s)', 'LOB', 'Status', 'Transaction', 'Sort By', 'More Options', and 'My Filters'. Below the header is a search bar labeled 'Search for a customer by name (3 character minimum)'. The main area contains five cards, each representing a work item. The first card, 'Add New Work Item', has a plus sign icon and a dashed border. The other four cards are for specific customers: 'Yummy Food, Inc %' (ID 1228), 'Penny Smith %' (ID 1227), 'Oak Craft, Inc. %' (ID 1216), and 'Trucks R Us %' (ID 1201). Each card displays the customer's name, ID, effective date, premium, status, and transaction type. A progress bar on the right side of each card indicates completion percentage (e.g., 91%).

The following is an example of the classic view work list:

The screenshot shows a classic table-based work list interface. At the top, there is a header bar with buttons for '81 Work Item(s)', 'LOB', 'Status', 'Transaction', 'Sort By', 'More Options', and 'My Filters'. Below the header is a search bar labeled 'Search for a customer by name (3 character minimum)'. The main area features a table with columns: ID, LOB, Name, Status, Updated, Effective, Premium, Transaction Type, and Complete %. The table lists five work items corresponding to the ones shown in the card view: Yummy Food, Inc %, Penny Smith %, Oak Craft, Inc. %, Trucks R Us %, and Trucks R Us %. Each row provides detailed information such as the customer's name, ID, effective date, premium, status, and transaction type.

ID	LOB	Name	Status	Updated	Effective	Premium	Transaction Type	Complete %
1228	Commercial Auto	Yummy Food, Inc %	In-Progress	04-29-2015 06:10:02	04-28-2015	0	Full Application	91
1227	Personal Auto	Penny Smith %	In-Progress	04-29-2015 06:07:59	04-28-2015	0	Full Application	91
1216	Workers Comp	Oak Craft, Inc. %	In-Progress	04-29-2015 05:31:58	05-31-2013	0	Full Application	10
1201	Workers Comp	Trucks R Us	In-Progress	04-28-2015 12:01:11	04-27-2015	0	Full Application	91
1210	Workers Comp	Trucks R Us %	In-Progress	04-28-2015 10:58:31	04-27-2015	0	Full Application	91

An example of a work list is the work in progress queue. The work list is also used to display audit events. In the case of the work in progress work list, a user can take actions, such as opening or copying a work item; however, for audit items, users can only view the items as a list. The work list supports both of these scenarios.

Work list configuration allows the implementer to specify columns within the list and actions (if any) associated with the member items. Implementing a work list provides all the conveniences of the base implementation, including pagination, sorting and filtering of list items. Even though work lists can co-exist with other items on a page, they are not designed to be used within a transaction.

Refer to the [Work List Configuration and Customization](#) topic for more information on configuring and customizing work lists.

## Work in Progress Feature

The work list feature provides information on transactions that were previously entered into the application. These lists can be configured to display work items based on the usertype. An underwriter may have the ability to view all work items entered, whereas an agent would only be able to see work items that he or she entered. Refer to the [Work List](#) topic for more information on how the work list feature works and displays for a user.

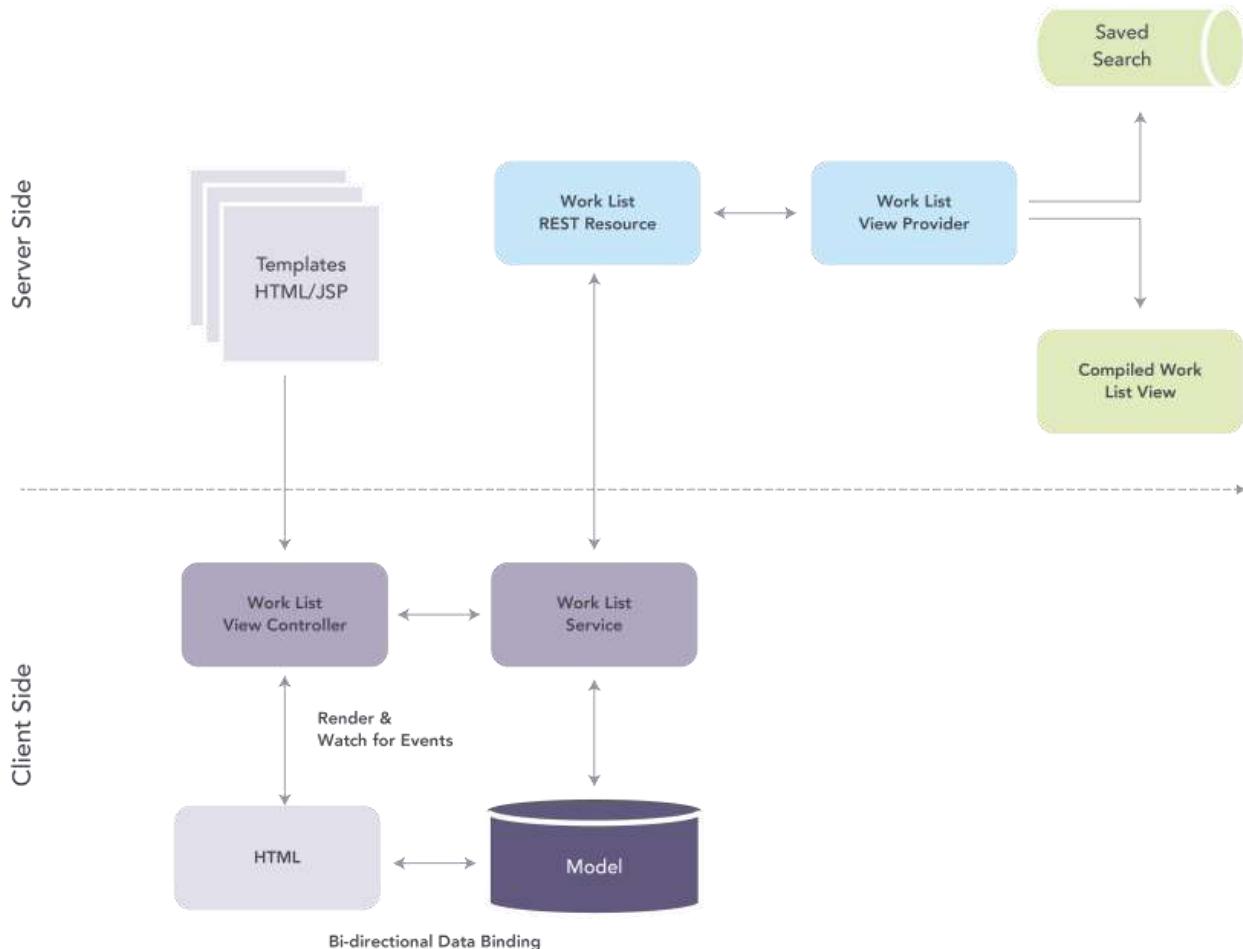
### Work in Progress Columns

The following columns are included in the classic view work list out of the box:

WIP Column	Description
ID	The number systematically assigned when a user continues from the first page of the application.
LOB	The line of business.
Name	The name of the applicant or insured.
Status	The status of the work item. Examples include: In-Progress, Referred, Approved, Rejected, Declined.
Updated	The time and date the work item was last updated.
Effective	The time and date the work item goes into effect.
Premium	The premium amount returned when a work item is submitted, if applicable.
Transaction Type	The type of transaction. Examples include: Quick Quote, New Business, Endorsement, Renewal
Complete	The percent complete calculation percentage for the work item.

# Work List Architecture

The work list in AgencyPortal 5.2 provides an efficient, simplified configuration model for work list views and their associated Solr index mappings. This helps ease the burden of work list operations that a project team must carry out to enhance or customize various work list instances. Most of the work list functionality is driven with configuration instead of code while still accommodating the alteration of default behavior through extension or complete interface implementation.



## Server Side Architecture

This section details the components that make up the server side component of the work list.

### REST API

A set of data services based on REST (REpresentational State Transfer) is used to serve up work list view definitions, including various list views (card versus tabular) with their respective fields, filters, sorting information and additional query fields. This same API also paves the way for serving up work list view structures to other channels and devices, such as consumer, self-service and non-browser based clients.

### XML Based Configuration

XML based configuration structures are included to represent how work item/account data values map to an external index repository, such as Solr. This provides a data driven approach to map data to an external search index repository. XML based configuration structures are also included to easily model concise work list view definitions during configuration. A work list view definition includes the definitions to various list views, filters, sorts and additional query fields, and refers to fields from the index mapping.

## Resource Types

A product configuration artifact type (resource type), work list view definition, supports work list rendering. The resource type has its own unique product resource provider and launches during application bootstrap time. The resource provider processes the XML artifacts registered in the application's product database for its respective type into its own specific compiled in-memory representation. These representations are used at runtime rather than its raw JDOM elements.

The following illustrates the work list view definition resource type, including its resource providers and respective compiled runtime representations:



## Resource Providers

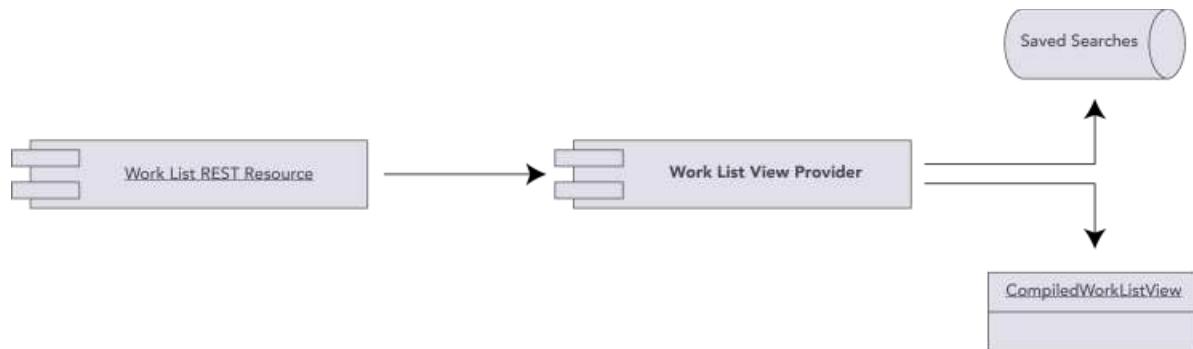
### Work List View Provider Interface

The work list provider interface prescribes the standard API contract for the various CRUD operations for work list metadata entities. This includes those operations for work list filters, sort criterion, additional query fields, list views, saved searches, etc. It is unlikely that applications will ever need to implement this interface; however, a factory mechanism allows an application to create its own implementation of this interface if the need arises to develop an implementation that is not based on the XML metadata structures native to the framework. The work item factory class will honor an application implementation of this interface under the `application.work_list_view_provider_classname` application property.

Each application can have one (and only one) work list view provider. If an application needs to support work list view operations that embrace both the default work list view metadata structures and custom data structures, then the application will need to extend the framework's `WorklistViewProvider` class and dynamically examine the parameters that are supplied on each method and react accordingly by delegating to the super's implementation or relying on its own.

### Work List View Provider

The work list view provider is the framework's default implementation of the work list view provider interface. It provides access to filters, sort information and work list view information that are configured using the framework's work list XML product artifacts. of a more generalized interface. It also provides the CRUD (create, read, update and delete) operations for saved searches. A saved search is basically a work list view that has been persisted to the database. It negotiates all work list access between the REST resource class, the compiled work list view and any saved searches. It is the extension point through which applications can modify or alter basic work list view definition behavior.



### Work List Search Provider Interface

The work list search provider interface prescribes the standard API contract for performing work list queries. Applications may need to override the standard Solr based work list search provider. The most common scenario relates to how security filtering enters into the work list query. In this case, applications should extend the Solr based work list search provider class (`SolrWorkListSearchProvider`) and register this class name in the `application.work_list_search_provider_classname` application property.

If an application needs to source a work list from another source than a Solr index then they will need to create its own implementation of this interface and register that implementation under the `application.work_list_search_provider_classname` application property.

If an application needs to serve up both Solr based work lists and non-Solr based work lists, then the best approach is to subclass the `SearchIndexFactory` class and register that class name under the `application.searchindex_factory_classname` application property. This will give the application the ultimate flexibility on which work list search provider implementation support the various work list needs of the application.

#### Solr Work List Search Provider

The work list search provider is the framework's default implementation of the work list search provider interface for rendering work lists that are based on Solr indices. It accepts work list view instances that have specific filters, sorts and engaged queries. It also prepares a Solr query statement and then executes the query against the specific Solr index. It takes the results from the Solr query and compiles them into a generalized result set. It is the extension point through which applications can modify or alter basic search index query functionality, including any security filtering specific to the application.



Compiled Runtime Representations

#### Compiled Work List View

The compiled work list view serves the following purposes:

- Serves as the compilation runtime unit for index mapping structures conforming to the XML schema. Refer to the [Work List Definition Schema](#) for more information.
- Links to one index mapping resource. This link allows work list views to refer to field definitions in the index mapping resource without having to replicate all of the properties for each field.
- Delineates the various elements of a work list, including:
  - **Views** - Each view is a collection of filters, sort information, additional query fields and zero or more list views. View elements can have no permissions, which means that any security role will have access to that view. Views can also have one or more permissions associated with it. The framework will allow users with just one of those permissions to access that view.
  - **List View** - Each list view defines which fields are visible to the user, the order in which those fields display and the type of list that displays: card, tabular or custom. The fields included in the list view are references to the real field definition in the index mapping resource.
  - **Filters** - Each filter has its own type, name and localization information. Each filter refers to its respective field in the index mapping resource.
  - **Sorts** - Each sorting information instance contains a default sort criterion clause that is used as the initial sort for any search index query. Each instance is made up of zero or more field references. Ascending/descending collation is supported.
  - **Additional Query Fields** - Additional query fields can be defined, which drive additional filtering when the search index query is rendered. Some query fields are fixed and some are interactive. The fixed queries provide a mechanism for establishing unconditional, additional filtering, such as filtering out work items in a deleted status. Interactive queries may have backing TDF constructs to support the data collection of those facets.

#### Communication and Invocation

##### Salient APIs

This section outlines the salient provider interfaces that provide the bulk of this functionality.

###### `com.agencyport.api.worklist.pojo.WorkListView`

This construct is used to convey work list view information back to the client. It is serializable/de-serializable as JSON and XML. It is composed of the filtering, sorting, query field, list views, the user's current search information and their saved search information for a given work list view name.

All of the various entities that comprise a `WorkListView` instance have Boolean selected flags that govern which entities are active for a given search index query or saved search instance. Application groups can extend this class if custom information is necessary to support additional business requirements. This is achieved by registering the class name of their extension under the `application.worklist_view_classname` application property.

##### Work List View Definition Logical Data Model

The following diagram depicts what the work list view is composed of:

## Work List View



Provider Interface	Description
<b>com.agencyport.worklist.vie.w.IWorkListViewProvider</b>	<p>This interface describes the contract for accessing work list view metadata instances (both saved and default configurations). The framework's work list REST resource classes utilize this interface for all of the functions that need to operate on work list view instances.</p> <p>This interface serves as the mechanism for negotiating the CRUD operations upon a work list view meta data instance. These provider instances are brought to life by calling the <code>WorkItemFactory.createWorkListProvider</code> method. This factory method creates a <code>com.agencyport.worklist.view.WorklistProvider</code> instance by default unless the application has registered a custom extension of this class or a custom implementation of the base interface under the <code>application.work_list_view_provider_classname</code> application property.</p> <p>If an application requires a custom filtering option, then the application should extend the default implementation <code>com.agencyport.worklist.view.WorkListViewProvider</code> class. If the application is required to create a work list experience or has a work list whose structure is represented by metadata other than those based on the XML metadata structures native to the framework, then the application must either extend the default implementation.</p> <p>Each application can have one, and only one, work list view provider. Therefore, if an application needs to support work list view operations that embrace both the default XML work list view metadata structures and custom data structures, then the application will need to extend the framework's <code>WorkListViewProvider</code> class and dynamically examine the parameters that are supplied on each method and react according by delegating to the super's implementation or relying on its own. The most notable methods of this interface are as follows:</p> <ul style="list-style-type: none"> <li>• <code>getView()</code> - This method delivers the view. If there is a supported saved search information record, it will be merged in.</li> <li>• <code>getSavedSearches()</code> - This method retrieves all of the saved search instances for a given user/view name combination.</li> <li>• <code>saveView()</code> - This method saves the work list view instance to the database.</li> <li>• <code>deleteView()</code> - This method deletes a saved work list view from the database.</li> </ul> <p>Refer to the <a href="#">JavaDoc</a> for more information.</p>
<b>com.agencyport.searchindex.IQueryResults</b>	<p>This interface is designed to unify a set of query results as a non-vendor specific data structure. The following are notable methods:</p> <ul style="list-style-type: none"> <li>• <code>getQueryResults()</code> - This method returns a query result instance for the given index.</li> <li>• <code>getNumberOfHits()</code> - This method returns the total number of hits in the search index for this query.</li> <li>• <code>getStartingRecordNumber()</code> - This method returns the starting record number represented in this query.</li> <li>• <code>getNumberofResults()</code> - This method returns the number of result records in this query result instance.</li> </ul> <p>Refer to the <a href="#">JavaDoc</a> for more information.</p>
<b>com.agencyport.searchindex.IWorkListSearchProvider</b>	<p>The framework's work list REST resource classes utilize this interface for all of the functions that need to perform search index queries. This interface prescribes the standard API contract for performing work list queries.</p> <p>Applications may need to override the standard Solr based work list search provider. The most common scenario that drives this need relates to how security filtering enters into the work list query. In this case, applications should extend the Solr based work list search provider class, <code>com.agencyport.searchindex.solr.SolrWorkListSearchProvider</code>, and register this class name as the <code>application.work_list_search_provider_classname</code> application property.</p>

Provider Interface	Description
	<ul style="list-style-type: none"> <li>If, however, an application needs to source a work list from another source than a Solr index, then they will need to create its own implementation of this interface and register that implementation under the <code>application.work_list_search_provider_classname</code> application property.</li> <li>If an application needs to serve up both Solr based work lists and non Solr based work lists, then the best approach is to subclass the <code>SearchIndexFactory</code> class and register that class name under the <code>application.searchindex_factory_classname</code> application property.</li> </ul> <p>This will give the application the ultimate flexibility over the factory based creation mechanism of work list search provider implementations that support the various work list needs of the application. The notable methods are:</p> <ul style="list-style-type: none"> <li><code>initialize()</code> - This method initializes the state on the worklist search provider and is called before and in preparation of the <code>search()</code> method.</li> <li><code>search()</code> - This method searches an index based on the filters, sort information and query information initialized from calling the <code>initialize()</code> method.</li> </ul> <p>Refer to the <a href="#">JavaDoc</a> for more information.</p>

## REST Services

Two REST resources are introduced to support the work list view and search index query operations. One for work item based interactions and one for account based interactions. All methods support both XML and JSON as a format. Client applications must pass which format they desire in the Accepts HTTP header and designate the format of input requests via the Content-Type HTTP header when passing data into the server. The following functions are supported:

### Get Account Work List View

HTTP Verb: GET

Retrieves an account work list view image from the server.

URI: /api/worklists/accounts/views/{view\_name}

Parameter Name	Type	Required	Default Value	Comments
view_name	Path	True	None	The view name links to the <code>view_name</code> of the work list view definition resource.
saved_search_id	Query	False	None	If present, then a saved search image will be merged into the configuration instance for this work list view.

### Get LOB Work Item Work List View

HTTP Verb: GET

Retrieves an LOB work item work list view image from the server.

URI: /api/worklists/workitems/views/{view\_name}

Parameter Name	Type	Required	Default Value	Comments
view_name	Path	True	None	The view name links to the <code>view_name</code> of the work list view definition resource.

<b>Parameter Name</b>	<b>Type</b>	<b>Required</b>	<b>Default Value</b>	<b>Comments</b>
saved_search_id	Query	False	None	If present, then a saved search image will be merged into the configuration instance for this work list view.

#### Save Account Work List View

HTTP Verb: PUT

Saves an account work list view image to the server. An updated copy of the saved work list view is returned to the caller.

URI: /api/worklists/accounts/views/{view\_name}

<b>Parameter Name</b>	<b>Type</b>	<b>Required</b>	<b>Default Value</b>	<b>Comments</b>
view_name	Path	True	None	The view name links to the view_name of the work list view definition resource.
Work list view	Request body	True	None	The current search name located at workListView/savedSearchInfo/currentSearch@name needs to be initialized by the client for named saved searches. For the default unnamed search, the client should set workListView/savedSearchInfo/currentSearch@id to -1.

#### Save Work Item Work List View

HTTP Verb: PUT

Saves an LOB work item work list view image to the server. An updated copy of the saved work list view is returned to the caller.

URI: /api/worklists/workitems/views/{view\_name}

<b>Parameter Name</b>	<b>Type</b>	<b>Required</b>	<b>Default Value</b>	<b>Comments</b>
view_name	Path	True	None	The view name links to the view_name of the work list view definition resource.
Work list view	Request body	True	None	The current search name located at workListView/savedSearchInfo/currentSearch@name needs to be initialized by the client for named saved searches. For the default unnamed search, the client should set the workListView/savedSearchInfo/currentSearch@id to -1.

#### Delete Account Work List View

HTTP Verb: DELETE

Deletes an account work list view image from the server.

URI: /api/worklists/accounts/views/{view\_name}

<b>Parameter Name</b>	<b>Type</b>	<b>Required</b>	<b>Default Value</b>	<b>Comments</b>
view_name	Path	True	None	The view name links to the view_name of the work list view definition resource.
saved_search_id	Query	True	None	The id of the saved search record.

#### Delete Work Item Work List View

##### HTTP Verb: DELETE

Deletes an account work list view image from the server.

URI: /api/worklists/workitems/views/{view\_name}

Parameter Name	Type	Required	Default Value	Comments
view_name	Path	True	None	The view name links to the view_name of the work list view definition resource.
saved_search_id	Query	True	None	The id of the saved search record

#### Query Account Index

##### HTTP Verb: POST

Submits a work list view to the server, runs the query against the corresponding search index and returns the query results.

URI: /api/worklists/accounts/views/{view\_name}

Parameter Name	Type	Required	Default Value	Comments
view_name	Path	True	None	The view name links to the view_name of the work list view definition resource.
Work list view	Request body	True	None	

#### Query Work Item Index

##### HTTP Verb: POST

Submits a work list view to the server, runs the query against the corresponding search index and returns the query results.

URI: /api/worklists/workitems/views/{view\_name}

Parameter Name	Type	Required	Default Value	Comments
view_name	Path	True	None	The view name links to the view_name of the work list view definition resource.
Work list view	Request body	True	None	

#### Upgrading from AgencyPortal 5.0

The impact on AgencyPortal 5.0 applications to upgrade to this work list architecture depends largely on the following:

- Project teams must understand how to leverage the new product artifacts to the extent necessary to support additional Solr index field and custom search criterion. The extend of this understanding depends on the scope of those customizations.
- Since most of the 5.0 JavaScript, JSP and supporting work list configuration artifacts supporting work list functions are retired, any customizations that project teams made to these artifacts under 5.0 must be retrofitted into the new 5.1 JSP/JavaScript artifacts. All of the client artifacts will be replaced with new constructs.

- Any additions/subtractions/alterations made to the 5.0 columns, filters, sorts, queries in the work list views from what was delivered out of the box must be retrofitted over to the new 5.1 work list view product configuration artifacts. This includes changes to styling, wording and other visual aspects.
- Angular is a paradigm shift from previous JavaScript libraries. Project teams must become competent in this discipline.
- Any extension to any of the following classes must be moved to the corresponding new extension point. The following is a table of 5.0 extension points, alongside their moral equivalent 5.1 point:

5.0 Extension Point	5.1 Extension Point	Description
ISolrManager.search	IWorkListSearchProvider.search	Application modifies search function
IWorkListSearchProvider.applySecurityFilter	Application needs to alter how security impacts search index searching	
IWorkItemFilteringManager and IWorkListMiscDataPopulator all methods	IWorkListViewProvider	Access to all filtering and sorting information has shifted to the work list view provider

- Any changes project teams have made to their 5.0 Solr index schemas will need to be retrofitted over to the new 5.1 Solr index schemas. Note that the account Solr schema was updated since there were many unnecessary fields being maintained.

Refer to the [Upgrade Guides](#) topic for more detailed information on upgrading to AgencyPortal 5.1 from AgencyPortal 5.0.

## Client-Side Architecture

5.1 introduced the use of an AngularJS framework and REST APIs to construct the client-side architecture of AgencyPortal. The following components have been re-worked to use this framework:

- work item list
- account list
- work item list that displays on the account details page
- recent quotes
- work item counts
- Solr batch updater

This new architecture allows us to:

- use templates instead of creating DOM elements in JavaScript or hardcoding HTML in JavaScript
- use bi-directional data binding to avoid manipulating DOM elements and event registration in JavaScript
- provide a data driven architecture to decouple service side rendering from client side rendering
- build reusable components
- write less code and provide more configurations

The following outlines the folder structure of this new framework:

- **Web Root**
  - **worklist**
    - **partials**
      - account-addnewcard.tpl.jsp
      - account-card.tpl.jsp
      - cardView.jsp
      - tabularView.jsp
      - worklist-addnewcard.tpl.jsp
      - worklist-card.tpl.jsp
    - **modals**
      - pleasewait.mdl.jsp
      - workitem-approveWorkitem.mdl.jsp
      - workitem-deleteWorkitem.mdl.jsp
      - workitem-linkWorkitem.mdl.jsp
      - workitem-moveWorkitem.mdl.jsp
      - workitem-success.mdl.jsp
  - **Web Source**
    - js

- **worklist**
  - worklist.addnewcard.drv.js
  - worklist.app.js
  - worklist.data.ctrl.js
  - worklist.search.ctrl.js
  - worklist.srv.js
  - worklist.workitemactions.mdl.js
  - worklist.workitemactions.srv.js
  - worklist.workitemdata.drv.js

The following is a reference to the naming conventions used in the work list files:

*.mdl	Modal	Deals with the modal dialog
*.ctrl	Controller	Controller for the UI
*.drv	Directive	Creates and manages reusable HTML components (e.g., card)
*.tpl	Template	Consists of a template used to render the UI
*.fltr	Filter	Consists of any custom Angular filters
*.srv	Service	Deals with the REST calls to the server

# Server Side Work List Configuration and Customization

Work list configurations are maintained as XML artifacts, registered in the product database and loaded during application bootup. The configuration controls filtering, sort criterion, additional query fields, which list view(s) is/are available, when multiple are available which one (CARD or TABULAR) is the default, and which fields display on which list view type. The work list view definition encompasses a lot more than its previous 4.x/5.0 counterpart and ties to the Solr index mapping resource. Refer to the [Solr Configuration](#) section for more information. This default behavior can be altered to source data from the database.

## Work List Definition

There can be one to many work list view definition files in a given application for the various work lists and queues. Each work list view definition file can have configurations for one or more views. Each view can have one or more list views and references to filters, sort criterion and additional query information.

Views can have permissions or no permissions. When a view is configured with permissions, then the permissions configured on the current user's security profile come into play when the view is selected. If a view has more than one permission configured, then the current user needs only one of the configured permissions for it to be selected. Views can also be configured with no permissions.

When a work list view definition has both views with and without configured permissions, then the one with a match on the permission name will take precedence over the one that doesn't. This is another difference from the 5.x/5.0 work list definitions: the 4.x/5.0 configuration model required a lot of redundant configuration across userroles/permissions, where the 5.2 modal allows the same configuration to be shared while accommodating those rare circumstances where view differentiation by role/permissions is deemed necessary.

## Supporting Data Structures

### Work List View Definition Schema

A work list view definition schema defines the structure for conveying various work list view instances. Examples of a work list view are the My Work and My Account pages, as well as the Most Recent and Waiting for Underwriter lists on the home page. Each work list view instance has a work item type, view name and related search index name. View names must be unique within the application. The work item type refers to whether the work list view is for accounts or regular work items. The related search index name must refer to the index mapping resource that shares the same index name.

#### Filters

Each work list view allows zero or more filters to be defined. Each filter has one or more localization content elements that supply the localization information. Each filter is comprised of the following properties:

Property Name	Data Type	Required	Default Value	Comments
id	xs:ID	True	N/A	Provides the unique identity of this filter. The value must be unique within a work list view instance. This value is only used for making references from within the same work list view instance.
name	xs:string	True	N/A	Provides the name of this filter.
fieldRefId	xs:string	True	N/A	Links to the field definition in the related index.
type	Enumeration: <ul style="list-style-type: none"><li>• LOB</li><li>• TRANSACTION_TYPE</li><li>• ACCOUNT_TYPE</li></ul>	True	N/A	Establishes the type of filter. The framework supports an LOB and transaction type for non-account based work lists and account type (personal or commercial) for account based work lists. Custom filters are supported and require customJava code to be provided by extending the framework's <code>WorkListViewProvider</code> class and supplying/overriding the

Property Name	Data Type	Required	Default Value	Comments
	• CUSTOM			IWorkListViewProvider.getCustomFilterOptions method.
isSaveable	xs:boolean	False	True	Indicates whether the filter should be saved by save search function.

#### Sort Information

Each work list view allows zero to more sort information elements to be defined. Each sort information instance has one or more localization content elements that supply the localization information. Each sort information instance has zero or more field references (fieldRef) to the field definitions in the respective index mapping resource. It is comprised of the following properties:

Property Name	Data Type	Required	Default Value	Comments
id	xs:ID	True	N/A	Provides the unique identity of this sort information instance. The value must be unique within a work list view instance. This value is only used for making references from within the same work list view instance.
name	xs:string	True	N/A	Provides the name of this sort information instance.
defaultSortClause.fieldRefId	xs:string	True	N/A	Defines the initial sort criterion. Links to the field definition in the related index.
defaultSortClauseascending	xs:boolean	False	False	Defines whether the initial sort will be ascending or descending.
fieldRef (repeating)	xs:string	True	N/A	Links to the field definition in the related index.
isSaveable	xs:boolean	False	True	Indicates whether the filter should be saved by save search function.

#### Additional Query Fields

Additional query fields are a special kind of filtering parameters that require more than just a user to select a value from a predefined list. These typically require the user to enter data values, such as data ranges, name snippets, identifiers, etc. Each work list view instance can contain zero or more query information instances. Each query information instance is comprised of the following properties:

Property Name	Data Type	Required	Default Value	Comments
id	xs:ID	True	N/A	Provides the unique identity of this query information instance. The value must be unique within a work list view instance. This value is only used for making references from within the same work list view instance.
name	xs:string	True	N/A	Provides the name of this query information instance.
transactionId	xs:string	False	None	Links to the TDF ID that supports the UI layout.
pageId	xs:string	If transactionId is present, then this is required	None	Links to the TDF page ID that supports the UI layout.

Each query information instance is comprised of zero or more query fields and are arranged as follows:

Property Name	Data Type	Required	Default Value	Comments
Interactive	xs:boolean	False	True	Determines whether the query field is fixed or interactive. True if interactive.
pageFieldRefId	xs:string	False	None	Refers to the related TDF page's field unique ID. This is required when the type attribute is interactive.
fieldRefId	xs:string	True	N/A	Links to the field definition in the related index.
isSaveable	xs:boolean	False	True	Indicates whether the filter should be saved by save search function.
opCode	Enumeration: <ul style="list-style-type: none"><li>• EQUALS</li><li>• LESS_THAN</li><li>• BETWEEN</li><li>• CONTAINS</li><li>• ONE_OF</li><li>• NOT_EQUALS</li></ul>	True	None	Defines the operator code to apply the field value with the operands.
operands.operand (repeating)	xs:string	One only.  Exceptions: <ol style="list-style-type: none"><li>1. Two required when opcode is BETWEEN</li><li>2. One or more allowed when opcode is ONE_OF</li></ol>	None	Defines the value or values that will filter the search results.

## Views

Each work list view allows zero or more views to be defined. Each view instance is composed of the following elements:

Property Name	Data Type	Required	Default Value	Comments
filterRefs	Container	False	N/A	Container for one or more filter references
filterRefs.filterRef (repeating)	Container			A filter reference
filterRefs.filterRef.filterRefId	xs:IDREF	True	N/A	Links to the filter instance to include.
sortInfoRef	Container	Flase	N/A	Container for the sort information reference
sortInfoRef.sortInfoRefId	xs:IDREF	True	N/A	Links to the sort information instance to include.
queryInfoInfoRef	Container	False	N/A	Container for the query information reference.

<b>Property Name</b>	<b>Data Type</b>	<b>Required</b>	<b>Default Value</b>	<b>Comments</b>
queryInfoInfoRef.queryInfoRefId	xs:IDREF	True	N/A	Links to the query information instance to include.
listView (repeating)	Container	False	N/A	Container for list view instances. A list view delineates those fields that are visible to the user.
listView.type	Enumeration: <ul style="list-style-type: none"><li>• CARD</li><li>• TABULAR</li><li>• CUSTOM</li></ul>	True	N/A	Determines whether this list view is tabular, card or custom based.
listView.viewPortSize	xs:integer	False	60	Establishes the view port size of this list view.
listView.selected	xs:boolean	False	False	When this property is true, this list view instance becomes the initial active one selected on the UI.
listView.fetchSize	xs:integer	False	60	Designates how many rows to fetch in one search request.
permission (repeating)	Container	False	N/A	View elements can have no permissions, meaning that any security role will have access to that view. Views can also have one or more permissions associated with it. The framework will allow users with just one of those permissions to access that view.
permission.name	xs:string	True	None	Designates the permission name that the current user's security profile must have to access this view instance.

### Sample Work List Definitions

The following work list view definition can be found in the 5.2 template bits @/templates/Web Root/WEB-INF/shared/worklist/solrWorklist.xml:

```

- <workListViewDefinition viewName="WorkItemsView" index="worklist" workItemType="regular" xsdNamespaceSchemaLocation="http://reference.agencyport.com/schemas/5.1/worklist
/workListViewDefinition.xsd">
- <!--
 http://reference.agencyport.com/schemas/5.1/worklist/workListViewDefinition.xsd C:\Projects\ sdk\apwebapp\resources\schemas\worklist\workListViewD
-->
- <filters id="filters1">
- <filter id="filter1" name="lob" fieldRefId="lob" type="LOB">
 <!-- Will inherit the title from its related field -->
</filter>
- <filter id="filter2" name="statuses" fieldRefId="status" type="STATUS">
 <content localId="1" title="Status"/>
</filter>
- <filter id="filter3" name="transaction_types" fieldRefId="transaction_type" type="TRANSACTION_TYPE">
 <content localId="1" title="Transaction"/>
</filter>
- <!--
<filter id="filter4" name="custom_filter" fieldRefId="custom-selectable-item" type="CUSTOM">
 <content localId="1" title="Custom Filter"/>
</filter>
-->
</filters>
- <sortInfos>
- <sortInfo id="sort1" name="workitem_sort_by">
 <defaultSortClause fieldRefId="last_update_time" ascending="false"/>
 <fieldRef fieldRefId="work_item_id"/>
 <fieldRef fieldRefId="entity_name"/>
 <fieldRef fieldRefId="last_update_time"/>
 <fieldRef fieldRefId="effective_date"/>
 <fieldRef fieldRefId="premium"/>
 <fieldRef fieldRefId="complete_percent"/>
 <content localId="1" title="Sort By"/>
</sortInfo>
</sortInfos>
- <queryInfos>
- <queryInfo id="query1" name="workitem_query_info" transactionId="worklist" pageId="worklistAdvancedSearch">
 <queryField fieldRefId="work_item_id" pageFieldRefId="workItemId" isSaveable="true"/>
 <queryField fieldRefId="effective_date" pageFieldRefId="effectiveDate" isSaveable="true"/>

```

```

- <queryField fieldRefId="entity_name">
 <opCode>CONTAINS</opCode>
 - <operands>
 <operand>*</operand>
 </operands>
</queryField>
- <queryField interactive="false" fieldRefId="status">
 <opCode>NOT_EQUAL</opCode>
 - <operands>
 <operand>DELETE</operand>
 </operands>
</queryField>
- <queryField interactive="false" fieldRefId="commit_flag">
 <opCode>EQUALS</opCode>
 - <operands>
 <operand>1</operand>
 </operands>
</queryField>
<content localeId="1" title="More Options"/>
</queryInfo>
</queryInfos>
- <views>
- <view>
 - <filterRefs>
 <filterRef filterRefId="filter1"/>
 <filterRef filterRefId="filter2"/>
 <filterRef filterRefId="filter3"/>
 <!--
 <filterRef filterRefId="filter4"/>
 -->
</filterRefs>
<sortInfoRef sortInfoRefId="sort1"/>
<queryInfoRef queryInfoRefId="query1"/>
- <listView type="CARD" selected="true" fetchSize="20" viewPortSize="20">
 <fieldRef fieldRefId="work_item_id"/>
 <fieldRef fieldRefId="account_id"/>
 <fieldRef fieldRefId="lob"/>
 <fieldRef fieldRefId="entity_name"/>
 <fieldRef fieldRefId="effective_date"/>
 <fieldRef fieldRefId="premium"/>
 <fieldRef fieldRefId="transaction_type"/>
 <fieldRef fieldRefId="status"/>
 <fieldRef fieldRefId="complete_percent"/>
</listView>
- <listView type="TABULAR" fetchSize="40" viewPortSize="40">
 <fieldRef fieldRefId="work_item_id"/>
 <fieldRef fieldRefId="account_id"/>
 <fieldRef fieldRefId="lob"/>
 <fieldRef fieldRefId="entity_name"/>
 <fieldRef fieldRefId="status"/>
 <fieldRef fieldRefId="last_update_time"/>
 <fieldRef fieldRefId="effective_date"/>
 <fieldRef fieldRefId="premium"/>
 <fieldRef fieldRefId="transaction_type"/>
 <fieldRef fieldRefId="complete_percent"/>
</listView>
</view>
</views>
</workListViewDefinition>

```

## Customizing Work List

### Adding a New Column - Account/Work Item

Adding field to an existing account view or work item view involves the following:

1. Add the new field to the Apache Solr schema definition.
2. Add the new field to the Solr index mapping definition. Refer to [Index Mapping Schema](#) for more information.
3. Add the new fieldRef to the list views where you want the field to show up.
4. Add another fieldRef to the sortInfo section if you want the field to be sortable.
5. Add a filter element, as appropriate, if you want the field to be filterable. Filterable fields are typically sourced from a list of values. Selecting one of the list values is sufficient. If the filter is custom, then you must extend the `com.agencyport.worklist.view.WorkListViewProvider` and provide an implementation for the `getCustomFilterOptions` method, which is responsible for serving up the list of filter options for that field.
6. Perform the following if the field can be queried by value and is not a simple filterable field:
  1. Create a small one page TDF to model the data collection for the new field. If there is an existing TDF, then add a `fieldElement` for the new field to the page definition.
  2. Create a `queryInfo` element if one does not exist and configure the `transactionId` and `pageId` page for the TDF under consideration.
  3. Add a `queryField` element to the `queryInfo` element. Configure the `fieldIdRef`, `pageFieldRefId` and `isSaveable` properties.

The following captures changes that were made to work list definition XML, the index map and the Solr schema to display TaxId in an agent's account view.

#### Work List View Definition Change

```
<views>
 <view>
 <filterRefs>
 <filterRef filterRefId="filter1"/>
 </filterRefs>
 <sortInfoRef sortInfoRefId="sort1"/>
 <queryInfoRef queryInfoRefId="query1"/>
 <listView type="CARD" selected="true" fetchSize="20" viewPortSize="20">
 <fieldRef fieldRefId="work_item_id"/>
 <fieldRef fieldRefId="account_number"/>
 <fieldRef fieldRefId="account_type"/>
 <fieldRef fieldRefId="entity_name"/>
 <fieldRef fieldRefId="address"/>
 </listView>
 <listView type="TABULAR" fetchSize="40" viewPortSize="40">
 <fieldRef fieldRefId="work_item_id"/>
 <fieldRef fieldRefId="tax_id" /> (highlighted)
 <fieldRef fieldRefId="account_type"/>
 <fieldRef fieldRefId="entity_name"/>
 <fieldRef fieldRefId="last_update_time"/>
 <fieldRef fieldRefId="address"/>
 </listView>
 </view>
</views>
```

#### Solr Index Mapping Change

```

<indexMapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" index="account"
 xsi:noNamespaceSchemaLocation="http://reference.agencyport.com/schemas/5.1/searchIndex/indexMapping.xsd">
 <!-- http://reference.agencyport.com/schemas/5.1/searchIndex/indexMapping.xsd C:\Projects\ sdk\apwebapp\resources\schemas\searchi
 <field id="work_item_id" name="account_id" indexFieldName="id" type="INTEGER" workItemPropertyName="WORKITEM_ID"></field>
 <field id="account_type" name="account_type" type="STRING" ></field>
 <field id="entity_name" name="entity_name" indexSearchFieldName="entity_name_search" type="STRING" ></field>
 <field id="last_update_time" name="last_update_time" type="TIME" ></field>
 <field id="address" name="address" type="STRING" ></field>
 <field id="city" name="city" type="STRING" ></field>
 <field id="state_prov_cd" type="STRING" ></field>
 <field id="postal_code" type="STRING" ></field>
 <field id="user_group_id" indexFieldName="usergroupid" type="INTEGER" workItemPropertyName="USERGROUP_ID"></field>
 <field id="account_number" type="STRING" ></field>
 <!--<field id="tax_id" type="STRING" >
 <content localeId="1" title="Tax Id"/>
 </field>-->
 <field id="creation_time" type="TIME" ></field>
 <field id="phone_number" type="STRING" ></field>
 <field id="account_pas_id" type="STRING" ></field>
 <field id="other_name" isSearchable="false" type="STRING" ></field>
</indexMapping>

```

#### Solr Schema Change

```

<fields>
 <!-- general -->
 <field name="_version_" type="long" indexed="true" stored="true"/>
 <field name="type" type="string" indexed="true" stored="true" multiValued="false"/>

 <field name="id" type="int" indexed="true" stored="true" multiValued="false" required="true"/>

 <field name="account_pas_id" type="string" indexed="true" stored="true" multiValued="false" required="true"/>
 <field name="entity_name" type="string" indexed="true" stored="true" multiValued="false" required="false"/>
 <field name="account_number" type="string" indexed="true" stored="true" multiValued="false" required="true"/>
 <!--<field name="tax_id" type="string" indexed="true" stored="true" multiValued="false" required="false"/>
 <field name="account_type" type="string" indexed="true" stored="true" multiValued="false" required="true"/>
 <field name="address" type="partial_search_type" indexed="true" stored="true" multiValued="false"/>
 <field name="city" type="partial_search_type" indexed="true" stored="true" multiValued="false"/>
 <field name="state_prov_cd" type="string" indexed="true" stored="true" multiValued="false"/>
 <field name="postal_code" type="partial_search_type" indexed="true" stored="true" multiValued="false"/>
 <field name="usergroupid" type="string" indexed="true" stored="true" multiValued="false"/>
 <field name="creation_time" type="date" indexed="true" stored="true" multiValued="false"/>
 <field name="last_update_time" type="date" indexed="true" stored="true" multiValued="false"/>
 <field name="phone_number" type="string" indexed="true" stored="true" multiValued="false"/>

 <field name="entity_name_search" type="partial_search_type" indexed="true" stored="true" multiValued="true"/>
 <copyField source="entity_name" dest="entity_name_search"/>
</fields>

```

Changes to support adding TaxId as a queryable field:

TDF Changes

```

<?xml version="1.0" encoding="UTF-8"?>
<transaction id="worklist" title="Advanced Search Dialogs"
 target="shared"
 lob="shared"
 type="new_business"
 supportsFastForward="false"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://reference.agencyport.com/schemas/5.1/definition/transactionDefinitionFile.xsd">

 <description>Page definitions for Advanced Search Dialogs for My Accounts and My Work</description>
 <page id="worklistAdvancedSearch" title="More Options" type="dataEntry">
 <page id="accountlistAdvancedSearch" title="More Options" type="dataEntry">
 <pageElement id="accountlistAdvancedSearch" type="fieldset" legend="More Options">
 <fieldElement type="text"
 id="SP.accountId"
 uniqueId="accountId"
 label="Account Number"
 size="9" maxLength="9">
 </fieldElement>
 <fieldElement type="text"
 id="SP.taxId"
 uniqueId="taxId"
 label="Tax Id"
 size="9" maxLength="9">
 </fieldElement>
 <fieldElement type="text"
 id="SP.city"
 uniqueId="city"
 label="City"
 size="30" maxLength="30">
 </fieldElement>
 <fieldElement type="selectlist"
 id="SP.state"
 uniqueId="state"
 label="State">
 <optionList reader="xmlReader" source="codeListRef.xml" target="state"/>
 </fieldElement>
 <fieldElement type="text"
 id="SP.zipCode"
 uniqueId="zipCode"
 label="Zip Code"
 size="10" maxLength="10">
 </fieldElement>
 </pageElement>
 </page>
 </page>
</transaction>

```

#### Work List View Definition Change

```

<queryInfos>
 <queryInfo id="query1" name="account_query_info" transactionId="worklist" pageId="accountlistAdvancedSearch">
 <queryField fieldRefId="work_item_id" pageFieldRefId="accountId" isSaveable="true"/>
 <queryField fieldRefId="tax_id" pageFieldRefId="taxId" isSaveable="true"/>
 <queryField fieldRefId="city" pageFieldRefId="city" isSaveable="true"/>
 <queryField fieldRefId="state_prov_cd" pageFieldRefId="state" isSaveable="true"/>
 <queryField fieldRefId="postal_code" pageFieldRefId="zipCode" isSaveable="true"/>
 <!-- The following query field accommodates the search for an account by name -->
 <queryField fieldRefId="entity_name" >
 <opCode>CONTAINS</opCode>
 <operands>
 <operand>*</operand>
 </operands>
 </queryField>
 <content localeId="1" title="More Options" />
 </queryInfo>
</queryInfos>

```

#### Adding New Work List

Perform the following to add a new work list:

1. Define a new work list view definition in the templates/worklist folder (WEB-INF/shared/worklist). Using the available metadata configuration, set the appropriate fields based on your requirements. Refer to the [Work List Configuration and Customization](#) topic for more information.
2. Add the definition file to the product database XML so that it can be loaded by the system.
3. Define the menu item for the new view in the menu.xml file similar to the My Work menu.

Refer to the Adding a New Work List section in the [My Work and My Account Configuration](#) topic for details on how to configure the client side work list.

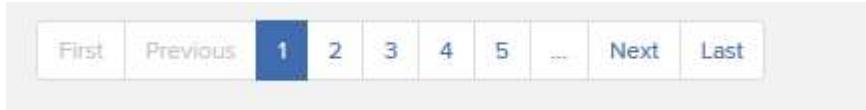
## List View

The `listView.type` property allows you to configure which view displays by default. Valid values are `CARD`, `TABULAR` and `CUSTOM`.

The work list view defaults to the last view you selected before you logged out. The `listView.selected` property allows you to turn this feature on or off.

## Pagination Configuration

The My Work and My Accounts work lists provide pagination to view multiple pages of items.



The number of items that display on each page is configurable. The default display for the card view of the My Work and My Account work list pages is 20 and the tabular view is 40. If there are less than the configured amount, the pagination feature does not display at the bottom of the page.

Change the `listView.viewPortSize` property to decrease or increase the number of items that display on the page.

You can also configure how many items display on each page from a search via the `listView.fetchSize` property.

# Client-Side My Work and My Account Configuration

Work items and accounts have a lot in common, which is why the code used to render the work lists for these two entities is the same. The following describes the main components of these work lists:

## Work List Search Controller (worklist.data.ctrl.js)

This controller is the entry point in the work list view. It performs the following tasks:

- Initiates a call to REST API (/api/worklists/{index\_name}/views/{view\_name}) to retrieve the metadata needed to paint the UI.
- Delegates to the data controller to render data in the appropriate view mode (card or tabular) as defined in the metadata.
- Reacts to the user actions, such as the selection of filters, sort options, name search, advanced search, my filters, switching between card/tabular view and calls the appropriate REST API to update the UI.

## Work List Data Controller (worklist.data.ctrl.js)

This controller deals with rendering the work item data while engaging the appropriate templates based on the view mode (card or tabular). It performs the following tasks:

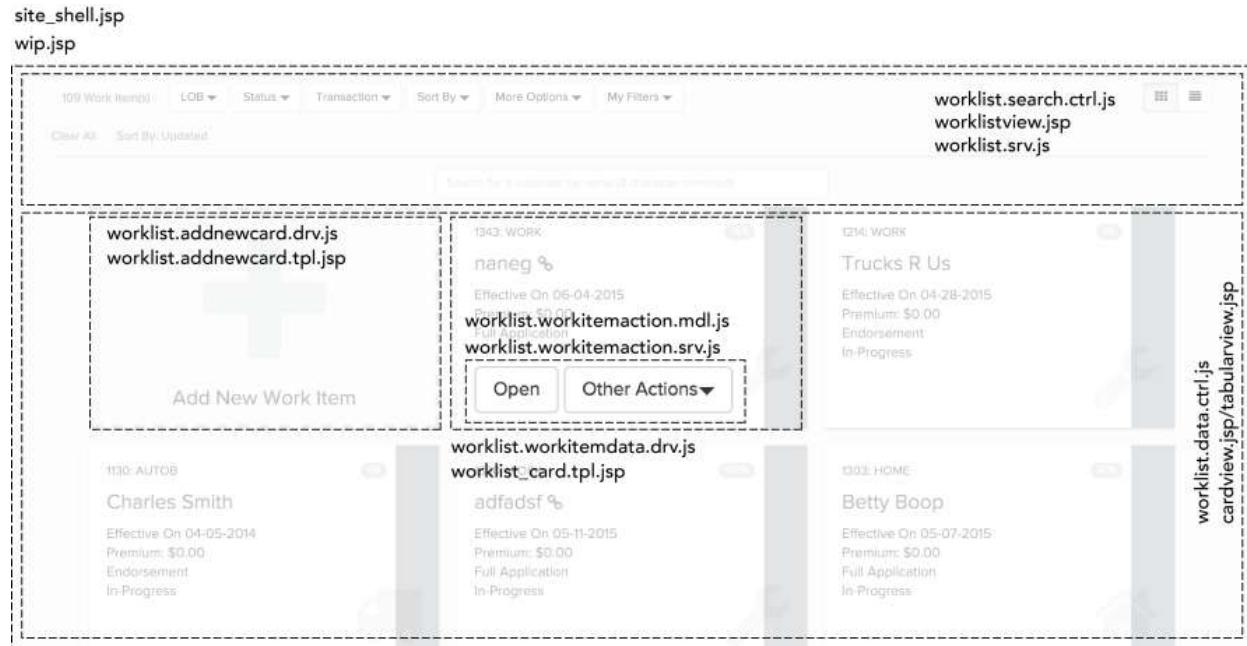
- Presents the actions available for a work item when the work item is selected.
- Performs the actions selected by the user for a particular work item.
- Deals with pagination by initiating a call to the REST API to get data appropriate for the view port.

## Work List Service (worklist.srv.js)

All of the calls to the REST API are directed through this service. The URLs used are parameterized so that they can be used for both the account and work item list

(/api/worklists/:worklistType/:views/:listId?:csrfTokenName=:csrfToken&:savedSearch\_id=:searchName). This service also serves as the datastore for the metadata and work item data. It also updates the CSRF token used for API calls with the new token available in the response (i.e., no two API calls will use the same CSRF token).

The following diagram indicates which files are responsible for each part of the UI. This should allow a developer to identify which files they have to modify if they have to debug or customize the UI:



## Adding New Fields to the UI

The following are tips to follow when adding new fields to the UI:

- For a view that uses a fixed layout (e.g., tabular view), a newly added field automatically displays without any change to the HTML or JavaScript. For a view that does not use a fixed layout (e.g., card view), you need to add the data using the appropriate HTML template (e.g., account-card.tpl.jsp, worklist-card.tpl.jsp).
- If the field data needs to be formatted for display, then you should add the appropriate formatting in the corresponding template file. You can use AngularJS built-in filters to do the formatting.

### Example

The following HTML snippet displays the premium field in currency format:

```
<li class="premium"><%= workItemRB.getString("label.WorkList.Premium") %> {{workitemData.premium | currency}}
```

## Adding a New Work List

Adding a new custom work list is made easy with the REST API architecture introduced in 5.1. If a project wants to add a new work list (e.g., My Renewals), a developer can add the list to the view by performing the following steps:

1. Add the parameters needed for the REST API and the new list in the worklist service (worklist.srv.js). If the view is just a different view of work items/accounts, then all you have to do is change the list ID.

### Example

The following is a new entry for RenewalWorkItems View:

```
self.urls = { WorkItems View: {listId: 'WorkItemsView', worklistType: 'workitems' }, AccountItems View: {listId: 'AccountItemsView', worklistType: 'workitems' }, Accounts View: {listId: 'AccountsView', worklistType: 'accounts' }, WorkItems View: {listId: 'RenewalWorkItemsView', worklistType: 'workitems' }, }
```

2. If the UI will be rendered differently than the existing My Work list, you must make the appropriate changes in the HTML templates that are available in the Web Root/worklist/partials folder.

## Adding a Custom LOB Link

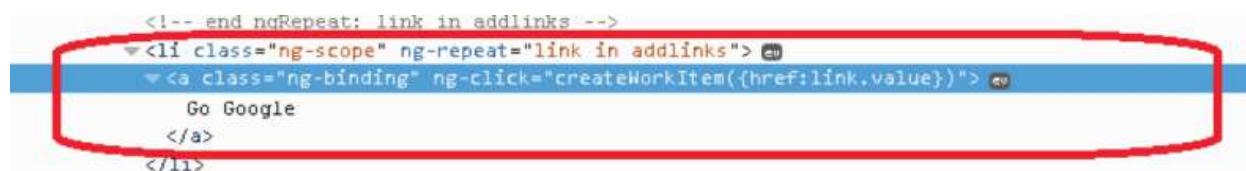
To add a custom link for a non-AgencyPortalLOB, extend the `BasicLookupProvider` class and include the URL that you want the browser to launch when the LOB item is selected.

### Example

```
/* * Created on Jul 28, 2015 by nbaker AgencyPort Insurance Services, Inc. */ package com.carrier.lookup; import javax.ws.rs.core.MultivaluedMap; import com.agencyport.api.lookup.pojo.Code; import com.agencyport.api.lookup.pojo.CodeList; import com.agencyport.lookup.BasicLookupProvider; import com.agencyport.lookup.LookupRealm; import com.agencyport.security.profile.ISecurityProfile; import com.agencyport.shared.APException; /** * The CarrierBasicLookupProvider class illustrates how to override the default product* lookup provider implementation. */ public class CarrierBasicLookupProvider extends BasicLookupProvider { /** * The <code>NAME_OF_INTEREST</code> is the name that we are interested in. */ private static final String NAME_OF_INTEREST = "NewWorkItemLink"; /** * Constructs an instance. */ public CarrierBasicLookupProvider(){ } /** * @inheritDoc */ public CodeList get(LookupRealm realm, String name, MultivaluedMap<String, String> queryParams, ISecurityProfile securityProfile) throws APException { CodeList codelist = super.get(realm, name, queryParams, securityProfile); if (NAME_OF_INTEREST.equals(name)){ // Here you can subtract existing or add other stuff to return list // I suppose you could resort stuff too? // Here is a very contrived example codelist.getCode().add(new Code("http://www.google.com", "Go Google")); } return codelist; } }
```

### Note

Make sure to change the package names to reflect your custom branding.



The above example launches Google.com when the Go Google option is selected from the Line of Business list.



# Work Item Quotes

Quote boxes provide a count of work item groups by status. Statuses include In-Progress, Decline, Approve and Bind. Clicking on a quote box for a particular status launches the My Work page and applies the corresponding status filter to only display the work items for that status. This component can be configured using the work list view definition configuration.



The quote box HTML is defined in the home.jsp file:

```
<section class="quotes"><div id="quote_container" class="container"><h3 class="header">Quotes</h3><div class="row"><div class="col-xs-6 col-md-3" ng-repeat="status in recentQCtrl.quoteBoxes.statuses"><a ng-href="DisplayWorkInProgress?WorkListType=WorkItemsView&status={{status}}"><div class="stat_label-{{status | lowercase}}"><h1> {{recentQCtrl.quoteBoxes.workItems | filter: {status:status}}.length }</h1><p class="bottom-right">{{recentQCtrl.lookupValue('STATUS',status)}}</p> </div> </div> </div> </div> </section>
```

The following is the work list view definition used to render the quotes component (solrQuotesByStatus.xml):

```
<workListViewDefinition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" viewName="QuotesByStatus" index="worklist" workItemType="regular" xsi:noNamespaceSchemaLocation="http://reference.agencyport.com/schemas/5.1/worklist/workList ViewDefinition.xsd"> <sortInfos> <sortInfo id="sort1" name="workitem_sort_by" > <defaultSortClause fieldRefId="status" ascending="false"/> <fieldRef fieldRefId="status"/> <content localeId="1" title="Sort By" /> </sortInfo> </sortInfos> <queryInfos> <queryInfo id="query1" name="quotes_by_status_info" ><!-- The following query field accommodates the search for a work item by name --> <queryField interactive="false" fieldRefId="status" > <opCode>ONE_OF</opCode> <operands> <operand>INPROGRESS</operand> <operand>DECLINE</operand> <operand>APPROVE</operand> <operand>BIND</operand> </operands> </queryField> <queryField interactive="false" fieldRefId="commit_flag" > <opCode>EQUALS</opCode> <operands> <operand>1</operand> </operands> </queryField> <content localeId="1" title="Queue Search" /> </queryInfo> </queryInfos> <views> <view> <filterRefs> </filterRefs> <sortInfoRef sortInfoRefId="sort1" /> <queryInfoRef queryInfoRefId="query1" /> <listView type="TABULAR" fetchSize="500" > <fieldRef fieldRefId="work_item_id" /> <fieldRef fieldRefId="status" /> </listView> </view> </views> </workList ViewDefinition>
```

# Work Item Queues

By default, the work list normally lists all of the work items available to the user group, including those not created or owned by the current user. The work items queue feature provides a useful place to list a user's *own* work items. Work item queues were designed to give users a quick way to navigate to the work items most relevant to them. Out of the box, AgencyPortal's home page contains two work item queues:

- a most recent queue
- a waiting for underwriter queue

## Queue

Most Recent				Waiting for Underwriter			
Name	Status	Type	Duration	Name	Status	Type	Duration
Darla Crane	In-Progress	WORK	1 days	test test	Referred	AUTOP	5 days
John Doe	In-Progress	WORK	4 days	Betty Boop	Referred	AUTOP	20 days
Charles Smith	In-Progress	AUTOB	4 days	Catrina Moriarty	Referred	HOME	186 days
John Doe	In-Progress	WORK	4 days	Donal Murphy	Referred	HOME	186 days
Charles Smith	In-Progress	AUTOB	4 days	Damien Tusk	Referred	HOME	186 days

These work item queues are configurable and removable. You can also add more than two queues if necessary. By default, the queues are configured as follows:

- Most Recent Queue - displays the last five updated work items that are in a status other than Referred and the creator of the work item is the currently logged in user.
- Waiting for Underwriter Queue - displays the last five updated items that are in a Referred status and the creator of the work item is the currently logged in user.

This is how these queues are defined in the home.jsp file:

```
<div ng-app="app.worklist.queues" ng-cloak data-ng-controller="app.worklist.recentWorkItemsQueueCtrl as recentCtrl">
 <section class="queue">
 <div class="content">
 <h3>{{coreRb.getString("label.Queue")}}</h3>
 <div id="agent_queue" class="queue-agent col-xs-12 col-sm-6 animated fadeIn">
 <div>
 <i></i>{{coreRb.getString("label.MostRecent")}}</div>
 </div>
 <table class="table table-striped sortable">
 <thead>
 <tr>
 <th ng-repeat="field in recentCtrl.agentqueue.metaData.listViews[0].fields">{{field.title}}</th>
 </tr>
 </thead>
 <tbody>
 <tr ng-repeat="workitem in recentCtrl.agentqueue.workItems">
 <td>{{workitem.work_item_id}}</td>
 <td>{{workitem.entity_name}}</td>
 <td class="hidden-xs" style="text-align: right;">{{recentCtrl.lookupValue('STATUS', workitem.status)}}</td>
 <td>{{workitem.lab}}</td>
 <td class="hidden-xs hidden-sm hidden-md" style="text-align: right;">{{workitem.effective_date}}</td>
 <td>{{recentCtrl.getDaysFromToday(workitem.last_update)}}</td>
 </tr>
 </tbody>
 </table>
 </div>
 </div>
 <div id="underwriter_queue" class="queue-underwriter col-xs-12 col-sm-6 animated fadeIn">
 <div>
 <i></i>{{coreRb.getString("label.WaitingForUnderwriter")}}</div>
 </div>
 <table class="table table-striped sortable">
 <thead>
 <tr>
 <th ng-repeat="field in recentCtrl.underwriterqueue.metaData.listViews[0].fields">{{field.title}}</th>
 </tr>
 </thead>
 <tbody>
 <tr ng-repeat="workitem in recentCtrl.underwriterqueue.workItems">
 <td>{{workitem.work_item_id}}</td>
 <td>{{workitem.entity_name}}</td>
 <td class="hidden-xs" style="text-align: right;">{{recentCtrl.lookupValue('STATUS', workitem.status)}}</td>
 <td>{{workitem.lab}}</td>
 <td class="hidden-xs hidden-sm hidden-md" style="text-align: right;">{{workitem.effective_date}}</td>
 <td>{{recentCtrl.getDaysFromToday(workitem.last_update)}}</td>
 </tr>
 </tbody>
 </table>
 </div>
 </section>
</div>
```

The following is the work list view definition configuration files for the most recent and underwriter queues:

```

<worklistViewDefinition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" viewName="WorkItemsRecentAgentQueue"
 index="worklist" workitemType="regular" xsi:noNamespaceSchemaLocation="http://reference.agencyport.com/schemas/5.1/worklist/worklistViewDefinition.xsd">
 <sortInfos>
 <sortInfo id="sort1" name="workitem_sort_by" >
 <defaultSortClause fieldRefId="last_update_time" ascending="false"/>
 <fieldRef fieldRefId="last_update_time"/>
 <content localeId="I" title="Sort By"/>
 </sortInfo>
 </sortInfos>
 <queryInfos>
 <queryInfo id="query1" name="recent_workitems_query_info" >
 <!-- The following query field accommodates the search for a work item by name -->
 <queryField interactive="false" fieldRefId="status" >
 <opCode>EQUALS</opCode>
 <operands>
 <operand>INPROGRESS</operand>
 </operands>
 </queryField>
 <queryField interactive="false" fieldRefId="status" >
 <opCode>NOT_EQUAL</opCode>
 <operands>
 <operand>DELETE</operand>
 </operands>
 </queryField>
 <queryField interactive="false" fieldRefId="creator_id" >
 <opCode>EQUALS</opCode>
 <operands>
 <operand>*</operand>
 </operands>
 </queryField>
 <content localeId="I" title="Queue Search" />
 </queryInfo>
 </queryInfos>
 <views>
 <view>
 <filterRefs>
 </filterRefs>
 <sortInfoRef sortInfoRefId="sort1"/>
 <queryInfoRef queryInfoRefId="query1"/>
 <listView type="TABLE" fetchSize="5" >
 <fieldRef fieldRefId="work_item_id"/>
 <fieldRef fieldRefId="entity_name"/>
 <fieldRef fieldRefId="status"/>
 <fieldRef fieldRefId="lob"/>
 <fieldRef fieldRefId="effective_date"/>
 <fieldRef fieldRefId="last_update_time"/>
 </listView>
 </view>
 </views>

```

```

<workListViewDefinition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" viewBox="WorkItemsRecentQueue"
 index="workList" workItemType="regular" xsi:noNamespaceSchemaLocation="http://reference.agencyport.com/schemas/5.1/worklist/workListViewDefinition.xsd">
 <sortInfos>
 <sortInfo id="sort1" name="workitem_sort_by" >
 <defaultSortClause fieldRefId="last_update_time" ascending="false"/>
 <fieldRef fieldRefId="last_update_time"/>
 <content localeId="1" title="Sort By"/>
 </sortInfo>
 </sortInfos>
 <queryInfos>
 <queryInfo id="query1" name="recent_workitems_query_info" >
 <!-- The following query field accommodates the search for a work item by name -->
 <queryField interactive="false" fieldRefId="status" >
 <opCode>NOT_EQUAL</opCode>
 <operands>
 <operand>INPROGRESS</operand>
 </operands>
 </queryField>
 <queryField interactive="false" fieldRefId="status" >
 <opCode>NOT_EQUAL</opCode>
 <operands>
 <operand>DELETE</operand>
 </operands>
 </queryField>
 <queryField interactive="false" fieldRefId="creator_id" >
 <opCode>EQUALS</opCode>
 <operands>
 <operand>*</operand>
 </operands>
 </queryField>
 <content localeId="1" title="Queue Search" />
 </queryInfo>
 </queryInfos>
 <views>
 <view>
 <filterRefs>
 </filterRefs>
 <sortInfoRef sortInfoRefId="sort1"/>
 <queryInfoRef queryInfoRefId="query1"/>
 <listView type="TABULAR" fetchSize="5" >
 <fieldRef fieldRefId="work_item_id"/>
 <fieldRef fieldRefId="entity_name"/>
 <fieldRef fieldRefId="status"/>
 <fieldRef fieldRefId="lob"/>
 <fieldRef fieldRefId="effective_date"/>
 <fieldRef fieldRefId="last_update_time"/>
 </listView>
 </view>
 </views>

```

JavaScript code that calls the REST APIs and renders the UI is located in `worklist.recentWorkItemQueue.ctrl.js`.

# Work List Customizations

This topic includes instructions on how to customize the work list.

## Custom Work Item Actions

Perform the following to add custom work item actions:

1. Add your custom action to the workitemAction.xml file.

### Example

```
<workItemAction actionCode="Withdraw" title="Withdraw" requiredPermission="accessCanWithdraw" isPrimary="false"> <when workItemType="regular" status="125" isCreator="true" isOwner="true" /> <url workItemType="regular">WorkItemAction?WORKITEMID=${WORKITEM_ID}&action=Withdraw&WorkListType=WorkItemsView</url> </workItemAction>
```

2. Set the appropriate permission for the action in the database.
3. Add a new JavaScript file that extends the product controller and paints the work list (ap.worklist.worklistDataCtrl).

### Example

```
Sample code: cust.worklist.worklistDataCtrl.js (function() { 'use strict'; angular.module('ap.worklist.main') .controller('cust.worklist.worklistDataCtrl', ['$scope', '$controller', 'WorkItemActionsSrv', 'csrf', function($scope, $controller, WorkItemActionsSrv, csrf) { var self = this; angular.extend(self, $controller('ap.worklist.worklistDataCtrl', {$scope: $scope})); self.performAction = function(selectedWorkItem, action){ //customlogic }; self.performCustomAction = function(selectedWorkItem,action){ console.log("add customlogic here"); }; }]); })();
```

4. Override the performAction or performCustomAction functions to handle the custom action.
5. Add the js file to Gruntfile.js.
6. Change the JSP's that use the controller to use the new custom controller.

### Example

cardView.jsp

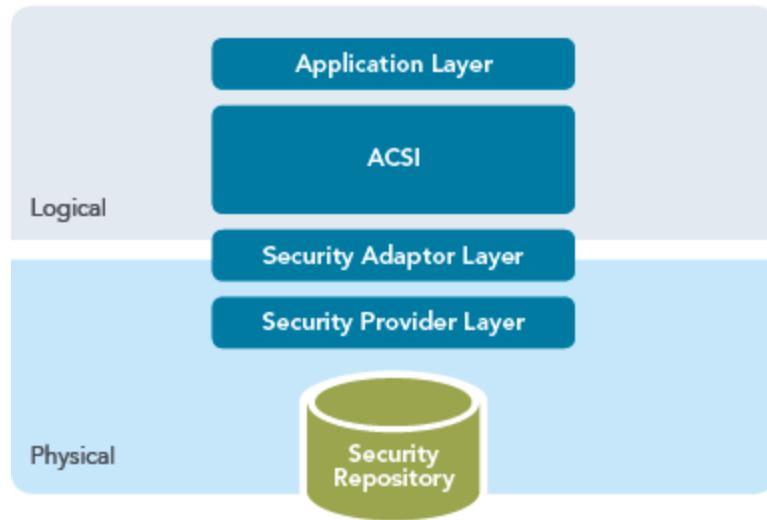
```
<section id="worklist_contents_section" class="worklist-section" data-ng-controller="ap.worklist.worklistDataCtrl as dataCtrl">
replace "ap.worklist.worklistDataCtrl" with "cust.worklist.worklistDataCtrl"
<section id="worklist_contents_section" class="worklist-section" data-ng-controller="cust.worklist.worklistDataCtrl as dataCtrl">
```

# Security

Agencyport Common Security Interface (ACSI) standardizes how Agencyport applications and core Agencyport products make authentication and authorization assertions throughout the course of the application execution via a common set of generalized abstractions and interfaces.

The following logical diagram identifies the major logical functional units that comprises an ACSI-based application:

## ACSI Layered Logical Architecture



This section includes a variety of information about security and AgencyPortal, including effective security programming, additional security considerations (including the OWASP Top 10), implementing a single sign on and security references for implementation.

# Effective Security Programming

This section clearly outlines the steps that a developer should follow to implement a sound ACSI based security implementation that aligns with the tenets of ACSI design principles and best practices. This information is targeted to a Java developer who has been given one of the following tasks:

- To build out the security implementation for 5.x and 4.x applications.
- To upgrade the security implementation of a 3.x application to conform to the ACSI 4.x interface mode.

# Security Programming Terminology

The following terms are used throughout this section:

- Core security library - otherwise known as `apsecurity.jar`. This runtime JAR contains all of the core interfaces and abstractions supporting the ACSI. It contains a fully concrete implementation for the security model entities, including the subject, role, permission and user/group entities. Your application will need this JAR file.
- Security reference library - otherwise known as `apsecurityreference.jar`. This contains a concrete security provider and supporting classes that inflate the security model from a nearby physical relational data model that has no relevance to how you will inflate the security model for your application. Your application does NOT need this JAR file nor will it need any of the tables associated with the DDL found in the `securityreference_sql`.



Name	Type	Modified	Size	Ratio	Packed	Path
drop_table.aci.sql	Microsoft SQL Server Query File	11/17/2009 4:09 PM	164	58%	69	sql\securityreference_sql\
create_aci_tables.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	3,857	72%	1,064	sql\securityreference_sql\db2\
load_aci_nextkey.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	106	6%	100	sql\securityreference_sql\db2\
load_aci_tables.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	26,191	92%	2,111	sql\securityreference_sql\db2\
load_aci_user.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	5,907	81%	1,147	sql\securityreference_sql\db2\
nested_agent_roles.load_aci_tables.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	24,500	92%	2,040	sql\securityreference_sql\db2\
real_aci_tables.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	3,858	72%	1,065	sql\securityreference_sql\db29\
load_aci_nextkey.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	107	6%	101	sql\securityreference_sql\db29\
load_aci_to_mysql.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	26,192	92%	2,112	sql\securityreference_sql\db29\
load_aci_user.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	5,908	81%	1,148	sql\securityreference_sql\db29\
nested_agent_roles.load_aci_tables.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	24,501	92%	2,041	sql\securityreference_sql\db29\
create_aci_tables.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	3,867	72%	1,065	sql\securityreference_sql\mysql\
load_aci_nextkey.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	108	6%	102	sql\securityreference_sql\mysql\
load_aci_tables.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	26,193	92%	2,113	sql\securityreference_sql\mysql\
load_aci_user.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	5,909	81%	1,150	sql\securityreference_sql\mysql\
nested_agent_roles.load_aci_tables.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	24,502	92%	2,043	sql\securityreference_sql\mysql\
create_aci_tables.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	3,830	72%	1,062	sql\securityreference_sql\oracle\
load_aci_nextkey.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	109	6%	103	sql\securityreference_sql\oracle\
load_aci_tables.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	25,574	92%	2,098	sql\securityreference_sql\oracle\
load_aci_user.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	5,800	80%	1,140	sql\securityreference_sql\oracle\
nested_agent_roles.load_aci_tables.sql	Microsoft SQL Server Query File	10/13/2011 3:32 PM	23,883	92%	2,027	sql\securityreference_sql\oracle\

The files in the above WinZip listing are only needed to run the AgencyPortal template applications in some demo mode context. Product development has deliberately separated the security model and associated abstractions from how the security model is to be inflated or represented in some relational data model.

- Sample Security Package - a brand new set of Java classes and DDL that accompany this document to illustrate various techniques and principles of ACSI, including the support of fine-grained permissions. It illustrates fine-grained permissions, as well as numeric identity provisioning to support subject and work (user) group identity. This package is positioned as the ACSI implementation of choice for illustrative purposes going forward, supplementing any need (presently or in the future) to refer to the security reference library or its associated DDL for any reason.

# Step-by-Step to ACSI

This section includes step-by-step procedures for creating an ACSI implementation for an AgencyPortal based web application. It is recommended that the developer read the entire [Security](#) section before embarking on any ACSI implementation. Refer to the [Sample Security Package](#) for sample security package information.

The following procedures are included in this section:

- [Setting Up an acsi.properties File](#)
- [Fleshing Out the ACSI Security Model](#)
- [Implementing the Security Provider](#)
- [Extending the Security Profile Manager](#)
- [Implementing the Security Filter](#)
- [Extending the Security Profile](#)

## Sample Security Package

All of the sample code referred to in this section is available for download from your FTP site. The code illustrations referred to in this documentation are packaged as source code and contained in a zip file called [security\\_samples\\_src.zip](#).

### Note

This zip file is not a runtime unit and should not be deployed with your application. The code found in this package replaces any need to refer to the security reference implementation ever again.

This sample code illustrates concepts and techniques only. Do not take this code as is without first taking the time to understand it and determine which segments can and cannot be used. Besides the obvious use of the text fragment "carrier" in both Java package naming and various Java classes, there are several places in the sample code where assumptions have been made, especially with regards to the particulars of a carrier's SSO authentication handshake, the granularity of their back end security data provider calls and the technical protocol for calling those back end data services.

The Java classes contained in the Java package named `com.carrier.security.fabrications` and the `callBackEndService()` method on each of the entity providers are pure fabrications. These fabrications are intended to illustrate the where, but not the how or the what. Product development cannot predict the nature or number of back end data services that you will need to interface with to inflate security model entities.

The sample code implementation is based on the following list of requirements. Every implementation needs a set of requirements, including this fabricated scenario. If some of these requirements coincide with the realities of your application, then this is entirely a coincidence. The importance of you getting your own set of requirements for security, such as the ones listed, cannot be overstated.

The following is our set of fabricated security requirements:

- The SSO handshake request passes the login ID via an HTTP cookie named "UID." The application can, in this case, assume that this represents an authenticated user.
- The carrier back end exposes three interfaces:
  - Interface #1 returns information on the current user, including their name and whether the user is an internal user. Internal users are typically underwriters, administrators and other support personnel. External users are agents, producers, brokers and other personnel situated out at the agency.
  - Interface #2 returns the roles that a user participates in. Most users participate in just one role, but there are some that participate in more than one role.
  - Interface #3 returns the work (user) group that an external user is a member of. Internal users are assumed to belong to their own internal user group.
- External users are grouped by agency into the same user group. Users within the same agency can access each others work items . External users in different user groups cannot access work items from another user group. Internal users are grouped as one entity and have access to all work items through the entire application, regardless of the user group the work item was created in or has current ownership of.

## Setting Up an acsi.properties File

### Best Practice

Every project should store all of the application properties specific to ACSI in its own acsi.properties file. This file will look something like the following:

```
#####
Properties specific to the support of ACSI
ACSI properties - engages correct
implementations for various application ACSI extensions
security.profile_classname=com.agencyport.security.profile.builtin.PortalSecurityProfile
security.profile_manager_classname=com.carrier.security.CarrierSecurityProfileManager
#####
ACSI # Factory list for implementations of
com.agencyport.security.resource.ISecureResourceFactory interface. # Each factory implementation is a factory for a
com.agencyport.security.resource.ISecureResource instance # which is sensitive to the current security profile. # Examples of secure resources
are menus and front servlet (front controller). # These are used by security filter. The menu resource is also used # to support the menu UI.
#####
secure_resource_factories=com.agencyport.secure.menu.provider.SecureMenuFactory;com.agencyport.secure.front.SecureFrontControllerFactor
y
```

Add a reference in the `additional_properties_to_load` property list of the main application properties file so the `InitializerServletContextListener` can locate the properties file:

```
#####
Link to other properties files
resources_root=${my_context_path}WEB-INF/
additional_properties_to_load=${resources_root}acsi.properties
```

## Fleshing Out the ACSI Security Model

The ACSI security model typically varies very little from project to project. In fact, it is our belief that the default Java classes for the security model entities found in the `com.agencyport.security.model.impl` package can be used out of the box in many cases. However, there may be a situation when your application requires that one or more of these entities need an additional property to fulfill a specific business requirement set forth by your customer.

### Security Model Factory Design Pattern

The `com.agencyport.model.factory.SecurityModelFactory` class is the factory class for all security model entities.

### Best Practice

When creating new security entities, subjects, roles, userGroups, [even if the creation point resides in your own custom code](#), you should use the security model factory. There is no need to use the entity constructors outside the scope of this factory. Furthermore, there should be no reason to extend the product's own security model factory class either since it acquires the Java class name for each security model entity by referring to a unique application property for each specific security model interface.

The following table lists all security model entities, their interfaces and the default core security library implementations, alongside the application properties that you can configure for your custom security model entity class:

<b>Entity</b>	<b>Interface (from package com.agencyport.security.model)</b>	<b>Default implementation Java class when application property is missing (from package com.agencyport.security.model.impl)</b>	<b>Application property used by the standard security model factory class to override default Java class name</b>
subject	ISubject	Subject	security.subject_classname
user principal	IUserPrincipal	UserPrincipal	security.userprincipal_classname
user info	IUserInfo	UserInfo	security.userinfo_classname
role	IRole	Role	security.role_classname
role group	IRoleGroup	RoleGroup	security.rolegroup_classname
roles	IRoles	Roles	security.roles_classname
user group	IUserGroup	UserGroup	security.usergroup_classname
user groups	IUserGroups	UserGroups	security.usergroups_classname

Entity	Interface (from package com.agencyport.security.model)	Default implementation Java class when application property is missing (from package com.agencyport.security.model.impl)	Application property used by the standard security model factory class to override default Java class name
permission	IPermission	Permission	security.permission_classname
Profile	ISecurityProfile	N/A (requires explicit configuration)	security.profile_classname

Inspecting each supporting entity interface, alongside the specific requirements of your application will help you determine whether you need to extend the core entity class.

## Extending the Model

After you have studied all of the interfaces, you may come to the conclusion that the core security library implementation classes are adequate as is. In which case, you may skip to this step. However, if you have concluded that one or more security model entities needs augmentation, follow the following steps:

1. Identify which security model entities need augmented or modified based on your evaluation of the security requirements made on your application.
2. Extend those security model entity classes by adding data members or functionality that are not modeled in the core security model classes.

## Best Practice

Although the security architecture allows you to implement any of the security model entity interfaces entirely from scratch, [you should always consider extending the core security model class first](#) before contemplating a full custom implementation.

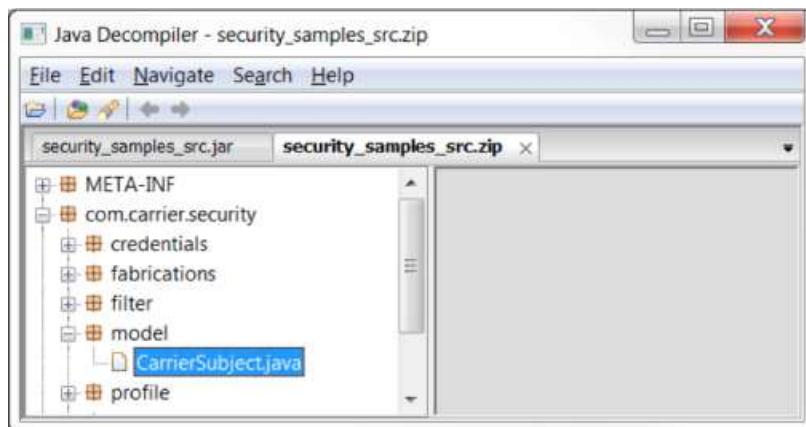
Remember that the base security model entity classes are part of the core security library and should be leveraged where and whenever possible.

## Best Practice

Ensure that all of your security model extensions contain a `toString` method. Moreover, it is a good practice in general to implement a `toString()` method on any application custom Java class that is "stateful."

## Sample References

Let's review the entity we selected to illustrate this principle in the samples security zip package. Again, the entities you extend may be different or the reason for extension may be rooted in a different cause. For our sample security package, we have augmented the subject entity to track whether the user is an internal user and to track whether the user has been authenticated. Review the `CarrierSubject` Java class located in the sample's security zip package for more details.



For the security factory to create this custom extension, you must register the full package Java class names in `aci.properties` for this class. The following is an example on how to register our custom carrier subject class:

## Example

```
Properties specific to the support of ACSI
ACSI properties - engages correct
implementations for various application ACSI extensions security.subject_classname=com.carrier.security.model.CarrierSubject
```

The next time the AgencyPortal framework or your custom application security provider calls either of the security model factory methods, `createSubject(IUserPrincipal)` or `createSubject(String, String)`, a `CarrierSubject` instance is created instead of the default one. Remember that all ACSI properties, like any other application properties, are loaded once at application bootstrap time only. During the development phase, if you happen to add or change one of these application properties while your application service is running, you must reboot the server for the new values to take effect.

## Importance of Supporting Fine-Grained Permissions

### Best Practice

Leveraging fine-grained permissions, also referred to as entitlements, for the purpose of evaluating whether a user is authorized to execute a particular request or functional unit is viewed as a superior technique to compare the role ID of name itself against a list of qualifying role IDs.

#### Making the Case for Fine-Grained Permissions

To illustrate the principle of leveraging fine-grained permissions, let's pretend there is a link on a custom page to your portal application that, when clicked, displays a message that indicates whether the system "thinks" the user has attempted to "game the system" in context of the current work item. This link is visible only to a limited set of users. Assume there are three roles in the system with the names of "admin," "underwriter," and "agent." The underwriter role is permitted to click on the link, but the agent is not. If you did not have explicit permissions tied to roles, your JSP code might look something like the following:

```
IPrincipal user = securityProfile.getSubject().getPrincipal(); IRole roles = securityProfile.getRoles(); if (roles.isUserInRole(user, "underwriter")) { // emit the HTML for the link in question }
```

If you had explicit permissions at your disposal, then your JSP could look something like the following:

```
IPrincipal user = securityProfile.getSubject().getPrincipal(); IRole roles = securityProfile.getRoles(); if (roles.checkPermission(user, "canAccessGamingTheSystemMessage")) { // emit the HTML for the link in question }
```

So what's the big deal? Pretend that the business requirements change midstream and the customer wants the admin role to also have permission to click the link. The role based would need adjustment as follows:

```
if (roles.isUserInRole(user, "admin") || roles.isUserInRole(user, "underwriter")) { // emit the HTML for the link in question }
```

Whereas, if you were employing fine-grained permissions, you could simply add the "canAccessGamingTheSystemMessage" permission to the admin role in the database without the need to change any JSP code.

The resilience of fine-grained permissions cannot be overstated. The only time you may need to add new code is when a new permission is added or deleted. Any changes in the relationships of permissions to roles can be driven entirely with metadata and should not require a coding change. An added benefit is that applications designed with RBAC decisions are driven entirely by fine-grained permission based applications, which accommodates the introduction of brand new role groupings without the need to change any code (assuming that no new permissions themselves have been introduced).

As part of the analysis and design phase, you must identify the set of permissions for your application. That does not mean you cannot add the new permissions later during the development phase since some of the permissions will be dictated by your customer and the functional set that needs protection. However, it is important to note that some of the AgencyPortal product subsystems also rely on the presence of fine-grained permissions to function properly. The following is a partial list of those subsystems:

- Work list - every work list action that displays is tied to a permission
- Dashboard report - each dashboard report is tied to a permission
- Menu (i.e., Home/My Work/Toolkit menu) - each menu entry is tied to a permission

If you come to the conclusion that fine-grained permissions are deemed unnecessary, note that this decision will compromise your ability to easily leverage those product features that do require explicit permissions. This decision will also greatly increase the brittleness of the security aspects of your application as changes are introduced in the future.

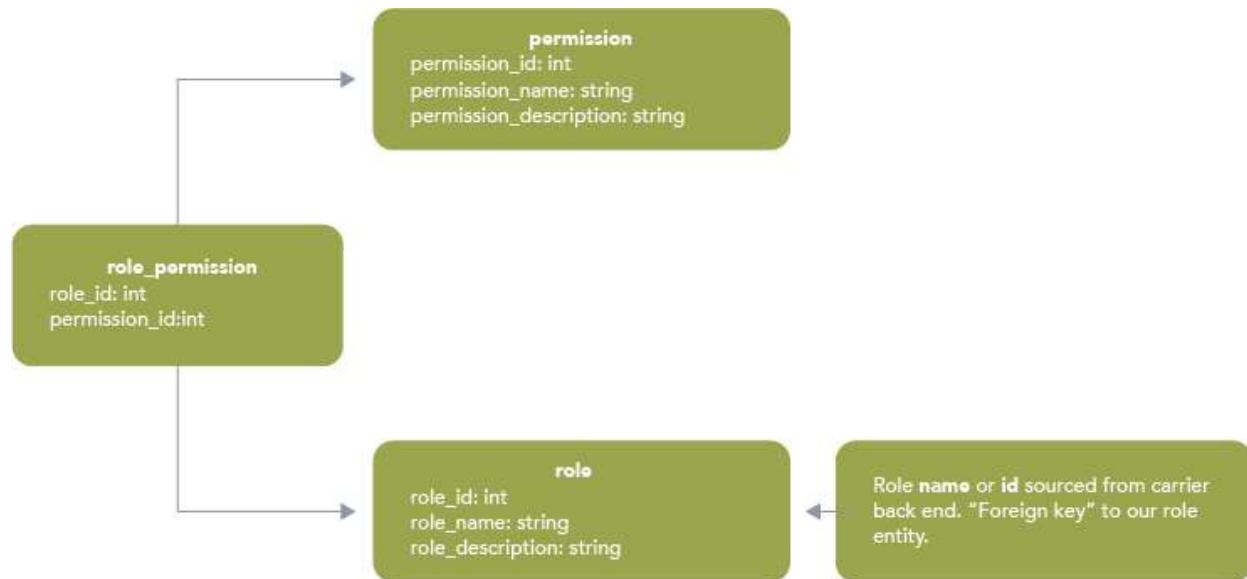
#### How to Introduce Fine-Grained Permissions

If you've decided to embrace fine-grained permissions, you must know that most carrier back end security interfaces provide very little information and, very rarely, return fine-grained permissions. In the event the carrier does, use fine-grained permissions directly to inflate the permission instances that live in the role.

If you're asking yourself "if the carrier security back end only returns a role name or ID, how am I to leverage the concept of explicit fine-grained permissions?" the answer is that you must maintain a small security data model in the portal database that augments the notion of a role with the explicit permissions. This data model is very simple and is comprised of three relational tables. The key into the model is the role name or ID, depending on the type of data that the carrier's back end returns to you.

Each distinct role name or ID returned by the carrier's back end service call will have a discrete unique row in the role table. It is duly noted that the technique of connecting a role identity returned by the back end to a nearby database table is brittle in and of itself. This recognized brittleness would be quickly neutralized by the fact that your application can now leverage explicit fine-grained permissions.

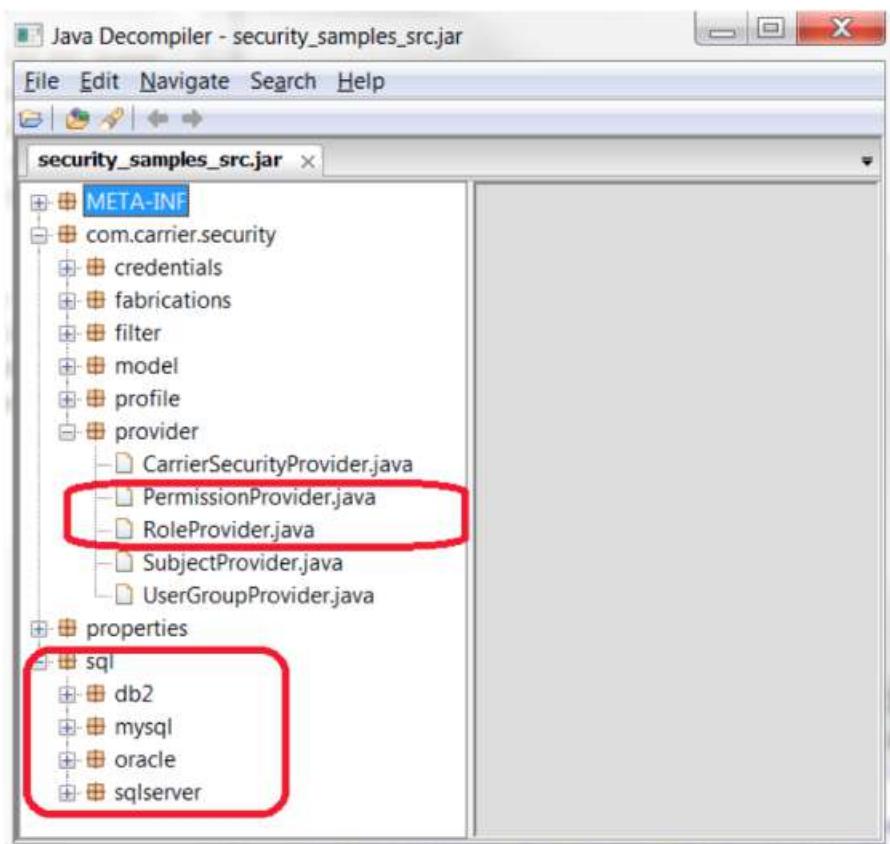
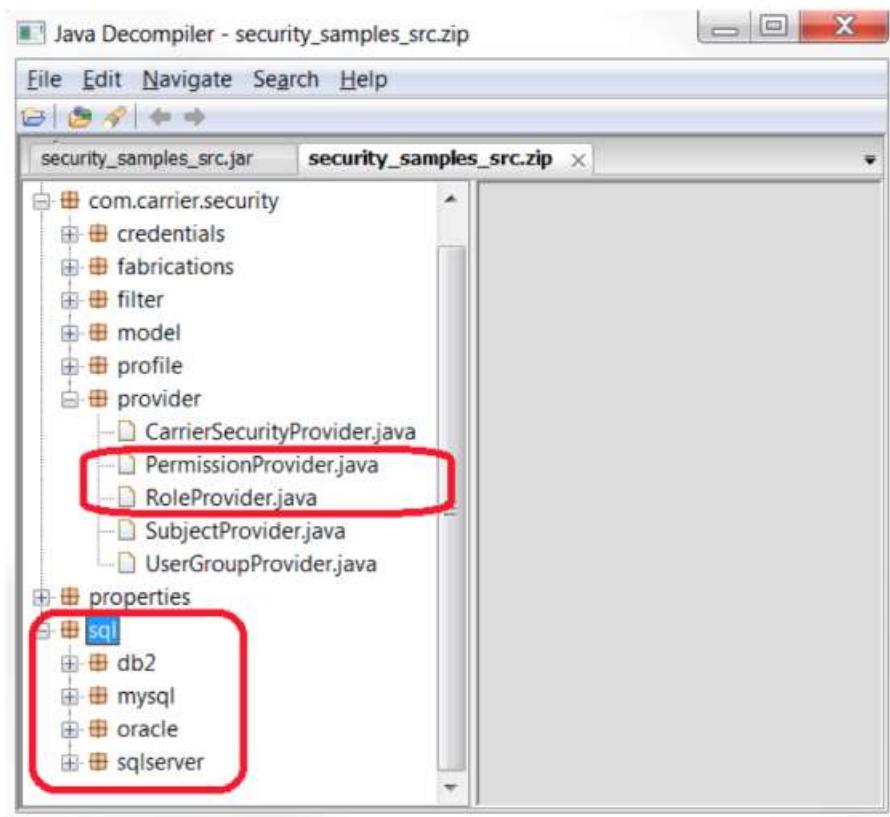
The following is a diagram of this simple data model, including the salient columns:



The following are the assumptions of this approach:

- There is one record in the role table for each unique role that can be returned by the carrier's security back end.
- Each role record(s) swept up for a given security profile will be based on one or more data values returned by the carrier's security back end. In other words, the back end needs to supply a data value that reflects the role the user participates in. This needs to be either in the form of an exact role ID, name match or some other value or set of values from which the role ID or name can be inferred.

The DDL and DOA code supporting this fine-grained permissions data model can be found in the sample security package zip.



## Implementing the Security Provider

### Serving Up Security Model Entities

An AgencyPortal web application needs to "connect" to a carrier's security system of records to inflate the security profile with various subjects, roles, permissions and user groups associated with the current user. There may be a few exceptions to this, but, generally, this is the rule of thumb. Since the administration of users, along with their respective role and work group associations, is typically administered at an enterprise level, one or more web service calls, API calls, active directory or LDAP model traversals are necessary to gain access to the various data values that constitute the security model entities contained within the security profile.

Regardless of how to access the data, the `com.agencyport.security.provider.SecurityProvider` interface implementation is where to locate this type of code. Resist the temptation to locate this type of code in the security profile manager. The closer you align with the API contract of the security provider interface, the easier it is to leverage those AgencyPortal components that, in turn, directly use your security provider interface.

The main function of the security provider is to "serve up" the security model entities (subjects, roles, user groups) that constitute the security profile.

One of the challenges you will most likely confront at first is the obvious mismatch between the carrier's back end security provider interface(s) and the 10 or so APIs that constitute the security provider interface.

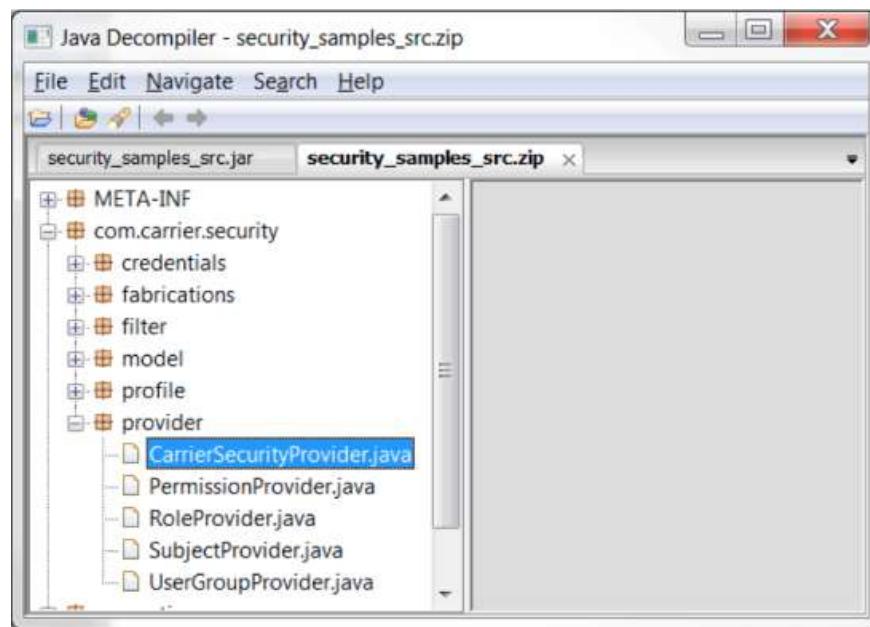
### Example

Your carrier's back end may bundle the notion of authentication and the mechanism for delivering back to the application into one single public interface. This interface serves as their single point of authentication and the mechanism for delivering back to the caller the user name, the role(s) they participate in and the work group(s) they are members of.

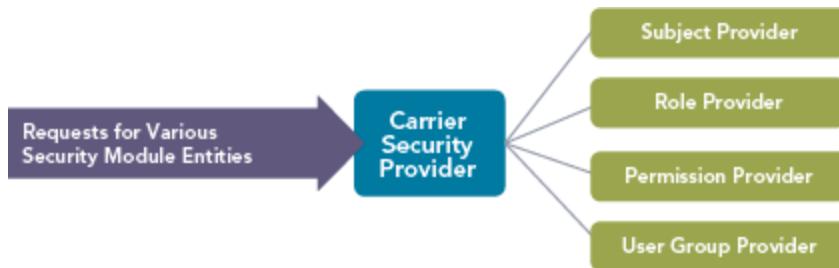
Some carrier back ends are limited to one interface call, but some may require a call to two or more interfaces. It is impossible to model a security provider interface that aligns perfectly with the granularity of all of the carrier security back ends. Refer to the [Supporting the Granularity of the Security Provider Interface API](#) section for more information.

### Supporting the Granularity of the Security Provider Interface API

The best way to describe the technique for achieving this goal is by reviewing the security provider interface implementation included in the sample security package:



A quick review of the security provider will reveal that this class serves as a facade between the application and the entity providers that do the real work. The following diagram illustrates this:



## Best Practice

Distributing the work across a set of entity providers is considered a good technique to adopt.

The sample security provider class contains some instance level data members that need explanation. Each sample security provider instance creates and holds onto its own subject provider, role provider and user group provider instance. Internally, the subject provider caches the most recently acquired subject, which it will use on subsequent requests when a request is made for a subject sharing the same identity.

To leverage this cache, the controlling unit in charge of the scope of the security provider needs to take this into account. Refer to the [Permission Provider](#) section for more information of this.

There may be some auxiliary methods that your security provider cannot honor. The implementation of those methods should throw an `UnsupportedOperationException`. You can inspect the sample for examples of these. The following is a list of security provider methods that you have no choice but to honor:

- `IPermission getPermission(String permissionName)` - getting a permission instance by name.
- `IRole getRole(String roleName)` - getting a role instance by name.
- `IRoles getRoles(Id subjectId)` - getting the roles for a specific subject.
- `ISubject getSubject(String loginId)` - getting a subject by their login ID. Typically, this is the first API exercised in an SSO handshake operation.
- `ISubject getSubject(Id subjectId)` - getting a subject by their internal numeric identity value.
- `IUserGroups getUserGroups(Id subjectId)` - getting the user groups for a specific subject.

## Permission Provider

The security provider does not contain a permission provider instance since the sample permission provider that supports the included fine-grained permissions model caches permission instances in a set of thread safe static data structures, allowing access to permission instances by name (or ID) without the need to create a permission provider instance. This is evident by the security provider method, `public IPermission getPermission(String permissionName)`, which delegates to the static method on the `PermissionProvider` class of the same signature.

## Subject Provider

This is probably the most important of all the entity providers since it is typically the first provider that is called during an SSO authentication handshake. You will notice that the sample subject provider supports access to subject instances by both login ID and subject ID, which results in the following two public methods:

- `public ISubject getSubject(String loginId) throws SecurityException;`
- `public ISubject getSubject(Id subjectId) throws SecurityException;`

These two methods directly support the `ISecurityProvider` interface methods of the same signature.

In reality, the carrier back end only supports access of the user by login ID. To support the retrieval of the subject by subject ID, perform a reverse lookup to reconcile the login ID from the subject ID and then call the `getSubject(String)` method, which calls the back end.

Refer to the [Numeric Identity Provisioning](#) section for more information.

## Numeric Identity Provisioning

Every security model entity, including the subject, role, permission and user group is composed of an identity and name. These are two separate and distinct attributes. The identity of the subject and user group is especially important because it comes into play with the `user_group_id`, `creator_id`, `owner_id`, and `owner_group_id` columns in the AgencyPortal work item table.

The `com.agencyport.id.Id` class supports both string and numeric based identities. However, consider the following recommendations/best practices before concluding that you only need to use string based subject identities, which also force you to alter the standard DDL that accompanies the work item table:

- Leveraging numeric identity values for subjects is considered a more secure practice than using plain text login IDs.
- Leveraging numeric identity values for subjects requires less customization to core AgencyPortal database schemas.
- Although most carrier security back ends do not return numeric values, the `com.agencyport.id.IdProvider` class provides a way to permanently associate a unique numeric value with a unique name. It also supports a reverse lookup mechanism so that a name can be derived from its unique numeric identity value.

The sample subject provider uses the `com.agencyport.id.IdProvider` class to support access to subjects (users) by both numeric identity and the string based login ID means. Unless your carrier security back end returns a unique numeric value for each user in the system, which is highly doubtful, you should consider adopting this technique.

The sample subject entity provider also contains a data member that stores the most recently retrieved subject to optimize the need to call the carrier's security back end repetitively.

## Role Provider

The sample role provider provides access to role instances by name and also supports the loading of the roles container for a particular subject via the following two methods:

- `public IRole getRole(String roleName, IUserPrincipal userPrincipalOfInterest) throws SecurityException;`
- `public IRoles getRoles(ISubject subject) throws SecurityException;`

These two methods support the following `ISecurityProvider` interface methods respectively:

- `IRole getRole(String roleName) throws SecurityException;`
- `IRoles getRoles(Id subjectId) throws SecurityException;`

Notice how the security provider makes use of the subject provider to first get the subject instance for a given before calling the `getRoles` method on the role provider:

```
public IRoles getRoles(Id subjectId) throws SecurityException { ISubject subject = subjectProvider.getSubject(subjectId); return roleProvider.getRoles(subject); }
```

This is why the subject providers need to cache the most recent subject retrieval.

## UserGroup Provider

The sample user group provider supports the loading of the user group container for a particular subject via the following method:

```
public IUserGroups getUserGroups(ISubject subject) throws SecurityException;
```

This method supports the following `ISecurityProvider` interface method:

```
IUserGroups getUserGroups(Id subjectId) throws SecurityException;
```

Again, notice how the security provider makes use of the subject provider to get the subject instance for the given before calling the `getUserGroups` method on the user group provider.

```
public IUserGroups getUserGroups(Id subjectId) throws SecurityException { ISubject subject = subjectProvider.getSubject(subjectId); return userGroupProvider.getUserGroups(subject); }
```

After you review the sample security provider and how it leverages the back end entity providers, it should become clearer how to map the security provider interface APIs on top of the set of back end security interfaces you are being asked to utilize.

## Extending the Security Profile Manager

### Inflating the Security Model

The security profile manager provides the mechanism for authenticating the user, inflating the security profile during an SSO handshake, as well as persisting the security profile to session. The security profile manager also serves as the single point for the acquisition of security provider instances.

The version of the AgencyPortalSDK with which you are working with dictates which class to extend from. Refer to the following best practices:

## Best Practice

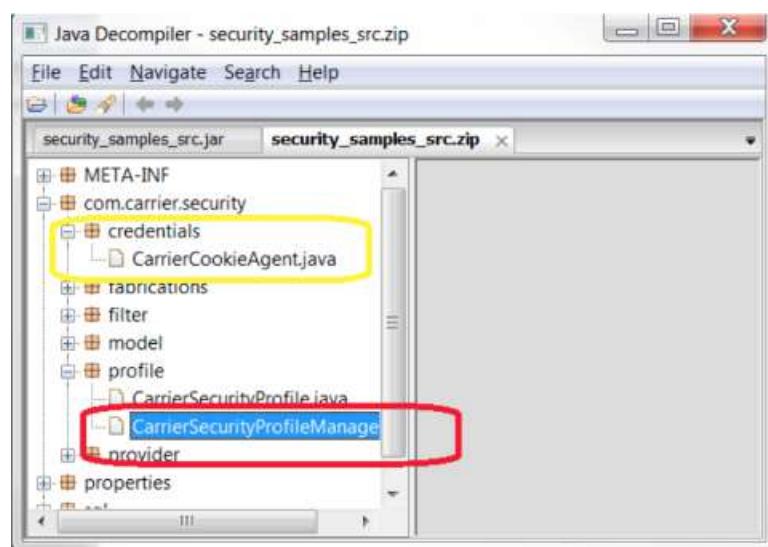
For versions 4.2.102 and greater, extend the `com.agencyport.security.profile.builtin.PortalSecurityProfileManager` class from `apwebapp.jar`.

For any earlier versions, extend the `com.agencyport.security.profile.impl.SecurityProfileManager` class found in the core security library.

**DO NOT extend the `BuiltinSecurityProfileManager` class found in the security reference library.**

If you are on a version of the SDK prior to 4.2.102, you are free to "lift" some of the methods from the `PortalSecurityProfileManager` class into your own manager class to satisfy the contract of the interface.

The sample implementation for this component is located in the samples security JAR as follows:



First, review the factory method for gaining access to security provider instances:

```
private final static ThreadLocalStorage<ISecurityProvider> securityProviderStore = new ThreadLocalStorage<ISecurityProvider>(); public ISecurityProvider getSecurityProvider() { ISecurityProvider securityProvider = securityProviderStore.get(); if (securityProvider == null) { securityProvider = CarrierSecurityProvider.create(); securityProviderStore.set(securityProvider); } return securityProvider; }
```

This particular technique uses a thread local data structure so that once a security provider is handed out on a given thread/request, subsequent access results in the same instances handed out. This allows the security provider to hold onto any request state from access to access. As long as the AgencyPortalSDK's security filter is "front ending" all of your application end points, the security filter will properly clear the thread local storage at the end of the HTTP request's scope.

## Note

It is important to remember that the security profile manager implementation follows a singleton design pattern, wherein one instance is shared across all threads, which precludes the possibility of adding instance data members to your security profile manager implementation.

Two methods that deserve review are the read subject (`readSubject`) and the authenticate subject (or login) methods.

The document reviews the read subject method first.

## Note

The `PortalSecurityProfileManager` class has a concrete `readSubject` implementation. However, if you extend this class, you should automatically override the `PortalSecurityProfileManager.readSubject` method since it cannot predict the peculiarities of how a subject is represented on an HTTP request in your application.

```
ISubject readSubject(HttpServletRequest request) throws SecurityException;
```

As the method name suggests, the main objective of this method is to inflate a subject instance from the incoming HTTP request, taking into account the various ways a subject might be represented on the request (i.e., Connect5 incoming request or SSO handshake request). The returned subject instance is considered unauthenticated and is "filled out" only to the extent needed to subsequently authenticate the user.

The implementation of this method varies drastically from application to application. The sample implementation's method is a total fabrication, except for the parts that deal with upload writer and URL pop requests initiated from Connect5. Discern the aspects of the method that are fabricated and those aspects that are not from the comments included in this class. This particular sample method hides the knowledge on how to extract the login ID from the request via the `CarrierCookieAgent.getLoginId()` method.

The second method of note is the method that is responsible for authenticating the subject. Again, the method you override depends on whether you are extending the `PortalSecurityProfileManager` or the `SecurityProfileManager` class. In the sample security profile manager, we are assuming a 4.2.102 SDK or above, hence the implementation of the `authenticateSubject` method. If you are using an SDK prior to that, you must implement the `login` method.

The main function of either of these methods is to return the authentication subject filled in with various properties. This subject instance is one stored in the security profile. Your implementation should call your security provider's `getSubject()` method to do this work.

If you are on an SDK version previous to 4.2.102, you must build out the following methods. The following table lists those methods and the potential suggested implementation:

Method	Suggested Implementation
<code>buildProfile</code>	<pre>public ISecurityProfile buildProfile(ISubject subject) throws SecurityException, AuthenticationFailedException, AccountLockedException, PasswordChangeRequiredException { ISecurityProfile securityProfile = create(); securityProfile.setSubject(subject); ISecurityProvider securityProvider = getSecurityProvider(); securityProfile.setRoles(securityProvider.getRoles(subject.getId())); securityProfile.setUserGroups(securityProvider.getUserGroups(subject.getId())); return securityProfile; }</pre>
<code>createUnauthenticatedSubject</code>	<pre>protected static ISubject createUnauthenticatedSubject(String loginId, String unencryptedPassword) throws SecurityException { SecurityModelFactory factory = SecurityModelFactory.get(); return factory.createSubject(loginId, unencryptedPassword); }</pre>
<code>logout</code>	Lift <code>PortalSecurityProfileManager.logout</code> from version 4.2.102 or above
<code>login</code>	Lift <code>PortalSecurityProfileManager.login</code> from version 4.2.102 or above
<code>getAgencyConnectUserCredentials</code>	Lift <code>PortalSecurityProfileManager.getAgencyConnectUserCredentials</code> from version 4.2.102 or above
<code>recoverSession</code>	Lift <code>PortalSecurityProfileManager.recoverSession</code> from version 4.2.102 or above

## Implementing the Security Provider

### Serving Up Security Model Entities

An AgencyPortal web application needs to "connect" to a carrier's security system of record to inflate the security profile with various subjects, roles, permissions and user groups associated with the current user. There may be a few exceptions to this, but, generally, this is the rule of thumb. Since the administration of users, along with their respective role and work group associations, is typically administered at an enterprise level, one or more web service calls, API calls, active directory or LDAP model traversals are necessary to gain access to the various data values that constitute the security model entities contained within the security profile.

Regardless of how to access the data, the `com.agencyport.security.provider.SecurityProvider` interface implementation is where to locate this type of code. Don't try to locate this type of code in the security profile manager. The closer you align with the API contract of the security provider interface, the easier it is to leverage those AgencyPortal components that, in turn, directly use your security provider interface.

The main function of the security provider is to "serve up" the security model entities (subjects, roles, user groups) that constitute the security profile.

One of the challenges you will most likely confront at first is the obvious mismatch between the carrier's back end security provider interface(s) and the 10 or so APIs that constitute the security provider interface.

## Example

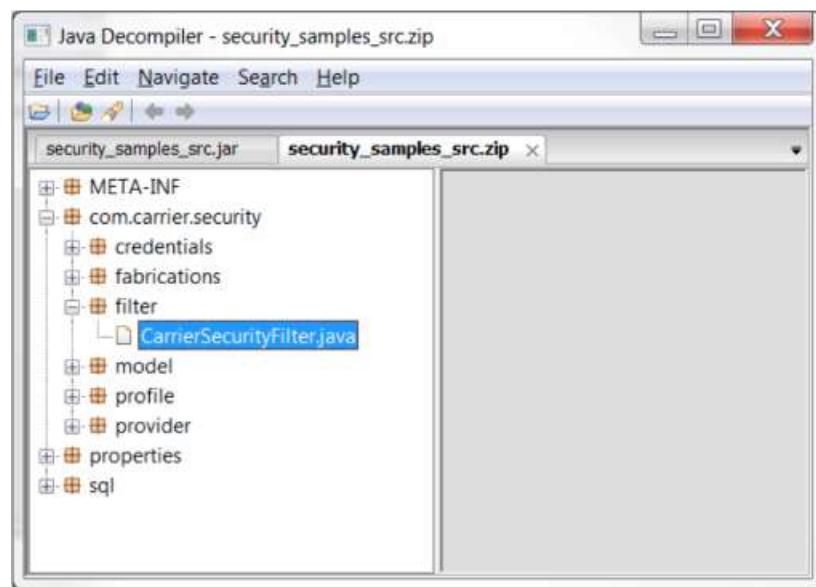
Your carrier back end may bundle the notion of authentication and the mechanism for delivering back to the application into one single public interface. This interface serves as their single point of authentication and the mechanism for delivering back to the caller the user name, the role(s) they participate in and the work group(s) they are members of.

Some carrier back ends are limited to one interface call, but some may require a call to two or more interfaces. It is impossible to model a security provider interface that aligns perfectly with the granularity of all of the carrier security back ends. Refer to the [Supporting the Granularity of the Security Provider Interface API](#) section for more information.

## Extending the Security Filter

### Best Practice

Extend the security filter (`com.agencyport.security.filter.SecurityFilter`) contained in the core security library.



The core security filter requires you to implement your own `handleSecurityException` method. This method is responsible for figuring out where to navigate the user whenever a security exception has been raised. The return value in almost all, but a few corner cases, should always be false, which prevents this filter from chaining to the filter list or the target servlet.

## Configuring the Security Filter

Configure the security filter in `web.xml` to guard as many edge points of the application server as possible.

There are two basic approaches to ensure that it is engaged on the appropriate edge points. The first one creates an explicit filter mapping for each end point to protect.

```
<filter> <description>AgencyPort Common Security Interface Servlet Filter</description> <display-name>ACSI Security Filter</display-name>
<filter-name>ACSI Security Filter</filter-name> <filter-class>com.carrier.security.filter.CarrierSecurityFilter</filter-class> </filter> <filter-mapping>
<filter-name>ACSI Security Filter</filter-name> <url-pattern>/FrontServlet</url-pattern> </filter-mapping> <filter-mapping>
<filter-name>ACSI Security Filter</filter-name> <url-pattern>/DisplayWorkInProgress</url-pattern> </filter-mapping> <filter-mapping>
<filter-name>ACSI Security Filter</filter-name> <url-pattern>/LaunchWorkItem</url-pattern> </filter-mapping>
```

The other technique uses a broader reaching, leveraging wild card in the filter mapping and a specific exclusion filtering feature built into AgencyPortal's core security filter implementation.

```
<filter> <description>AgencyPort Common Security Interface Servlet Filter</description> <display-name>ACSI Security Filter</display-name>
<filter-name>ACSI Security Filter</filter-name> <filter-class>com.carrier.security.filter.CarrierSecurityFilter</filter-class> <init-param>
<description>Exclusion filter is expressed as a list of tokens, each entry delimited by a semi-colon. The basic algorithm checks the request URI against each of these items and circumvents the usual security screening done against the request if the URI contains any one of the entries in this
```

```
list. </description> <param-name>security_exclusion_list</param-name> <param-value>Logon;js;gif;ico;.css;.png;.jpg</param-value>
</init-param> </filter> <filter-mapping> <filter-name>ACSI Security Filter</filter-name> <url-pattern>/*</url-pattern> </filter-mapping>
```

There is an obvious advantage to the latter technique. As new end points are introduced during development, the security filter will be engaged by default. In the first technique, each new end point requires an explicit addition to the web.xml to engage the security filtering. The latter technique is the preferred technique.

## Secure Resources

### Bringing Your Security Filter to Life

If you have no secure resources registered in your application, then the security filter does not add value to your pursuit of guarding against URL spoofing/tampering.

### Reviewing How Security Filter Works

Before the core security can make any assertions on the legitimacy of an incoming request (with respect to the user that initiated said request), it must bring the security profile into scope. The method that it calls to achieve this is the `ISecurityProfileManager.acquire(HttpServletRequest request, HttpServletResponse response)` method. After the security profile is acquired, the security filter engages each registered security resource implementation as follows:

#### Note

It is important to note that there is redundancy between the roles and permissions paths in the above interaction processing model. If your application embraces explicit fine-grained permissions, which hopefully it does, you can ignore anything to do with role identity based assertions.

Iterate through all of the secure resources, keeping track of the ones that recognize the current HTTP request. Iterate through all of the secure resources that "said" they recognize the current HTTP request { Ask the secure resource for the set of roles that the current user must possess at least one of to execute this request via the `Set<IRole> ISecureResource.getValidRolesNeededForExecutingRequest()` method. If the secure resource returns a non empty set of roles Check that the current user posses at least one of the roles by invoking the method `ISecureProfile.checkIsInOneOfRoles(Set<IRole>)`. This will throw an exception if the user does not participate in one of the roles. Endif Ask the secure resource for the set of permissions that the current user must possess all in order to execute this request via the `Set<IPermission> ISecureResource.getPermissionsNeededForExecutingRequest()` method. If the secure resource returns a non empty set of permissions Check that the current user possess all of the permissions by invoking the method `ISecureProfile.checkPermissions(Set<IPermission>)`. This will throw an exception if the user does not have all of the permission. Endif Ask the secure resource for the data item id (in a portal application work item id), which the current user must have the authorization to access via the `ISecureResource.getDataItemIdAssociatedWithRequest()` method. Note that the iteration through the secure resources will stop at the first secure resource that returns the first non null id value. If the secure resource returns a non null Id Check that the current user possess rights to access this data item by invoking the method `ISecureProfile.checkDataAccess(Id)`. This will throw an exception if the user does not have rights to access such data item. Endif }

### Implementing a Secure Resource

Each secure resource class must implement the `com.agencyport.secure.resource.ISecureResource` interface. The methods of this interface are bolded above. The AgencyPortal SDK comes with two concrete implementations for you to consider leveraging in your application before jumping in and writing one entirely from scratch. This section details the `com.agencyport.secure.resource.SecureFrontController` implementation. It provides the following features:

- Recognizes any HTTP requests that are targeting the `FrontServlet` servlet.
- Supports role based authorization of the LOB associated with the incoming `FrontServlet` request. Again, if you are using fine-grained permissions, this is not activated.
- Supports permission based authorization of the LOB associated with the incoming `FrontServlet` request.
- Supports permission based authorization of the work item open mode on the incoming `FrontServlet` request.
- Supports the extraction of the work item ID for subsequent data access authorization.

The second out of the box secure resource comes into play if your application leverages the menu bar in your application driven by AgencyPortal's own menu and menumember tables. The underlying class,

`com.agencyport.secure.menu.model.impl.MenuGroups`, verifies that the current user has an explicit permission for end point for the given menu entry.

Each secure resource has its own factory class that is responsible for bringing the secure resource instances to life. Secure resource instances are singleton in nature and cannot be used to hold onto request based data values.

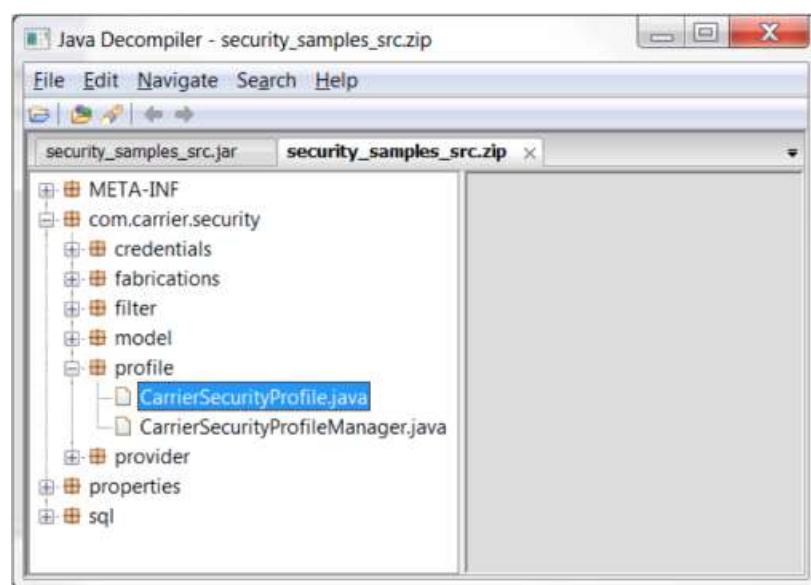
You may decide to write your own secure resource implementation to supplement one or both of the out of the box stock secure resource implementations. To do this, create a class that implements the `com.carrier.secure.resource.ISecureResource` interface. Then, create a factory class that implements the `com.carrier.secure.resource.ISecureResourceFactory` interface. For the framework to recognize your list of factories, you must configure the factory list in `aci.properties` as follows:

```
secure_resource_factories= com.agencyport.secure.menu.provider.SecureMenuFactory;\\
com.agencyport.secure.front.SecureFrontControllerFactory
```

## Extending the Security Profile

### Best Practice

Extend `com.agencyport.security.profile.builtin.PortalSecurityProfile` if you need to customize the security profile interface. It provides a reasonable implementation of the `com.agencyport.security.profile.ISecurityProfile` interface.



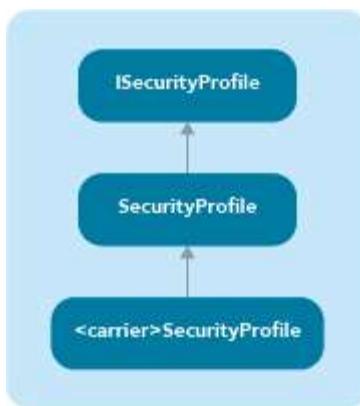
The security profile's main function is to serve as the container for the security model. However, it also contains several methods that are engaged as a result of the collaboration between the security filter and the set of registered secure resources. To support the requirement in our fabricated security implementation, which states "Internal users are grouped as one entity and have access to all work item through the entire application regardless of the user group the work item was created in," override the `checkDataAccess()` method in the portal's security profile base class as follows:

```
public void checkDataAccess(Id dataItem) throws DataAccessDeniedException { CarrierSubject subject = (CarrierSubject) this.getSubject(); if (subject.isInternalUser()) { // All internal users can see ALL work items return; } else { //The super class checks to see that the user group associated// with the current work item is one of the user groups associated// with the current user. super.checkDataAccess(dataItem); } }
```

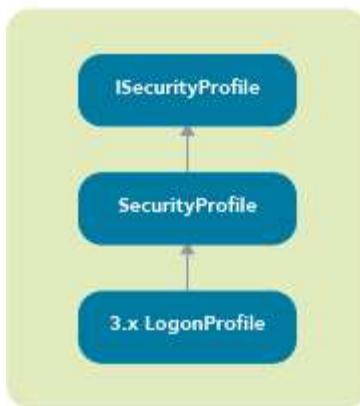
# Upgrading from 3.x

Consider the following guiding principles regarding security if you upgrade from a 3.x application to 4.x+:

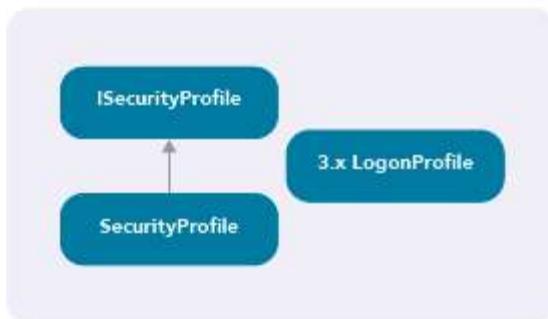
- If the application you are upgrading has its own security system of record and associated administrative functions, the following guiding principles apply:
  - Leave the administrative DAO security functions and their data structures intact.
  - Do not attempt to modify the old administrative DAO functions to accomplish some sort of parallel update bridging between the old security schema and the security reference schema. Remember that the security schema or library should not even be coming into play in any way.
  - Design your entity providers to function as read only access mechanisms to inflate the various security model entities.
- Your 3.x application probably used a `LogonProfile` and 4.x requires a `SecurityProfile`. Preferably the old `LogonProfile` interface should be retired; however, you can extend the 4.x profile class from the `LogonProfile` if necessary. Make sure you don't support both the old `LogonProfile` and the new security profile as two discreet unrelated artifacts. This will lead to confusion in future maintenance.



**Preferable**



**Acceptable**



**Not Acceptable**

- Custom 3.x security filters should be converted to use the 4.x ACSI security filter and corresponding secure resource facilities.
- Make sure the project plan has enough time allocated to accommodate the time required to design the data conversion strategy and any routines that are necessary to implement, which are going to be run to convert any existing production data records.

# Checklist

The following is a checklist of items that you should evaluate against the ACSI implementation of your application:

## Best Practice

The security reference library should not accompany any production application.

- Check your application for the inclusion of `apsecurityreference.jar`. Remove it from your application's library dependency list and see if your application still compiles. If it does not, figure out a way to remove those dependencies. The only exceptions are 4.0 and 4.1 applications that may need the `CookieAgent` or `BuiltinCredentialAgent` classes. As of 4.2, the functionality in these classes was moved to the core security library so that future references to the security reference library can be permanently severed.
- **Forwarded requests are not routed through security filters by the web container.** This is a little known fact. If you have an end point not guarded by the security filter that forwards a request to a server registered to be guarded, you might think you are in good shape; however, in reality, neither servlet is protected. Keep this in mind when making decisions to exclude edge servlets from the guard of the security filter that deletes to other functionality using the `forwardRequest JEE` method.

## Best Practice

It is preferable to use numeric identity provisioning with the `IdProvider` class than changing the data types for those columns on the `work_item` table that track ownership and creatorship.

The only tables that should be updated on the fly during authentication are the `id_store` and `NextKeyValue` tables in support of numeric identity provisioning. If you find that you are updating other tables, perhaps in the domain of the security reference data model, it is likely you are going down the wrong path.

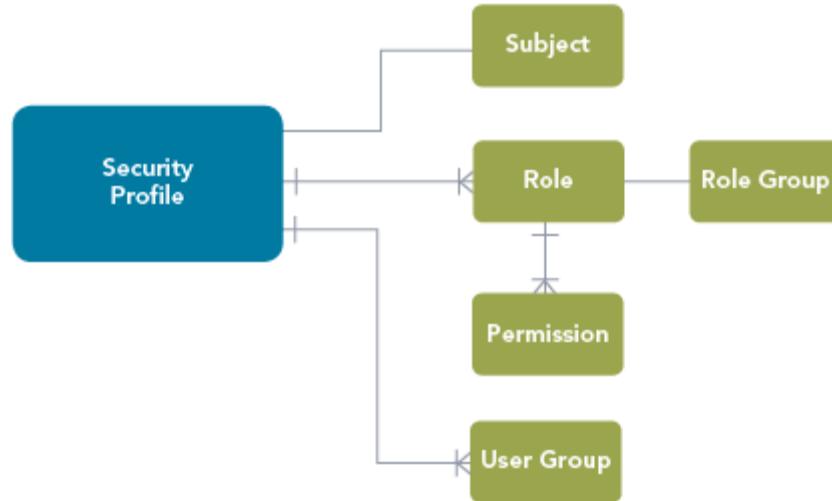
- Until further notice, application teams are being asked to review their security implementations with product development. There are two main milestones:
    - Reviewing the design of their security implementation before construction begins.
    - Reviewing the implementation after construction is done. The plan should accommodate enough time to make adjustments if they are deemed necessary.
- It is the responsibility of the tech lead and project manager to ensure that these reviews are carried out.
- Typically, ACSI implementations should take days and not weeks. If an implementation is taking longer than expected, contact the product solutions department for assistance.
  - For customers that also use Connect5, the portal ACSI implementations should be JAR packaged up to accommodate easy integration with Connect5's out of the box ACSI based authentication plug-in.

# ACSI

The ACSI layer is a set of interfaces that standardizes how a security model is represented, how entries that compose a security model/profile are brought to life and the set of supporting, collaborating entities that are responsible for responding to various authentication and authorization assertions on behalf of the current user.

The following diagram illustrates a basic security profile and its underlying security model:

## ACSI Model



### Subject

A subject refers to a user in this context. A subject has a name, user ID (by way of a principal) and a set of credentials.

### Role

A role represents a collection of permissions related to a subject by way of its identity. A role has, in many cases, assumed an intrinsic meaning in and of itself and has, historically, been used as the only entity in making authorization assertions ('is user in role?').

A role provides an easy way to maintain permissions across a larger group of users without having to maintain permissions at the user level. An "agent that is licensed to sell personal lines" is an example of a role.

One of the benefits of roles in security models is that it provides a short hand way of maintaining permissions across a large number of users without the need to visit the permissions that were individually assigned to each user when a broad level change is involved.

### Role Group

A role group provides a way to share a role across multiple subjects. A personal lines agent is an example of a role group.

### User Groups (Work Groups)

A user group is an aggregation of one or more subjects sharing a common business interest, typically within a common locale. This influences data ownership, visibility and, ultimately, access to data/work items.

### Example

Examples of user groups are:

- retail agency ABC
- carrier XYZ

User groups can be represented hierarchically, such as super agencies, agencies and sub agencies.

### Permissions

A permission represents the right to carry out a specific action or execute a specific type of request.

## Example

"Does the current user have the right to bind/issue policies?"

"Is this agent licensed to sell a particular LOB?"

Some security models view permissions as optional. The 'isUserInRole' functionality is viewed by some as adequate in many cases.

Permissions granted to the user are the union of the permissions granted through each role that the user participates in, as well as the permission directly assigned to the user. Permissions are typically related to a role that has a related role group with a distinct name, but a user can also be directly assigned permissions using a role specifically allocated for the user (in which case, there is no role group name).

Negative permissions are not currently supported.

One of the main considerations facing application teams is whether to embrace permissions. It is the expressed opinion of Agencyport's product development team that using permissions for authorization runtime assertions is a superior technique to inferring said authorizations from a role name or role identity by itself. Although ACSI supports authorization assertions based solely on role name or identity for backward compatibility, we believe the use of permissions is a more flexible, maintainable and sustainable approach. For instance, the following code is trying to verify whether the current user has the right to bind a policy. Assume that there are three roles: agent, superagent and underwriter; the two latter can bind policies.

Role-based approach (inferior):

```
if (roles.isUserInRole(principal, "superagent")||roles.isUserInRole(principal, "underwriter")){ // user can bind policies } else { // user cannot bind policies }
```

Permission-based approach (superior):

```
if (roles.checkPermission(principal, "canBind")){ // user can bind policies } else { // user cannot bind policies }
```

The latter approach makes the assumption that the `canBind` permission was created and attached to the role of underwriter and superagent at security profile creation time. The advantage of this is that the main line code does not care about the role per say, but just that one of the roles included in the current security profile contains a permission `canBind`. If the particular physical security provider at the application's disposal does not serve up discreet permissions, then the following logic is necessary when building the profile for the current user:

```
IRoles roles = securityProfile.getRoles(); IRole role = roles.getRoleByName(principal, "superagent"); if(role == null) role = roles.getRoleByName(principal, "underwriter"); if (role != null) { SecurityModelFactory factory = SecurityModelFactory.get(); IPermission canBind = factory.createPermission(new Id("canBind"), "canBind"); role.addPermission(canBind); }
```

## Security Profile

A container of the roles that a unique subject participates in and the user groups to which he/she is a member.

## Identity:

All of the security model entities, including subjects, roles, permissions and user groups, are comprised with both an:

- identity, and
- name

Identity is supported by the AgencyPortal Java class `com.agencyport.id.Id`, which supports both numeric and string-based identity. The important thing to note is that, internally at runtime, the property is what is leveraged to satisfy low level equality and hashing operations. All of the interfaces for these entities support alternate access mechanisms by name; thereby, making the coding effort easier at the application level.

## Security Object Instantiation

ACSI embraces a robust factory design pattern for all of the security model entities and supporting class implementations, making it unnecessary to "hijack" product code into your project as necessary. If you find you are still in the habit of hijacking Agencyport product class files into your project, then you are probably doing something fundamentally wrong.

# Application Layer

The application layer depicts the originator of security information requests. Security information requests are made upon the ACSI abstraction layer. The following are the types of requests typically made:

- Read the subject whose credentials are attached to an HTTP request (the subject in this case is viewed as the user of the application)
- Authenticate the subject (against the back end security model/provider)
- Build the security profile for an authenticated subject (the subject profile is the container for the subject, including the roles they participate in, and the permissions)
- Persist the security profile to session
- Acquire the security profile from session
- Check whether the current subject participates in a specific role currently under consideration
- Check whether the current subject possesses a specific permission currently under consideration
- Check whether the current subject is a member of a user (work) group currently under consideration
- Retrieve a permission by name
- Retrieve a role by name

# Security Adaptor Layer

The security adaptor layer is the bridge or translation layer between the ACSI abstraction layer and the physical back end security provider/model. This is the realm that application teams tend to concentrate on more than others. The following are the main interfaces that migrating application teams need to consider:

- **Security Filter** - ACSI comes with an almost 100% complete security filter implementation. It evaluates HTTP requests to ensure the request is originating from an authenticated user. It verifies that the user is authorized for both the action requested and the data item that the action acts upon.
- **ISecureResource** - the standard interface used by the security filter to verify authorization assertions for incoming HTTP requests. Part of a secure resource's responsibility is to determine whether the incoming HTTP request is recognized by the secure resource implementation. If it recognizes the request as one it can "understand," then it responds with a true to the following method:

```
boolean recognizesRequest(HttpServletRequest request, ISecurityProfile securityProfile) throws SecurityException;
```

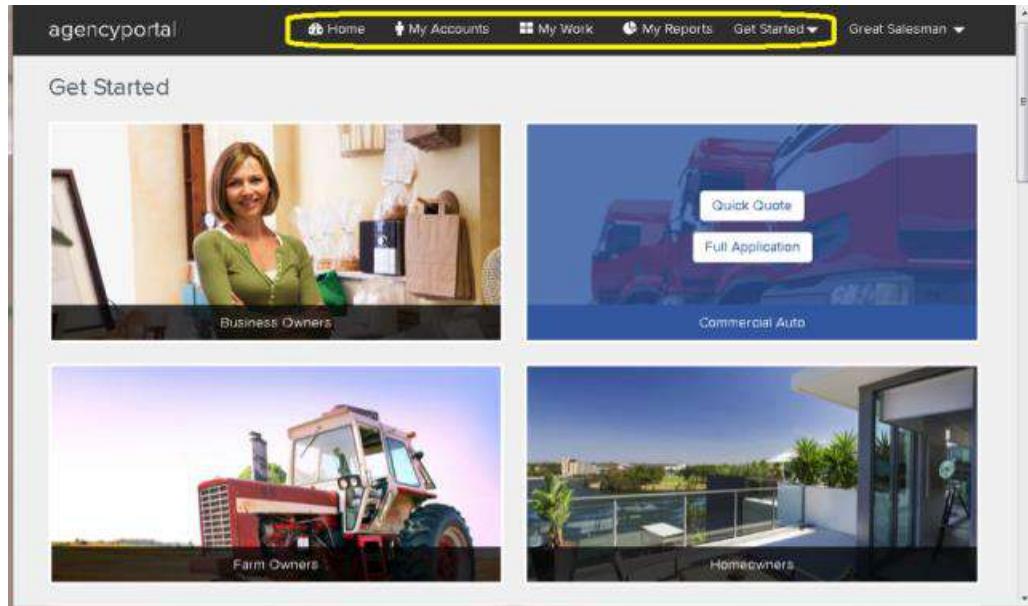
After a particular secure resource implementation recognizes the inbound request, it is then called upon to serve up the roles and/or permissions necessary to exercise said function. It is also called upon to parse the data item's identity (a.k.a, work item ID) from the request. The following interface methods are the methods that provide this functionality:

```
Set<IRole> getValidRolesNeededForExecutingRequest(HttpServletRequest recognizedRequest, ISecurityProfile securityProfile) throws SecurityException; Set<IPermision> getPermissionsNeededForExecutingRequest(HttpServletRequest recognizedRequest, ISecurityProfile securityProfile) throws SecurityException; Id getDataItemIdAssociatedWithRequest(HttpServletRequest recognizedRequest, ISecurityProfile securityProfile) throws SecurityException;
```

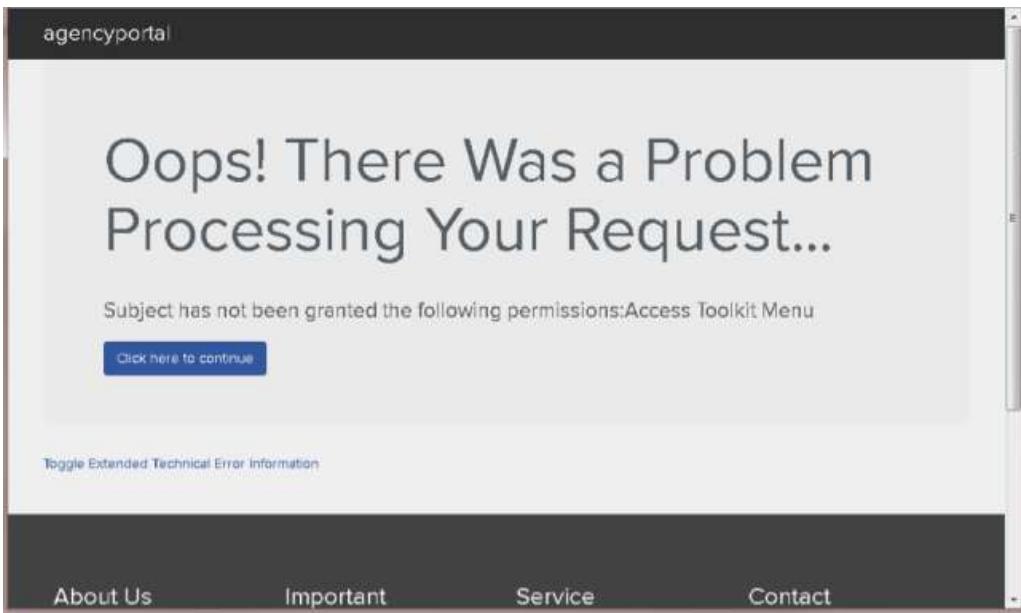
Applications will probably need more than one secure resource implementation to satisfy the various end points of an application. AgencyPortal comes with two basic security resource implementations that you can use as is or extend for modification. Each secure resource must have its own factory. Accordingly, each factory is registered in application properties under the following name:

```
secure_resource_factories=com.agencyport.secure.menu.provider.SecureMenuFactory;\ncom.agencyport.secure.front.SecureFrontControllerFactory
```

AgencyPortal's own `SecureMenuFactory` is responsible for the instantiation of the secure resource, which is responsible for verifying menu selections on the main menu bar against the current user's security profile. The underlying metadata for the menu is driven from the menu and menumember tables that all applications are free to add additional menu items. The visual manifestation menu is rendered by `menu/menu.jsp`.



For each menu selection to display, a corresponding permission must be contained in the security profile of the current user. From a security filtering standpoint, if a user does not have the security permission to only see the Home and My Work menu options, but attempts to URL spoof the Toolkit function, the following security exception will result:



About Us

Important

Service

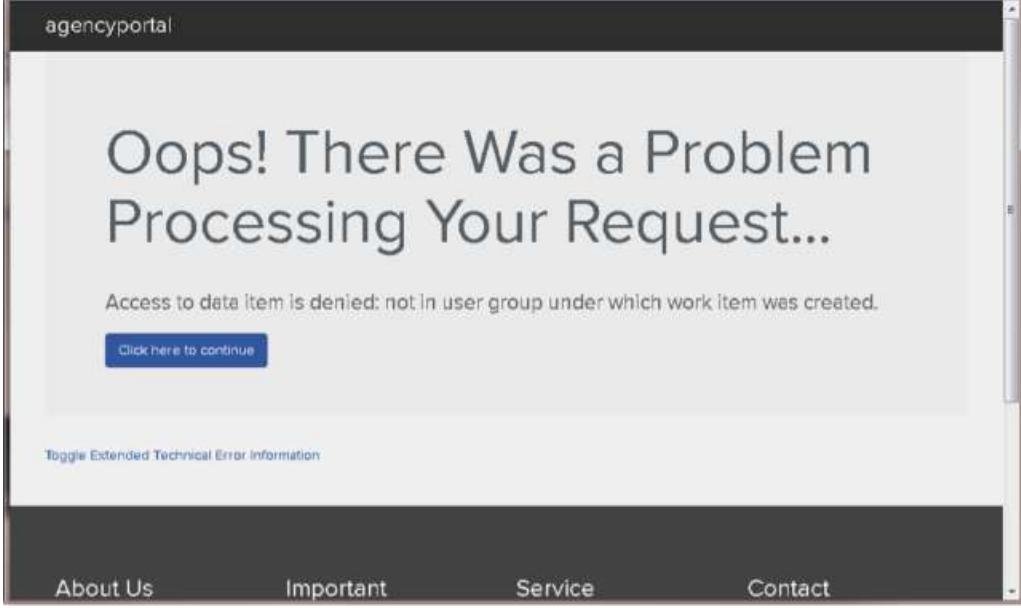
Contact

An audit record is also posted, which documents which user attempted something that was against the law. For example:  
26 front\_door\_access\_tracking 1000 subjectId Great Salesman access 0 0 urlId  
<http://localhost:8080/pompey/Toolkit> 903 permissionId Access Toolkit Menu 2009-05-27  
10:46:15.237 200 Example Agency Subject has not been granted the following permissions:  
Access Toolkit Menu

The other built-in secure resource verifies incoming FrontServlet requests to ensure:

- The current user is authorized for the LOB (indirectly via the transaction name) in context
- The current user is authorized to see the work item in context

If a user attempts to URL spoof a work item they are not authorized to view, the following exception displays:



About Us

Important

Service

Contact

- **ISecurityProfile** - every application is required to provide a security profile that, in turn, holds onto the security model for every authenticated user logged into the system. This is the standard interface for this function; it acts as the container for the security model for the authenticated logged in user and standardizes how the security model for the current user is exposed to the outside world. It acts as the provider of secure resources to the security filter via the `getSecureResources()` method. It also responds to functionality authorization and data access authorization requests from the security filter, such as pertaining to the following security profile methods:

```
void checkIsInOneOfRoles(Set<IRole> roles) throws RoleCheckFailedException; void checkPermissions(Set<IPermission> permissions) throws PermissionsDeniedException;
```

The security profile class name is configured in application properties under the `security.profile_classname` name.

#### Example

```
security.profile_classname=com.agencyport.security.profile.builtin
.PortalSecurityProfile
```

- **ISecurityProfileManager** is the interface that is responsible for parsing the HTTP request, authenticating the user's credentials as represented in the HTTP request and acting as the primary agent for creation and initialization of the security profile instances. It interacts internally with the security provider for access to the back end security model for the purposes of building the security model into the security profile for the current authenticated user. The security profile manager also acts as the factory for the underlying security profile.

# Security Provider

The security provider layer has intimate knowledge of the underlying security repository. Its main function is to render ACSI security model entities (subjects, roles, permissions and user groups) from the carrier's selected security system of record.

The application team has direct responsibility for transforming the entities served up by the carrier's security back end, regardless of the protocol, into those ACSI security model entities. The security back end could be an exposed LDAP or Windows Active Directory model, a relational database, such as SQL Server, or a set of proprietary web services.

The `ISecurityProvider` interface defines the standard mechanism for the acquisition of security model entities. It contains all of the inner workings and knowledge on how to call the back end security system of record. It acts as the general supplier of the security model entities on behalf of the security profile manager and other places in the product that may need said entities, such as subjects, roles, permissions and user groups. It is internally leveraged by the base security profile manager to build the various pieces of the security model.

## Important

Application teams must implement the entire interface from scratch since there is no abstract class to extend.

### Basic Steps to Make Your Application ACSI Compliant

- Implement the security provider interface: `com.agencyport.security.provider.ISecurityProvider`

## Best Practice

Relegate the use of roles as merely containers for permissions. If the back end security provider serves up permissions, then use them. If the back end security provider doesn't expose permissions explicitly, build a suitable model based on the role at the time the security model/profile is constructed.

- Extend the abstract security profile manager class:  
`com.agencyport.security.profile.impl.SecurityProfileManager`
- Register class name of security profile manager as an application property: `security.profile_manager_classname`

## Best Practice

Extend the above abstract class rather than implementing the interface from scratch.

- Extend the security profile abstract class: `com.agencyport.security.profile.impl.SecurityProfile`
- Register class name as an application property: `security.profile_classname`
- Extend the security filter abstract class: `com.agencyport.security.filter.SecurityFilter`
  - o Configure security filter in web.xml as appropriately
  - o Implement new and/or re-use existing secure resource functionality as appropriate
    - Register factories for secure resources as an application property list under the name of `secure_resource_factories`

## Best Practice

All AgencyPortal applications should leverage the 4.0 security filter to protect against unauthorized URL spoofing. If you currently have a 3.x security profile implementation, consider to adopt the product's own security filter as the standard and customize only if necessary. If you currently don't use a security filter, seriously consider adopting the 4.0 implementation.

AgencyPortal comes with a security filter implementation that only lacks the `handleSecurityException` method. This method is in charge of navigation details when a security exception is raised and is left up to the application to fill in. Applications should extend AgencyPortal's `com.agencyport.security.filter.SecurityFilter` abstract base class rather than creating a custom security filter.

## Best Practice

Decisions to use the security implementations in product applications needs to be approved by product development. Unless the specific application is using the security reference implementation, which is a VERY rare occurrence, the application should never have a compile or runtime dependency on `apsecurityreference.jar`. You should rarely deploy this JAR file.

# Security Reference Implementation

AgencyPortal is accompanied by a full reference implementation of ACSI with its own proprietary relational security model. The runtime support for this implementation is contained in `apsecurityreference.jar`. This implementation supports the full specification of ACSI and embraces the notion of explicit permissions fully. Decisions by application project teams to use the security reference implementation in whole or part must first get product development approval. Production portal implementations should never have any dependency on this JAR file or have a need to deploy this component.

# Additional Security Considerations (OWASP Top 10)

Included in this section are some additional security considerations that applications need to address. These considerations are based on the top 10 concerns brought to the computing industry by the Open Web Application Security Project (OWASP) initiative (refer to the [OWASP Top 10](#) site for more details), as well as some addition security policy content.

## Important

It is important for project managers and tech leads on all application projects to understand all of these concerns and how their application, along with the AgencyPortal framework, can minimize exposure to each of these potential threats. This is especially important for any custom developed pages or dialogs (light boxes) that may not automatically inherit the protective agents of the product.

The following sections include a short explanation of each concern followed by how applications can safeguard against threats of that nature.

## SQL Injection

Injection flaws, such as SQL, OS and LDAP injection occurs when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization. To safeguard against this threat, ensure that all parameters in all JDBC queries and insert/update/delete commands are bound to SQL statements using the safe API `PreparedStatement.setXXX()` or `CallableStatement.setXXXX()` methods.

## Broken Authentication and Session Management

Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys or session tokens, or to exploit other implementation flaws to assume other user's identities. To safeguard against this threat:

- Ensure that all HTTP requests flow through the ACSI security filter except for browser requests that are requested static content, such as JavaScript, CSS, images, etc. This filter authenticates that each request is coming from a trusted source. This is standard if the security interfaces (ACSI) were correctly implemented.
- Configure session timeouts as short as the customer can withstand. This should be negotiated with the customer since this configuration falls within the jurisdiction of their own systemadministration group. Ensure all cookies have the HTTP only flag set to true so that client side JS cannot get its hands on the session ID or any other cookie.

## Cross-Site Scripting (XSS, aka Safe Strings)

XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser, which can hijack user sessions, deface web sites or redirect the user to malicious sites. To safeguard this threat:

- Ensure that all web content is properly encoded. This is important on any pages that are whole or in part rendered by non-product (custom) logic. Areas of concern are custom quote/policy summary pages, light boxes or any other content that might contain one or more of the following control characters:
  - double quote
  - ampersand
  - less than symbol
  - greater than symbol
  - equals sign
  - forward slash
  - back slash
  - semi-colon
  - apostrophe
- The following are several notable product supplied APIs that should be used to ensure that these control characters are properly escaped:
  - `JSPHelper.prepareForHTML()` - this is a sever side API that should be used in the presentation layer to properly escape character sequences for HTML.
  - `JSPHelper.prepareForJavaScript` - this is a server side API that should be used in the presentation layer to properly escape character sequences for JavaScript.
  - `ap.htmlEncode` - this is a client side API that should be used in the presentation layer to properly escape character sequences for JavaScript.
  - `JDK URLEncoder.encode()` - used to properly encode URLs.

- As with any type of encoding, you must make sure you do not double encode a character sequence. In cases, such as a sequence where 'my friend's' would display as 'my friend's'. Most product data APIs do not encode content.
- Testing XSS vulnerabilities can be achieved both from automated and manual contexts. Neither approach should preclude the other. The AgencyPortal team uses a product called ZED Attack Proxy for automated XSS testing.

## Insecure Direct Object References

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data. The product verifies the following parameters:

- TRANSACTION\_NAME
- PAGE\_NAME
- WORKITEMID
- CSRF\_TOKEN

If the application invents any custom URL GET parameters that the user could potentially spoof, then additional safeguards are necessary. Engage verification as early in the request processing (by the security filter) as possible. The application can create a new `ISecureResource` implementation for the purpose of validating URL parameters. The `ISecureResource` method most appropriate for carrying out this task is the `ISecureResource.getPermissionsNeededForExecutingRequest()`. In this method, you will have access to the HTTP request, along with all of the parameters therein.

## Security Misconfiguration

Good security requires that you have a secure configuration defined and deployed for the application, frameworks, application server, web server, database server and platform. Define, implement and maintain secure settings since defaults are often insecure. Additionally, keep all software up to date. This concern highlights the need for:

- Hardened production servers, including database, application and web server installations. Store any secret cryptographic keys in key store files rather than application properties.
- Specific review procedures to ensure that the application and runtime libraries, which it depends on, has undergone careful scrutiny. This includes procedures with the following application security verification techniques:
  - Automated Verification
    - Dynamic Scan (Partial Automated Verification) OWASP ZED Attack Proxy or some other equivalent tool
    - Source Code Scan (Partial Automated Verification) SONAR code analysis
  - Manual Verification
    - Penetration Test (Partial Manual Verification)
    - Code Review (Partial Manual Verification)
- Automated repeatable process for deploying application to various environments (dev, QA, UAT, prod, etc).

## Sensitive Data Exposure

Many web applications do not properly protect sensitive data, such as credit cards, tax IDs and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft or other crimes. Sensitive data deserves extra protection, such as encryption at rest or in transit, as well as special precautions when exchanged with the browser. To safeguard against this threat, the application should ensure that the following guidelines are followed:

- All production application network protocols should use SSL and never consider straight HTTP as an option.
- Applications must select specific client sensitive XPaths, which should be encrypted when the data is at rest. The configuration that communicates to the framework which fields to encrypt is through a list of XPaths configured on the `application.encryption_fields` application property.

### Example

```
application.encryption_fields=InsuredOrPrincipal.GeneralPartyInfo.NameInfo.TaxIdentity.TaxId;CommlAutoLineBusiness.CommlDriver.DriverInfo.License.LicensePermitNumber
```

We also provide the ability to specify encryption fields via properties to support each LOB (e.g., `WORK.encryption_fields` or `ACCOUNT.encryption_fields`).

By default, encryption fields are excluded from a work item copy action. The `copy_encryption_data` LOB property allows project teams to override the default behavior. If there is a need to dynamically determine whether encrypted data should be included or excluded during work item copy using custom Java code, a custom `CommandCopyImpl` implementation can override the `shouldCopyEncryptedData` method.

## Secure Fields

AgencyPortal also provides the ability to mask secure fields in the browser to prevent unnecessary user access. Secure fields are configured in the TDF (refer to the [Transaction Definition/Page Library](#) topic for more information).

For more information on securing fields and masking field data, refer to the [Field Security](#) topic.

## Missing Function Level Access Control

Most web applications verify function level access right before making that functionality visible in the UI; however, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers can forge requests to access functionality without proper authorization. The product asserts that the current user has the authority to execute the following actions on work items:

- Open
- Copy (work items only)
- Merge (accounts only)
- Delete
- Move (work items only)

## Verification of Work Item Actions

Request level verification of work item actions is carried out to protect against the unauthorized execution of actions against work items. This is in support of the OWASP Top 10 #7 tenet titled: A7 - Missing Function Level Access Control.

### Supporting Configuration

The configuration that governs the verifications of these actions is contained in acsi.properties under the `security.verify_work_item_actions` and `security.action_to_permission_map` property names. If the application invents new actions, then those actions should be handled as explicitly as the above actions. In general, the application can never assume that the current HTTP request is properly constructed with the security constraints of the current user.

Verification of work item actions is controlled by application property configuration. The following are the various configuration options:

```
#####
ACSI - Work Item and Account Action Verification Configuration
the framework will verify that user has the authority to execute the action on work items and accounts.
#####
Setting the following flag to false will deactivate work item action verification.
security.verify_work_item_actions=true # The following map relates incoming work item action values to their permission name counterparts. This is used to verify that user has the authority to execute # the incoming action on the work item.
security.action_to_permission_map=MoveWorkItems,accessCanMove;\MergeAccount,accessCanMerge;\Delete,accessCanDelete;\DoAssign,accessCanAssign;\Copy,accessCanCopy
```

## Cross-Site Request Forgery (CSRF)

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie, and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim. Refer to the [Cross-Site Request Forgery \(CSRF\)](#) site for a detailed explanation of this security topic.

As security expectations continue to escalate by the security officers at the insurance carrier, omitting support for this in our product will be harder and harder to define as time goes on.

### Basic Approach

The technique for protecting against CSRF is based on a mechanism that dispenses a unique, secure, hard to guess identifier to the browser when the content is rendered. When a subsequent request (HTTP GET or POST) is made on the server, one of the first things done by the security filter is to check that the incoming identifier matches the identifier on record. If the identifier is missing or does not match the one on record, then a security exception is raised, an audit trail record is written to the database and the user is directed to an error page.

### Technical Design

#### Dispensing of CSRF Tokens

One technical approach maintains one single unique identifier at user/session scope that is dispensed over and over again with each request. Although this is an improvement over nothing at all, the product team decided to adopt an approach that assigns a new, unique, secure and hard to guess identifier with each inbound request when one is called. These identifiers, hereafter, are referred to as CSRF tokens. These CSRF tokens

are unique and distinct to inbound HTTP request/user combination, and are stored in session and referred to later to verify a valid CSRF inbound request.

To mitigate memory bloat, a circular buffer of n CSRF token instances (where n is configurable) was implemented. Some approaches remove the CSRF token as the first request processed. This is by far the most secure way of handling this, but also prevents the user from using the browser back button. It also introduces a lot of complexity to AJAX requests.

The design approach we embraced is to retain the CSRF token in a circular buffer until it is overwritten. CSRF token instances generate on the fly when requested by the JSP subsystem, each HTTP request having its own unique CSRF token. If multiple forms or links need a CSRF token value, the same token value is used within that particular request.

### Targeted Processing

The most secure application would protect every inbound HTTP request with a CSRF token. We adopted a more targeted approach, protecting only those HTTP requests that result in either an insert, update or delete transaction to the database or that navigate through a work item. All `FrontServlet` (both GET and POST methods) and `WorkItemAction` (both GET and POST methods) URLs are guarded with CSRF protection. Product teams can alter supported CSRF configuration to allow an application project to protect other types of requests, assuming they also update their custom JSPs/JavaScript/AJAX logic to dispense out CSRF token values appropriately.

### Verification of CSRF Tokens

The ACSI security filter ensures that inbound requests for those URLs that match the requisite configuration contain a valid CSRF token. Action will be taken, regardless of whether the CSRF token is wrong OR missing.

CSRF token verification is forgone if there is no session allocated to the request, assuming that other authentication means are undertaken to ensure the request is being delivered from a legitimate source.

### Supporting Public APIs

`com.agencyport.security.csrf.ICSRCFGuard` is the main interface that exposes the primary mechanism for dispensing CSRF tokens and for verifying that inbound requests are appropriately provisioned with CSRF tokens.

Methods:

```
/** * Installs the CSRF guard onto the session. * @param session is the session for the user. * @throws com.agencyport.security.exception.SecurityException if the installation fails. */ void install(HttpSession session) throws com.agencyport.security.exception.SecurityException; /** * Returns whether or not the CSRF guard is active or not. Typically during performance testing CSRF guards are turned off. * @return <code>true</code> if CSRF protection is on. */ boolean isActive(); /** * Prepares a request for CSRF handling. This is called at the very beginning by the security filter allowing the guard to initialize itself. * @param request is the current HTTP request. * @throws com.agencyport.security.exception.SecurityException if the installation fails. */ void initialize(HttpServletRequest request) throws com.agencyport.security.exception.SecurityException; /** * Generates a new/returns the current CSRF token. * @return the current CSRF token. */ String getCurrentCSRFToken(); /** * Returns the CSRF tag name to use in the application. * @return the CSRF tag name to use in the application. */ String getCSRFTokenTagName(); /** * Verifies that the inbound request is properly provisioned with a valid CSRF token. * @param request is the current HTTP request. * @throws CSRFViolationException if the request is not properly provisioned with a valid CSRF token. */ void verifyRequest(HttpServletRequest request) throws CSRFViolationException;
```

`com.agencyport.security.csrf.CSRFHelper` is a helper utility class that simplifies getting access to the `ICSRCFGuard` instance for the current request/session.

Methods:

```
/** * Returns the guard currently installed on the HTTP session. This accesses the session by way of the LoggingContextStore giving access to the current HTTP request which gives access to the underlying session. * @return the guard currently installed on the HTTP session. */ public static ICSRCFGuard getCSRFGuard(); /** * Returns the guard currently installed on the HTTP session. * @param request is the current request from which the session is derived. * @return the guard currently installed on the HTTP session. */ public static ICSRCFGuard getCSRFGuard(HttpServletRequest request); /** * Returns the guard currently installed on the HTTP session. * @param session is the current session. * @return the guard currently installed on the HTTP session. */ public static ICSRCFGuard getCSRFGuard(HttpSession session);
```

`com.agencyport.jsp.JSPHelper` is a helper utility class that was enhanced to provide CSRF tokens from JSP constructs.

New Methods:

```
/** * Returns the current CSRF token value. * @return the current CSRF token value. * @since 5.0 */ public String getCSRFToken(); Start Example <script> ap.workItemAssistant.init('<%=jspHelper.getTransaction().getId().toString() %>', '<%=jspHelper.getPage().getId().toString() %>', '<%=jspHelper.getWorkItemId() %>', '<%=jspHelper.getClientUpdateInterval() %>', '<%=jspHelper.getClientUpdateMaxTimeout() %>', '<%=jspHelper.getCSRFToken()%>'); </script> End Example /** * Appends the CSRF token to the HREF. * @param href is the HREF to append to. * @return the HREF with the CSRF token appended. * @since 5.0 */ public String appendCSRFToken(String href); Start Example JSPHelper jspHelper = JSPHelper.get(securityProfile.getRequest()); url = jspHelper.appendCSRFToken(url); End Example tags/page/csrf.tag is
```

a new JSP tag in the 'ap' name space that simplifies the addition of a CSRF token hidden field to an HTML form. Start Example <form action ="FrontServlet" id="accountForm" name= "accountForm" method ="post" class="form-horizontal"> <input type ="hidden" name= "TRANSACTION\_NAME" id="TRANSACTION\_NAME" value ="<%=tran.getId()%> " /> <input type ="hidden" name="WORKITEMID" id= "WORKITEMID" value ="<%=details.getAccountId().toString()%> "> <ap:csrf /> End Example

tags/page/csrf.tag is a new JSP tag in the 'ap' name space that simplifies the addition of a CSRF token hidden field to an HTML form.

Start Example <form action ="FrontServlet" id="accountForm" name= "accountForm" method ="post" class="form-horizontal"> <input type ="hidden" name= "TRANSACTION\_NAME" id="TRANSACTION\_NAME" value ="<%=tran.getId()%> " /> <input type ="hidden" name="WORKITEMID" id= "WORKITEMID" value ="<%=details.getAccountId().toString()%> "> <ap:csrf /> End Example

## Supporting Configuration

The CSRF module is driven in part by configuration. Since this infrastructure is related to security, the best home for any specific application configuration is in acsi.properties file. The following are the various configuration options:

- `security.csrf_protection` is a Boolean true/false flag that allows an application to turn off CSRF protection at the application level. If this flag is absent, then its default value is true. When this flag is off, verification of CSRF tokens by AgencyPort security filter are not carried out. The only time to turn this flag off is during performance testing. Since performance testing is based on the execution of pre-recorded scripts, the CSRF tokens that were records are longer valid when the scripts run.
- `security.csrf_uri-include_list` specifies which HTTP requests to engage CSRF verification. The out of the box product currently supports CSRF protection against all `FrontServlet` and `WorkItemAction` HTTP requests (both GET and POST methods). There is no default and a proper configuration for this option looks like this:

```
#####
ACSI - CSRF protection configuration # CSRF supports a
list of URIs and methods (POST, GET) which # the framework will protect against potential CSRF attacks.# The configuration is
formatted as a list of URIs delimited by semi-colons with # the specific methods within parenthesis following each URI # each method
delimited with a comma. #####
security.csrf_uri_include_list=FrontServlet(POST,GET);\ WorkItemAction(POST,GET)
```

- `security.csrf_token_repo_max_size` is an integer based application property that allows an application to control the size of the circular buffer. The default value is 128. An application will rarely need to override this value and should consult product development before doing so.

## Testing Strategies

### Performance Testing

When running performance tests, the `security.csrf_protection` flag must be set to false; otherwise, all `FrontServlet` and `WorkItemAction` HTTP requests will fail. When recording performance test scripts, the setting of this flag is not important.

### Functional/Vulnerability Testing

There are many places in the framework where `FrontServlet` and `WorkItemAction` requests are launched. Any place where either of these end points are executed, whether it be the commit of a work item page, a URL link on a work item's left navigation menu or a work item action taken from the home page or work queue, CSRF tokens should be rendered out as form variables or JS data.

The following is an example from the work queue where the CSRF token is located:

```

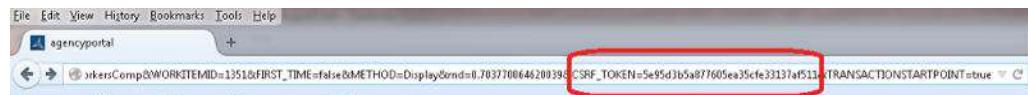
<input id="WORKITEMID" type="hidden" value="1351"></input>
<input id="action" type="hidden" value="Open"></input>
<input id="openmode" type="hidden" value="E"></input>
<!--
 tags\page\csrf.tag
-->
<input id="CSRF_TOKEN" type="hidden"
value="aa8509d8ff294f98949cb18cac76cad6" name="CSRF_TOKEN"></input>
<!--
 End tags\page\csrf.tag
-->
```

</ span>

The following is also an example of a CSRF token on a TDF page:

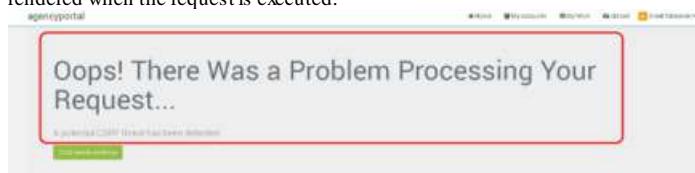
```
<p> <input type="hidden" name="NEXT" value="SaveToDefaults" /> <input type="hidden" name="WORKITEMID" value="1351" /> <input type="hidden" name="FROMLIGHTBOX" value="lb_save_to_default" /> <input type="hidden" name="METHOD" value="Process" /> <input type="hidden" name="PAGE_NAME" value="generalInfo" /> <input type="hidden" name="TRANSACTION_NAME" value="workersComp" /> <!-- tags\page\csrf.tag--> <input type="hidden" name="CSRF_TOKEN" id="CSRF_TOKEN" value="0e86482067f91af07561e0956637cb66" /> <!-- End tags\page\csrf.tag-->
```

You will also notice that the URL in the address bar has a CSRF token when a work item is opened. For example:



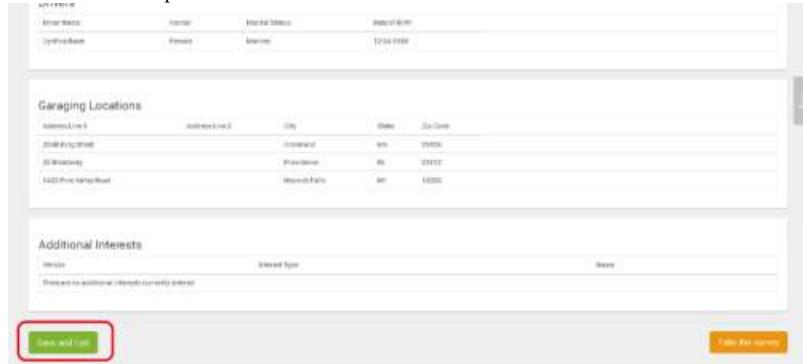
When testing, ensure the following:

1. If the CSRF token values on the address bars, form fields or other JS variables are altered or removed, the following pages should be rendered when the request is executed:



2. If the CSRF token values on address bars, form fields or other JS variables are altered or removed, the following action is taken and an audit trail record should be written to the audit log `audit_msc` table when the request is executed:  

audit_id	success_flag	action_beat_value	action_beat_type	action_beat_name	action_code	target_beat_value	target_beat_type	target_beat_name	tag_id	tag_label	target_id	description
316	005	0	systems	systems	0							A potential CSRF threat has been detected
3. Test to ensure that all aspects and all nooks and crannies of the four supported LOB templates work as expected. CSRF tokens were provisioned using the `<ap:csrf/>` JSP tag on the customsummary pages on each LOB. These CSRF tokens are validated when the Save and Exit button is pressed.



4. Any comprehensive testing effort should include a test using the ZED Attack Proxy tool that was introduced in v.4.6. There are specific configuration options available in this tool specifically for testing the CSRF protection aspect of security. Refer to the [OWASP Zed Attack Proxy \(ZAP\)](#) document for more information on using this tool.

## Using Components with Known Vulnerabilities

Components, such as libraries, frameworks and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts. It is important that the application identify all components and the versions being used. Using Maven, Ant or Jenkins for build dependencies and the SDK's `JarVersionManager` class for runtime dependency checking helps position an application in a better place regarding this concern.

## Unvalidated Redirects and Forwards

Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

## Verification of Redirect Target URLs

All URLs that are a target passed to `HttpServletResponse.sendRedirect()` are validated by default. This is in support of the OWASP Top 10 #10: Unvalidated Redirects and Forwards.

### Supporting Configuration

Verification of work item actions is controlled by application property configuration. Since this infrastructure is related to security, the best home for any specific application configuration is in the `acs.i.properties` file. The following are the various configuration options:

```
ACSI - URL Verification Configuration
the framework will verify URLs redirected to.
#####
Setting the following flag to false will deactivate url verification.
security.verify_urls=true
The following list provides a white list of valid starting sequences for URLs
security.valid_url_start_sequences=${my_portal_app};\${APPLICATION_NAME};\${APPLICATION_NAME};\DisplayHomePage
```

## Content Security Policy

The Content Security Policy (CSP) allows you to restrict and limit what the browser can do. For example, you can instruct the browser to only load certain types of resources and from where, such as only allowing the browser to load Java when it comes from a certain domain or generate forms to a certain domain. CSP helps reduce the risk of [cross-site scripting \(XSS\)](#) attacks that inject malicious code into your application. Refer to the [Content Security Policy](#) page on the OWASP website for more detailed information.

To implement this policy in your application, add the following configuration to the HTTP response headers section of the `acs.i.properties` file:

```
security.content_security_policy=default-src 'none';object-src 'self';style-src 'self' 'unsafe-inline';img-src 'self';
security.content_security_policy+=form-action 'self';media-src 'self';font-src 'self';connect-src 'self';
security.content_security_policy+=plugin-types application/pdf application/x-shockwave-flash;reflected-xss block;
security.content_security_policy+=script-src 'self' 'unsafe-inline' 'unsafe-eval';frame-src 'self'
```

# Encryption

`EncryptionHelper` is a class that allows an application to use an encryption algorithm, such as AES (Advanced Encryption Standard), DESede (Triple DES Encryption) or Blowfish, to encrypt/decrypt data. First, the application must decide whether the built-in encryption agent that comes with the framework is adequate or whether a custom encryption agent needs to be implemented to fully support the security requirements. To help guide developers in that decision, they must ask the following questions:

- Is an encryption algorithm other than AES, DESede or Blowfish required?
- Are the default modes and padding parameters that accompany the JRE associated with the three supported encryption algorithms not sufficient?
- Are the key sizes of 128 and 192 bits for the AES and DESede encryption algorithms not sufficient?
- Does the application have custom "salting" requirements?
- Does the application need to access secret keys from key store files in a way that is different from how the built-in encryption agent does?
- Does the application require a string encoding algorithm other than Base 64 or Hex.

If the answer to any of these questions is yes, then the developer must implement their own custom implementation of the encryption agent interface (`com.agencyport.security.encryption.IEncryptionAgent`). After that implementation is available, you can register your own `com.agencyport.security.encryption.IEncryptionAgent` implementation by configuring the class name of that implementation under the `com.agencyport.security.encryption.EncryptionHelper.encryption_agent_classname` application property.

If you answered no to any of the above questions, you can use the encryption agent implementation that comes with the framework. In that case, there is no need to configure the aforementioned property. The default helper's encryption agent accommodates the three listed encryption algorithms. The default algorithm is Blowfish. If you want to use either AES or DESede, you must configure the `com.agencyport.security.EncryptionHelper.default_encryption_type` application property. The three acceptable values are AES, DESede and Blowfish (default).

Applications can register their own cipher provider transformations instead of using the one provided by default. For example, if an application uses AES or Blowfish and they don't want to use the default cipher modes and padding, they can configure their own cipher transformations using the following pattern:

```
security.<encryption_type>.cipher_transformation=AES/ECB/PKCS5Padding
```

(where ECB = the cipher mode and PKCS5Padding = the padding algorithm)

There are two ways to load the encryption secret key values into the encryption agent instance:

- as an application property, or
- within a keystore file

We prefer that you convey a secret key to the encryption agent using a keystore file since this file is password protected, which adds an extra layer of protection to the encryption/decryption secret key. This can be done using the `com.agencyport.security.encryption.EncryptionHelper.<encryption-type>.keystore_filename` (where `<encryption-type>` is the encryption algorithm used) application property. Secret keystore file formats have cipher affinity, which means that keystores must be created using the cipher that will eventually use them. Refer to the [Keystore Files](#) section below for more information.

The secret key can also be conveyed via the `com.agencyport.security.encryption.EncryptionHelper.secret_key` application property. However, this technique is not as secure as conveying the secret key by way of a keystore file.

- To gain access to the default encryption helper, use the `get()` method. The framework gains access to its encryption helper using this method.
- To gain access to an encryption helper other than the one based on the default encryption agent, use the `createEncryptionHelper(com.agencyport.security.encryption.IEncryptionAgent.EncryptionType)` method.
- To create secret key keystore resources, use the `BuiltinEncryptionAgent.main(String[])` utility command line program. However, you must be using the default `BuiltinEncryptionAgent` encryption agent implementation as the encryption agent in your application.

## Keystore Files

Keystore files are password protected files that add an extra layer of protection to encrypted data. The passwords to these files should be stored in an external resource. The default password file resource is bound in `apbase.jar` with the name of the resources/keystore/asci-info.bin. The password in the file is intentionally left obscured so that it is kept secured. A check sum is also used so that tampering or alterations of the file can be detected.

Project teams can create their own password files using the command utility included with apbase.jar.

### Example

In the following command example, a password file is created called mypassword-file with a password of foobar:

```
[devuseradmin]$ java -cp apbase.jar com.agencyport.security.encryption.SecurePasswordFile -mode
create -pw foobar -fn mypassword-file.bin
```

```
Saved password to resource id: /home/devuseradmin/test/mypassword-file.bin, key: mypassword-
file.bin, version: null, is stale: true, load method: FROM_FILE_SYSTEM, loaded as resource:
false, file: /home/devuseradmin/test/mypassword-file.bin
```

# Field Security

AgencyPortal's secure fields feature provides project teams with the ability to secure fields that contain sensitive data fields. This feature is categorized into two major areas: the secured data entry user experience and the data persistence layer's encryption of field data at rest.

- The secure data entry user experience can be enabled for a field via TDF XML configuration.
- The encryption at rest functionality, which is available within AXE, allows encrypted field paths to be registered to APDataCollection with application properties and with transaction configuration. Additionally, project teams can specify (via an application property or dynamically via Java override) whether encrypted fields should be transferred to a new work item when copying the xmlstore record.

## Architecture

### Secured Data Entry Architecture

#### Server Side API for Accessing TDF Field Security Metadata

The `FieldSecurityElement` Java class, included in the `com.agencyport.html.elements` package, is responsible for representing the object model for the TDF configuration options for the secure fields functionality. A `FieldSecurityElement` is instantiated for `BaseElement` objects, which have `fieldSecurity` metadata. `BaseElement` includes a `getFieldSecurityElement()` method, which can be used to obtain access to the field security information for a given TDF field.

#### Rendering Field Security Info to the Client

The HTML rendered for a secured text or date field is no different than any other text or date field except that the input element's value attribute can contain the masked representation of the string in scenarios where there is a raw value for the field, but the user is not authorized to see the raw value. `BaseElement` uses the `FieldSecurityElement` object's `getMaskedValueForDisplay()` method to derive the masked form of the underlying value.

The SDK's `<ap:form>` tag (as described by the "ap\_form.tag" file) is responsible for dynamically generating the client JavaScript necessary to instantiate an `ap.fieldSecurityHandler` object for the appropriate `ap.Field` objects. The information conveyed to the field's `initFieldSecurity` function is composed of the following parameters:

<code>secureDataEntryMode</code>	This is a string literal that corresponds to the <code>secureDataEntryMode</code> specified by a call to the <code>FieldSecurityElement</code> object's <code>getSecureFieldMode()</code> method. This is ultimately driven by static TDF metadata.
<code>numRevealCharacters</code>	An integer value as specified by a call to the <code>FieldSecurityElement</code> object's <code>getNumRevealCharacters()</code> method. This is ultimately driven by static TDF metadata.
<code>initialValue</code>	Unlike the previously described parameters, the <code>initialValue</code> [as returned by the <code>FieldSecurityElement</code> object's <code>getInitialValueForClient()</code> method] is the only parameter that is not derived from a static TDF configuration. This parameter is the means by which the server informs the client what the true raw value for the secured field is assuming the user has the special "canReadSecureData" permission. If there is no value, an empty string will be returned and if there is a value but the user does not have the "canReadSecureData" permission, then a special token, "\${NOT_RENDERED}", will be returned

#### Client Side Implementation

Other than making broad decisions about whether to secure data entry on a field and whether the client is allowed to see an existing value for a secured field, the server side doesn't have influence on the end-user experience. The majority of this logic is supported by the `ap.FieldSecurityHandler` JavaScript class. The `ap:form` JSP tag is responsible for dynamically generating the client JavaScript necessary to instantiate an `ap.fieldSecurityHandler` object for the appropriate `ap.Field` objects. The `ap.fieldSecurityHandler` object is then responsible for altering the data entry user experience for the field, based on the following scenarios:

As the user has entered data into a field, the value should be protected (or "masked" from view) while the user is still on the page.

The most common example of a mask-as-you-type experience on the web involves using a password input. Unfortunately, modern browsers will assume it's a literal password field on a sign in page and may cause some quirks. Browsers will offer to "save the password", which, for security reasons, can't be disabled since most browsers ignore the `autocomplete="off"` flag for sign-in pages.

User experience data indicates that mask-as-you-type can be particularly off-putting as users cannot validate that they are typing the intended characters.

As such, `ap.fieldSecurityHandler` masks the data when the user tabs off the field (onblur).

If the data entered is not valid (e.g., not enough characters in the SSN), the validation error should behave like a normal non-sensitive field. The validation is shown as normal and the data is not masked. This is because, if the data is not valid, there is very little benefit achieved by masking it, and we don't want to fight the user by hiding the invalid data.

After the user has tabbed off of the field and the data is valid, the application will now mask the field data to prevent shoulder snooping. When masking the data, any format masked characters are honored if a format mask is present (i.e., a masked SSN should look like '\*\*\*-\*\*-\*\*\*\*', as opposed to '\*\*\*\*\*'). Some number of suffix or prefix characters will be revealed per the `secureDataEntryMode` and `numRevealCharacters` configured for the field (i.e., a partially masked SSN with two suffix characters excluded from the mask would look like this '\*\*\*-\*\*-\*\*23').

When revisiting the page, if there is an existing value for the field, the user is shown the masked label with an edit icon. The `initialValue` provided to the `ap.fieldSecurityHandler` in this case will be "{\$NOT\_RENDERED}". When the user clicks edit, they will have to enter the value from scratch. This is consistent with the way credit card fields work across the web.

Fig. 1 Secured field with focus:

Tax ID Type    SSN

Tax ID    111-11-1111

Fig 2. Secured field without focus:

Tax ID Type    SSN

Tax ID    \*\*\*-\*\*-\*\*\*\*

Fig 3. Secured field with persisted value when revisiting the page:

Tax ID Type    SSN

Tax ID    \*\*\*-\*\*-\*\*\*\*

Fig 4. Secured field with persisted value after clicking edit:

Tax ID Type    SSN

Tax ID

A user with the "canReadSecureData" permission should be able to see the sensitive data on subsequent page visits. The user experience described by Fig. 1 and Fig. 2 above will be engaged for those users.

When the field is rendered in read only mode, it should render as a label with the masked text. There will be no edit action present, and therefore no way for a normal user to see the raw data. However, a user with the "canReadSecureData" permissions should be provided with a toggle option to see the raw text.

Fig 5. A read only secured field when the user does not have the "canReadSecureData" permission

SSN    \*\*\*-\*\*-\*\*\*\*

Fig 6. A read only secured field when the user has the "canReadSecureData" permission

SSN    \*\*\*-\*\*-\*\*\*\* [Show](#)

Since the text input that backs the field will be masked by the time the field is submitted and will therefore not contain the true value, as entered by the user, there is another mechanism in place that is responsible for transmitting the true value to the server. At the time of masking the value, the `ap.fieldSecurityHandler` object transfers the true value to a hidden input, which it is managing for the field. This hidden input will be created with a name attribute of "`_CONTROL_RV.${field.name}"`. When the form data is posted to the server, it is the server's responsibility to ignore text input's value parameter and retrieve the value from the hidden input instead.

### Secured Data Entry Persistence

During typical request processing for a data entry or roster page, an `HTMLDataContainer` object is inflated with the form data's request parameter values keyed by the TDF `fieldElement` ID attributes. Later, during the `APCommand.executeDataStaging()` phase, those values are transferred into the `APDataCollection` document. All other non-TDF form data that came in on the request is also added to the `HTMLDataContainer` with the original request parameter name from the form acting as the `HTMLDataContainer` key.

As a result of the client side changes for secure data entry, the true value for the secured field that originated from a hidden input with the prefix "`_CONTROL_RV`", initially comes into the `HTMLDataContainer` as part of the non-TDF data and the corresponding data under the TDF `fieldElement` ID key is the, somewhat useless, masked string which came from the field's text input. As such, the `CollectionProcessor` contains logic to correct the `HTMLDataContainer` in these scenarios by replacing the masked value with its true form.

Additionally, if the special token "`NOT_RENDERED`" comes in from the client under the hidden input, then `CollectionProcessor` has logic to load up the true value for that field from the persisted data collection and drop it into the `HTMLDataContainer` under the appropriate TDF `fieldElement` ID key.

### AXE Data Encryption Improvements

#### Registration of Encryption Fields

The transaction, and ultimately TDF XML configuration, can register encrypted fields after the static initialization of the `APDataCollection`. A static function named "`registerEncryptionField`" is included in `APDataCollection` to support adding to `APDataCollection`'s set of encryption fields post static initialization.

The intention of this function is that it is only used internally by the SDK during the transaction initialization process and invoked as part of bootstrap, although other uses are not prohibited as long as care is taken to ensure that the method is invoked consistently across all nodes in a cluster.

When a transaction object is being hydrated, as part of the `TransactionDefinitionManager`'s init process, the transaction based means of registering encryption fields are interpreted (i.e., the `transaction_id`.`encryption_fields` and `lob_code`.`encryption_fields` properties along with the ID attributes of all `fieldElement` elements that have a child `fieldSecurity` element where the `encryptAtRest` attribute is true are registered to `APDataCollection` as encryption fields).

#### Exclusion of Encrypted Fields during Work Item Copy

Work item copy functionality provides project teams with the ability to specify, either statically via an application property or dynamically via Java override, whether encrypted fields should be transferred to the new work item when copying the xmlstore record.

A transaction based LOB property named `copy_encrypted_data` allows transactions to specify whether encrypted data should be included during the work item copy process. A `shouldCopyEncryptedData` method is included in `CommandCopyImpl` and is responsible for interpreting this property. This method can be override if there is a need to dynamically determine whether encrypted data should be retained for a copy procedure.

### Server Side Configuration

#### TDF Schema

The standard [TDF XML schema](#) definition includes the `fieldElement`, which supports the `fieldSecurity` child element. The `fieldSecurity` element supports the following attributes, which are used to convey the level of sensitivity for the field:

Attribute	Description	Default Value
<code>encryptAtRest</code>	Indicates whether the <code>fieldId</code> for this <code>fieldElement</code> should be added to the <code>APDataCollection</code> 's list of fields to encrypt. The means of registering a field to encrypt exists in addition to using application properties.	False

Attribute	Description	Default Value
secureDataEntryMode	Defines what type of client-side secure data entry should be enabled for this field. For 'disabled' mode, the field behaves like a normal field. For 'revealPrefix' and 'revealSuffix', a part of the value is secured while a suffix/prefix whose length is determined by the numRevealCharacters attribute is shown to the user. For 'fullyMask', the whole string is secured from view. This attribute is only supported if the parent fieldElement's type attribute is "text" or "date".	Disabled
numRevealCharacters	The number of characters to reveal to the user if the secureDataEntryMode is 'revealPrefix' or 'revealSuffix'. This attribute is required if the secureDataEntryMode is 'revealPrefix' or 'revealSuffix'.	N/A

### Example

Consider a scenario where there is a need to treat a Driving License Number as sensitive data. In this scenario, the field should be encrypted at rest, the last two characters of the license number can be shown to the user, and drivers are captured on a roster page where roster copy functionality is enabled; however, there is a need to prevent the driving license number from being copied. The following is an example of what the fieldSecurity configuration would look like for this scenario:

```
<fieldElement type="text" id="CommAutoLineBusiness.CommlDriver.DriverInfo.License.LicensePermitNumber" label="Driving License Number" size="15" maxLength="15" required="true" groupId="CommAutoLineBusiness.CommlDriver.DriverInfo.License" copy="false">
<fieldSecurity encryptAtRest="true" secureDataEntryMode="revealSuffix" numRevealCharacters="2" /></fieldElement>
```

### Application Properties

The following application properties are included in the application framework to secure fields:

Property Name	Description	Default Value (if any)
\${transaction_id}.copy_encrypted_data \${lob_code}.copy_encrypted_data application.copy_encrypted_data copy_encrypted_data	This is a transaction/LOB based application property, which indicates whether encrypted fields should be copied during work item copy operations.	False
\${transaction_id}.encryption_fields \${lob_code}.encryption_fields	Not a true transaction/LOB based application property, but can be specified at the transaction or LOB level to supplement the list of AXE fields to encrypt, garnered from the separate "application.encryption_fields" property and the TDF's fieldSecurity configuration.	N/A
application.secure.field.firstTime	This is a property file that can be added to framework.properties and indicates whether data entry fields can be masked on the first view of a page.	True

# Session Timeout

The session timeout feature in AgencyPortal times out a user's session after a period of inactivity (no mouse or keyboard movements). This period of inactivity is configurable by using the `client_update_timeout` application property. This property controls the amount of time a user can become idle before they are logged out of the system. This property defaults to 600 seconds (10 minutes).

When the session approaches the time out period (e.g., nine minutes idle), an alert dialog displays to notify the user that their session is about to expire and they have 60 seconds (by default) to remain in the session. The amount of time that displays in this message is configurable by using the `idle_timeout_countdown_start` application property. This property defaults to 60 seconds (1 minute).

If the user wants to continue working in the application, they can simply click Continue; otherwise, they will be automatically logged out of the system and the sign in page will display.

Refer to the [Getting Started](#) topic in the User Workflow section for more information on how this feature works.

# Request Processing

The purpose of the request processing component is to:

- provide a mechanism to support a standard sequence of controlling tasks required to display and process a series of different "types" of HTML pages that, together, make up the web application "transaction."
- provide a mechanism to support customizations to the standard sequence of controlling tasks required to display and process a series of HTML pages.

## Transaction Page Processing

The typical web transaction presents a number of pages that generally fall into only three simple categories of page layouts:

<b>Basic Data Entry (for non repeating data)</b>	Contains standard HTML field elements.
<b>Roster (a list for repeating data)</b>	A roster form is used to capture repeating data (e.g., risk locations or loss history). The top part of the form contains the roster of previously entered items. The data displayed in the roster section is generally some "key" fields from the item; below the roster is a section for adding a new item.
<b>Summary (display only)</b>	A summary page is something like a "quote results" page or a page displaying the submitted application in a display only format.

## Displaying and Processing HTML Form Layouts

Based on the type of HTML form to process, a series of processing tasks must take place in a well-defined order. To ensure consistency of, and accuracy in, processing these tasks, a collection of processing commands are pre-defined.

These commands contain all the logic to completely process the display and process side of a series of web pages that make up a transaction. Many of the individual processing tasks need to happen during both the display and processing logic of page processing. Also, many of the helper objects required on the display side of page processing are required on the process side of page processing logic.

This combination of common objects and common processing tasks lead to the creation of a base APCommand object. A catalog of processing commands that inherit from APCommand are available to handle the processing of the various layouts mentioned above.

## Supporting a Standard Sequence of Controlling Tasks

APCommand is the base command class that contains all of the common methods required by both the display side and process side logic. APCommand also contains references to all of the common objects required for processing. The APCommand class is an abstract class containing the process() abstract method.

```
abstract public void process() throws ServletException, IOException;
```

The framework comes with a number of classes that extend APCommand and automatically process the standard page layouts. These classes implement the process() method. These are referred to as the base processing commands (or base processing classes).

## Framework Base Processing Commands

The following is the list of classes that extend APCommand:

<b>CMDBaseDisplayDataEntry</b>	Display a basic data entry page.
<b>CMDBaseProcessDataEntry</b>	Process a basic data entry page.
<b>CMDBaseDisplayFTDE</b>	For the very first page of the transaction, which has special needs (FTDE is First Time Data Entry).

<b>CMDBaseProcessFTDE</b>	Process the very first page of transactions.
<b>CMDBaseDisplayRoster</b>	Display a roster page.
<b>CMDBaseProcessRoster</b>	Process the added item on a rosterpage.
<b>CMDBaseDisplayRosterEdit</b>	Display the editing page for a roster line item.
<b>CMDBaseProcessRosterEdit</b>	Process the edit action for a roster line item.
<b>CMDBaseProcessRosterDelete</b>	Process the delete action for a roster line item.
<b>CMDBaseProcessRosterRecover</b>	Process the recover action for a roster line item (policy change transaction only).
<b>CMDBaseDisplayPage</b>	Display a display only/summary type HTML page.
<b>CMDBaseProcessPage</b>	Process a display only/summary type HTML page.
<b>CMDBaseDisplayPDF</b>	Display a PDF document (e.g., an ACORD 130 form).
<b>CMDBaseDisplayFile</b>	Display an uploaded file (e.g., a loss run worksheet).
<b>CMDBaseLaunchPASRequest</b>	Launch point for policy change based requests.
<b>CMDBaseProcessApplyDefaults</b>	Process the Apply Defaults action.
<b>CMDBaseProcessConvertToApplication</b>	Process the Convert to application action.
<b>CMDBaseProcessDeleteDefaults</b>	Process the Delete defaults item action.
<b>CMDBaseProcessRenameDefaults</b>	Process the Rename defaults item action
<b>CMDBaseProcessSaveToDefaults</b>	Process the Save to defaults action.

If the page has special processing need(s) that is not satisfied by the base command class, the developer subclasses the appropriate base processing command (not the APCommand) and adds the code to process the special need.

### Example

```
public class CMDProcessEquipment extends CMDBaseProcessDataEntry {
```

A mechanism termed the front controller pattern dynamically invokes the base processing class (or the developer written subclass).

### Front Controller Pattern

AgencyPortal software implements a front controller pattern. The front controller pattern centralizes request logic by presenting a single point of entry for the web application. A front controller typically consists of a request handler and a catalog of dynamically invoked commands that process the web request.

A series of well-known request parameters determines which cataloged command to invoke.

The following URL example is used to help illustrate the concept of the front controller pattern:

```
http://FrontServlet?TRANSACTION_NAME=workerscomp&PAGE_NAME=generalInfo&METHOD=Display&FIRST_TIME=true"
```

The front controller pattern specifies a single point for a web server to interface with a web application. In the example URL, that single entry point is that the `FrontServlet` extends `javax.servlet.http.HttpServlet`.

The `FrontServlet` uses a helper class, `servlets.base.RequestHelper`, to determine the command class to handle the web request. The `RequestHelper` uses values from the request parameters to build a fully qualified Java class name.

The class name is mapped to one of the base command class names or to a subclass if the page has special processing needs. The FrontServlet instantiates the resolved Java class name and calls the command's process () method.

The following is a snippet of a TDF:

```
<page id="generalInfo" title="General Information" type="dataEntry"><pageElement type="fieldset" legend="Mailing Address">
<fieldElement type="text" id="Insured.Party.Addr.Addr1" label="Address" required="true" help="" <fieldElement type="text"
id="Insured.Party.Addr.Addr2" label="Address Line 2" required="false" <fieldElement type="text" id="Insured.Party.Addr.City" label="City"
required="true" help="" size="20" <fieldElement type="selectlist" id="Insured.Party.Addr.StateProvCd" label="State" required="true"
<optionList reader="xmreader" source="codeListRef.xml" target="selectOne"/><optionList reader="xmlreader" source="codeListRef.xml"
target="state"/></fieldElement> <fieldElement type="text" id="Insured.Party.Addr.PostalCode" label="Zip Code" required="true"
</pageElement> <pageElement type="fieldset" legend="Coverage Information"> <fieldElement type="selectlist"
id="WCLineBusiness.Cov.Limit.Amt" label="Employers Liability Limit" <optionList reader="xmlreader" source="codeListRef.xml"
target="employersLiabilityLimit"/> </fieldElement> </pageElement> </page>
```

To display and process this page, the developer needs to:

- determine the type of page to display or process to determine which base command classes to use.
- determine whether the base display and process command classes are sufficient to display and process the page. If not, the developer subclasses the appropriate base commands.
- set up the correct configuration in the properties file.

If special processing is not required to display and process the page, the default base processing classes are used at runtime to process the page. The developer does not have to write any Java code. The only task to perform is configuration.

Specify the prefix of the command class name:

```
workerscomp.root_command_class_name=com.agencyport.servlets.wc.CMD
```

## Determining the Class Names

Using information in the transaction file, information contained in the HTTP request and information in the properties file, the framework (class RequestHelper) determines the class name to instantiate (serve the request). This class name is referred to as the processing class name. The specific information is as follows:

Take the value of the request parameter TRANSACTION\_NAME.

### Example

```
TRANSACTION_NAME=wc
```

And append the constant root\_command\_class\_name to find a properties file entry.

### Example

```
wc.root_command_class_name=com.agencyport.servlets.wc.CMD
```

The value from that properties file entry is: com.agencyport.servlets.examples.CMD.

The value is pre-pended to the value of the request parameter METHOD. The valid values for this parameter are Display or Process.

So far, the processing class name is:

```
com.agencyport.servlets.examples.CMD.Display
```

Next, the value of the request parameter PAGE\_NAME is appended to the class name. The value of the PAGE\_NAME request parameter is derived from the attribute ID, from the page element, in the transaction file.

```
<page id="generalInfo" title="General Information" type="dataEntry">
```

The processing class name then becomes:

```
com.agencyport.servlets.examples.CMD.DisplaygeneralInfo
```

Lastly, two optional request parameters are interrogated for values. Those parameters are FIRST\_TIME and EDIT. If these attributes are present and set to true, the values of FirstTime and Edit are appended to the class name.

## Example

com.agencyport.servlets.examples.CMD.DisplaygeneralInfoFirstTime

or

com.agencyport.servlets.examples.CMD.DisplaygeneralInfoEdit

The last step performed by `RequestHelper` is to determine whether a base class is going to process the request or if there is a subclass to handle the request since the page requires special processing.

This determination is accomplished by a lookup into the properties file for an entry matching the processing class name.

If an entry is found, the processing class name is mapped to the name of the base class that can handle the page. If no entry is found, the request cannot be handled by a base class and the processing class name is the name of a developer written subclass.

## Note

Even though this process sounds counter-intuitive at first, it must be done in this way.

## Example

An example properties file entry is:

com.agencyport.servlets.wc.CMDDisplaygeneralInfo=com.agencyport.servlets.base.CMDBaseDisplayDataEntry

This entry says, "the page represented by the name `CMDDisplaygeneralInfo` can be processed by the base command `CMDBaseDisplayDataEntry`."

## Special Action Processing

When the request is not related to a transaction page, then it is treated as a special action (e.g., AutoSave, Convert Quick Quote to Full Application, etc).

The SDK uses the following special actions:

SaveToDefaults	"com.agencyport.servlets.base.CMDBaseProcessSaveToDefaults"
ApplyDefaults	"com.agencyport.servlets.base.CMDBaseProcessApplyDefaults"
ConvertToApplication	"com.agencyport.servlets.base.CMDBaseProcessConvertToApplication"
DeleteDefaults	"com.agencyport.servlets.base.CMDBaseProcessDeleteDefaults"
RenameDefaults	"com.agencyport.servlets.base.CMDBaseProcessRenameDefaults"
AutoSave	"com.agencyport.servlets.base.CMDBaseProcessAutoSave"
AutoSaveExit	"com.agencyport.servlets.base.CMDBaseProcessAutoSave"
AutoSaveSave	"com.agencyport.servlets.base.CMDBaseProcessAutoSave"
WorkItemAssistant	"com.agencyport.servlets.base.CMDBaseProcessWorkItemAssistant"
WorkItemLock	"com.agencyport.servlets.base.CMDBaseProcessWorkItemLock"
Timeline	"com.agencyport.servlets.base.CMDBaseProcessTimeline"

If you have override or add new actions, you can do so using the `SPECIAL_RECOGNIZED_ACTIONS` application property.

## Example

```
SPECIAL_RECOGNIZED_ACTIONS=vinLookup,com.carrier.servlet.CMDBaseProcessVinLookup;addressLookup,com.carrier.servlet.CMDBaseProcessAddressLookup;
```

## Customizing the Standard Sequence of Processing Tasks

Many times, there is a need to customize the default behavior of a command task. During execution of the default set of controlling tasks, it may be necessary to "get a hook into" the controlling task to, for example, manipulate data before the standard task runs, short circuit the standard controlling task, etc.

Supporting this type of customization is accounted for by having the standard tasks "call out" to stub methods. The default implementation of stub methods is to simply return (doing nothing). To add customizations, a developer overrides the base command class and implements an overridden version of the stub method, providing the specific customizations required.

## Example

Suppose that whenever data from a certain page is updated, some special data handling logic is required.

The following is the section in the `CMDBaseProcessDataEntry` class where the data update logic is initiated:

```
try { executeUpdateDataAccess(dataAccessType); } catch (APEException apex) { logger.severe("executeDataAccess call caused exception ==> " + apex.getMessage()); displayApplicationErrorPage(apex, IServletsSecurityConstants.LOGON_URL); return; }
```

The `executeUpdateAccess()` method is found in the `APCommand` superclass. This method controls the update of the data store with the data that came in from the page.

In `APCommand`, a line that looks like the following is located in the `executeUpdateDataAccess()` method:

```
try { executeCustomUpdateDataAccess(dataAccessType, conn); ... } catch...
```

This method is the "hook" to add special logic.

The `executeCustomUpdateAccess()` method in `APCommand` does nothing, and looks like the following:

```
protected void executeCustomUpdateDataAccess(int dataAccessType, Connection conn) throws APEException { }
```

This illustrates the mechanism for adding special logic: `APCommand` is "calling itself," invoking a method that does nothing.

When `APCommand` is subclassed when `CMDBaseProcessDataEntry` (for example) is subclassed, the method is invoked at runtime instead of the empty superclass method if the `executeCustomUpdateDataAccess()` method, which contains the required special logic, is put in the subclass.

The following is a list of methods, which can be overridden:

<code>void initCustomControlData()</code>	This method is available to programmatically update the control data object.
<code>void determineFinalMenuDisposition()</code>	This method is available to programmatically suppress the display of the menu on the HTML page.
<code>String determineNextPage()</code>	Determine the next page to display based on the value of the NEXT button.
<code>void executeCustomDataStaging()</code>	<p>This method is available to perform any manipulation of the data before regular processing begins. It is invoked from <code>executeDataStaging()</code> before the <code>APDataCollection</code> is updated by framework. Typical types of logic:</p> <ul style="list-style-type: none"><li>• Index manager initialization for add roster scenarios for repeaters within repeaters</li><li>• Hold on to current aggregate references for roster edit/delete scenarios</li></ul>

<code>void executeCustomPostDataStaging()</code>	Invoked from <code>executeDataStaging()</code> after <code>APDataCollection</code> is updated by the framework, but before any connectors are executed.  Includes custom updates of <code>APDataCollection</code> from special fields, etc.
<code>void executeCustomInitHTMLDataContainer()</code>	This method is available to post process the data in the <code>HTMLDataContainer</code> (e.g., you may need to massage date information before database insertion).
<code>void executeCustomPrepareDataBundle()</code>	During the execution of the <code>ConnectorProcess.run()</code> method, the code is in a loop, calling each registered connector's <code>execute</code> method. Right before that executive call, this method is called. It is an opportunity to add any data required for the connector.
<code>void executeCustomReadDataAccess()</code>	This method is available to add custom reads from the database. It is invoked from <code>executeReadDataAccess()</code> after the default page initialization has taken place.
<code>void executeCustomUpdateDataAccess()</code>	This method is available to add custom updates to the database. Invoked from <code>executeUpdateDataAccess()</code> before the <code>APDataCollection</code> is persisted and all connectors have been executed. Updates to axillary databases or other persistent formats.
<code>void executeCustomUpdateMenuController()</code>	This method is available to add custom updates to the menu tracking data and invoked from <code>updateMenuController()</code> before the <code>APCommand</code> base class updates the current page's menu status.

Naming a subclass name must follow the naming conventions for processing a class name.

The package name for the class is:

```
com.agencyport.servlets.wc
```

And the algorithm for the class name is:

```
Display or Process key word,
```

Plus

```
page name as found in the transaction definition file,
```

Plus (optionally)

```
the keyword Edit, FirstTime, or Delete
```

Since the subclass itself must be run, there is not any entry for it in the properties file; recall entries in the properties file tell which base class to run for a given processing class name.

# Database

The AgencyPortal SDK contains some database connectivity components that sit on top of the standard JDBC interface. These live under the com.agencyport.database package. The two main focal points of the design are the DatabaseAgent mechanism, which normalizes interaction with the JDBC API for different RDBMS vendors, and the DatabaseResourceAgent, which is the centralized API for retrieving and managing JDBC objects.

## DatabaseAgent

DatabaseAgent is an abstract class that exposes an RDBMS vendor agnostic interface for the application's communication with the database. There are several vendor specific implementations present within the SDK: the most notable are listed below:

Database Vendor	DatabaseAgent Implementation
Microsoft SQL Server	com.agencyport.database.SQLDatabaseAgent
Oracle	com.agencyport.database.OracleDatabaseAgent
MySQL	com.agencyport.database.MySQLDatabaseAgent
IBM DB2	com.agencyport.database.DB2DatabaseAgent

DatabaseAgent achieves most of its vendor agnosticism by replacing known substitution tokens found within SQL string templates, which are sent to its DatabaseAgent databaseNormalizeSQLStatement(String) function. The three known substitution tokens are:

Token	Description
<code> \${db_table_prefix}</code>	This token is replaced with the desired table name prefix. The actual prefix replacement string is resolved from the <code>db_table_prefix</code> Application Property which is typically specified within the <code>framework.properties</code> file. You may need to add table prefixing in some installations when the user ID used to connect to the database is not owner of the tables or stored procedures. The SDK specifies the <code>db_table_prefix</code> token in front of every reference to a table name in a SQL template. If you use this facility, it is important to add the dot (.) at the end of the prefix value (e.g., <code>db_table_prefix=agencyportal.</code> ).
<code> \${db_current_date_function}</code>	The specific DatabaseAgent for each vendor determines what the appropriate syntax for getting the current date is. For example, the SQL Server DatabaseAgent (named SQLDatabaseAgent) replaces this token with "getdate()". This is typically used for insert statements which need to set a value for a "date updated" column.
<code> \${db_current_time_function}</code>	Similar to the <code> \${db_current_date_function}</code> described above, but used specifically when a full timestamp is desired. For some vendors, these two functions are equivalent (e.g., The SQL Server DatabaseAgent replaces both tokens with the same 'getDate()' string).

## Example

The AccountSolrBatchIndexer class uses the DatabaseAgent to normalize a simple select account statement, which contains the  `${db_table_prefix}` token like so:

```
private static final String SELECT_ACCOUNT = DatabaseAgent.databaseNormalizeSQLStatement("select * from ${db_table_prefix}account where status <> ?");
```

Assuming the following prefix value is provided in application properties:

```
db_table_prefix=agencyportal.
```

The following expression is returned for the following given input string:

- Input:

```

 select * from ${db_table_prefix}account where status <> ?
• Output:
 select * from agencyportal.account where status <> ?

```

In addition to the known token substitution functionality, another way that the DatabaseAgent helps the framework achieve database agnosticism is by dynamically providing vendor specific large objects (LOB) support for storing XML documents. In several instances, XML documents, such as the ACORD XML document for a given work item, are stored within the database. The default data type used to store these XML documents varies by database vendor. The DatabaseAgent interface exposes a getLOB() API, which is used whenever an XML document needs to be retrieved from a JDBC ResultSet. Each specific DatabaseAgent implementation chooses the appropriate data type and handles any ResultSet functionality variances for the corresponding vendor.

## Note

Alternative DatabaseAgent implementations exist for Oracle and DB2, which support using the XML data type for the storage of XML documents instead of the BLOB/CLOB data types. These are called com.agencyport.database.OracleXMLDatabaseAgent and com.agencyport.database.DB2XMLDatabaseAgent respectively. The SQL files distributed with the SDK for these vendors do not leverage the XML data type so you will need to adjust these manually.

Since custom Java code does not usually need to be database vendor agnostic, for the most part, implementation teams will not have to interface directly with the DatabaseAgent outside of utilizing the token substitution functionality described above. However, at the very least, applications do need to tell the framework which database agent implement to use by appropriately setting the `database_agent_class_name` application property.

## DatabaseResourceAgent

The DatabaseResourceAgent class is a simple facade around connections, prepared statements and result sets. Its main function is to conserve the number of times we get a database connection from the pool and for tracking connections, prepared statements and result sets to make resource management simpler when the resource is no longer needed.

There is another class within the SDK, TranManager, which also has the ability to hand out a database connection. However, the TranManager class simply serves as a raw interface for handing out a configured JDBC connection (whether it originated from the application server's connection pool or was created directly), it does have the resource sharing and management capabilities that the DatabaseResourceAgent exposes. In fact, the DatabaseResourceAgent leverages and sits on top of TranManager and should, therefore, be considered the bona fide interface to use when interacting with the database in Java code.

## Note

There are some cases when using the TranManager directly is appropriate.

A JDBC connection object can be retrieved by constructing a DatabaseResourceAgent instance and using the `getConnection` method. The connection parameters themselves are typically defined by a JNDI datasource. The JNDI name used to load the data source resource and is specified by the datasource application property. If the datasource property is set appropriately, and the application sever has a JNDI datasource set up with connection parameters, then the TranManager and, by extension, the DatabaseResourceAgent should be able to hand out working JDBC connection objects upon application startup after application properties have been initialized. If there is an additional datasource that needs to be accessed by the applications custom code, then the additional datasource name can be specified within application properties by adding a prefix to the datasource property in the application properties file.

## Example

The following two application properties specify the default AgencyPortal datasource to use for the standard tables, as well as an additional datasource for a database that contains some code list tables.

```

datasource=java:comp/env/agencyportal
codelist_db.datasource=java:comp/env/codelist_db

```

In custom Java code, connections to the additional datasource can then be retrieved by specifying the "codelist\_db" prefix string when retrieving connection objects, as the following code illustrates:

```

//using the DatabaseResourceAgent API
DatabaseResourceAgent dra = new DatabaseResourceAgent(this);
try{ dra.getConnection(true, "codelist_db");
PreparedStatement ps = dra.getPreparedStatement(DatabaseAgent.databaseNormalizeSQLStatement(templateSql));
dra.executeUpdate(ps); } catch(SQLException e){ ExceptionLogger.log(e, this.getClass(), "save"); } finally{ dra.closeDatabaseResources(this); }
//using the TranManager directly (condensed to one line of code, for brevity)
Connection conn = new TranManager("codelist_db").getConnectionByJNDI();

```

## Note

The `force_use_jdbc` property can be used to prevent usage of JNDI to retrieve the JDBC Connection parameters and instead derive the values from application properties. This option is typically used for connecting to a database during JUnit tests and in other scenarios where JNDI isn't available.

If `force_use_jdbc` is set to true, the `jdbc_driver`, `database_url`, `database_username` and `database_password` properties become required. Similar to the `database` property described above, these properties can also be prefixed with custom names to support connections to additional databases. However, the `force_use_jdbc` property is specified at the application level (i.e., you can't provide connection parameters via both JDBC and JNDI at the same time).

Every `DatabaseResourceAgent` instance needs an "owner" object. The intent is that the object that created the `DatabaseResourceAgent` would pass itself as the owner. The owner object reference is recorded so that later calls to certain, protected methods can be ignored if they don't pass in the same owner object references as the "caller" parameter. This means that only the owner, or somebody who has access to the owner object reference, can invoke these protected methods. The methods that are protected are methods that can alter the state of the underlying Connection object (i.e., commit operations, rollback operations and close operations). By protecting these operations, a `DatabaseResourceAgent` instance can be shared by the original creator and owner with other, less privileged objects who can use the underlying Connection object without the ability to alter the state of it, as long as they use the prescribed API.

## Note

In cases where the `DatabaseResourceAgent` is constructed with a "Shared Connection," the connection object itself becomes the owner of the instance.

## Example

To get a connection to the default datasource, the following pattern should be used. It is important to note how the Connection object is never assigned to a local variable and operated upon directly, even though it is returned by the `dra.getConnection()` method invocation. Instead, the `PreparedStatement` object is retrieved via the `dra` instance. This is so the `dra` instance can keep track of the `PreparedStatement` object reference and close it when the time comes. The database resources are all closed within a finally block. Crucially, the close operation would be Null-Op if the "caller" object passed into it not the same as the "owner" object, which was initially passed into the `DatabaseResourceAgent` constructor.

```
DatabaseResourceAgent dra = new DatabaseResourceAgent(this); try { dra.getConnection(true); PreparedStatement ps = dra.getPreparedStatement(DatabaseAgent.databaseNormalizeSQLStatement(templateSql)); dra.executeUpdate(ps); } catch(SQLException e){ ExceptionLogger.log(e, this.getClass(), "save"); } finally{ dra.closeDatabaseResources(this); }
```

## Property Reference

The following properties are related to database connectivity:

Property	Description	Default Value	Recommended Setting (if any)
<code>db_table_prefix</code>	You may need to add table prefixing in some installations when the user ID that is provided in the JNDI is not owner of the tables or stored procedures. The table prefix setting instructs the SDK database framework to apply the specific prefix in front of table names. If you use this facility, it is important to add the dot (.) at the end of the prefix value.	""	
<code>datasource_prefix</code>	Used to prefix the JNDI connection name (datasource) with the required prefix. Some application servers require a prefix to be provided as part of the JNDI name. This property can be left blank or the reference can be removed from the database property.	none	

Property	Description	Default Value	Recommended Setting (if any)
datasource	The JNDI name that refers to the database connection parameters. This can also include a prefix, if necessary. The datasource_prefix parameter is not part of the JNDI name setup in your application server. Additional datasource names can be specified by prepending a customprefix (e.g., codelist_db.datasource=codelistDb).	none	
database_agent_class_name	The SDK database agent class to use for the application.	com.agencyport.database.SQLDatabaseAgent	
force_use_jdbc	The force_use_jdbc property can be used to prevent usage of JNDI to retrieve the JDBC Connection parameters and instead derive the values from Application Properties. This option is typically used for connecting to a database during JUnit tests and, in other scenarios, where JNDI isn't available. If force_use_jdbc is set to true, the jdbc_driver, database_url, database_username and database_password properties become required. Connection parameters for additional databases can be specified by prepending a custom prefix to the property name.	false	
jdbc_driver	The JDBC driver class name to use for connection to the database. Required when force_use_jdbc is set to true. Connection parameters for additional databases can be specified by prepending a custom prefix to the property name.		
database_url	The database URL to use for connection to the database. Required when force_use_jdbc is set to true. Connection parameters for additional databases can be specified by prepending a custom prefix to the property name.		
database_username	The database username to use for connection to the database. Required when force_use_jdbc is set to true. Connection parameters for additional databases can be specified by prepending a custom prefix to the property name.		
database_password	The database password to use for connection to the database. Required when force_use_jdbc is set to true. Connection parameters for additional databases can be specified		

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
	by prepending a custom prefix to the property name.		
audit_connection_count	Boolean flag that governs whether each database connection acquisition and release is tracked and logged.	false	Should only be true to help diagnose a database connection leak or overrun.

## Data Model

The data model describes in full detail the core tables and columns used by the framework for AgencyPortal5.2. Click [here](#) to view the Data Model Reference spreadsheet.

# Client-Side Development

This section contains information/guidance relating to client-side development. If you're looking to work in any of the following areas of AgencyPortal, you should read this topic:

- Altering the HTML Markup
- Styling the User Interface
- Client-side programming using JavaScript

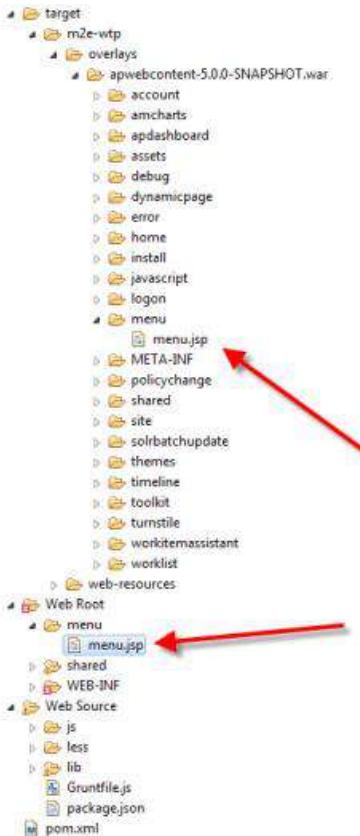
Before we get started, let's take a look at how the default build-process (described in the [Developer Setup](#) section) works.

## Working with SDK Web Artifacts

The Maven build pulls in the SDK's web artifacts from the repository at build time. You can view these artifacts within the target folder after a build, but any changes you make to these files are blown away after the next build.

The way to overwrite the SDK's version of a file is to create your own version of the file under the Web Root folder with a matching directory structure and file name. This way, when build occurs, your version of the file wins the conflict. This practice is commonplace for files, such as site/site\_shell.jsp, home/home.jsp and menu/menu.jsp.

### Example



## Working with Web Sources

The SDK ships with web sources, which live in their own directory and never directly make it into the WAR. These web sources are JavaScript files that are concatenated together and minified, as well as LESS files that are compiled to CSS.

We have provided you with the means to generate these combined/compiled files so you can add your own files into the mix.

## Note

You should not modify any of the existing files under Web Sources (apart from Gruntfile.js). You should treat the files provided to you as you would any third party library code (i.e., if you modify and we change it, then you'll have trouble upgrading to our latest code). As a best practice, when adding new files, add them to a new directory under Web Source called custom.

The source files are combined/compiled by a tool called GruntJS (<http://gruntjs.com/>). The real power of this tool is its ability to generate HTML5 source maps (<http://www.html5rocks.com/en/tutorials/developertools/sourcemaps/>).

You should have already downloaded nodejs as a [prerequisite](#), so you should be able to open a terminal/command prompt and navigate to the Web Source of your project and then run the following two commands to install GruntJS:

```
npm install
```

```
npm install -g grunt-cli
```

## Note

By default, this command will drop some files into a directory called node\_modules. You should not commit this directory to SVN (instead, add it to your SVN or Git ignore list).

Now that GruntJS is installed, you should be able to run the following command to run the build:

```
grunt
```

By default, the Grunt build is set up to output the following files into your project's Web Root/assets/directory:

- js/agencyportal.js
- js/agencyportal.js.map
- js/agencyportal-body.js
- js/agencyportal-body.js.map
- themes/agencyportal/agencyportal.css
- themes/agencyportal/agencyportal.css.map

Since these files are dropped into the Web Root directory, Maven takes care of getting them into the WAR file for us. For this reason, the Grunt build does not need to know anything about where or how you're deploying your code (locally or otherwise). One drawback to this approach is that you'll need to commit these build artifacts to SVN/Git. Otherwise, somebody will have to run the Grunt build, even if they're not working on JavaScript or LESS.

## Note

You may have noticed that these map files can get quite large. The good news is that their presence in the WAR file will not have any negative impact on page load times for regular users. This is because the browser knows not to download source map files until the developer console is opened.

## Altering the HTML Markup

In the Working with SDK Web Artifacts section, we mention how to override one of the JSP files that is shipped with the product. Forking one of these JSP files is a way to get your own custom HTML markup into the mix, but this method should be used as sparingly as possible.

## Note

The reasoning behind the above statement is that when you're taking SDK upgrades, the product versions of the files that you've forked could change. This means that you would have to manually differentiate each of these files for each SDK upgrade you take to make sure that you bring in upstream enhancements. It's not feasible to entirely avoid forking product JSP files, just try to keep the number of files forked to a minimum. Fundamental files, such as site/site\_shell.jsp, home/home.jsp, home/footer.jsp and menu/menu.jsp, are virtually always forked by implementation teams.

The markup with the transaction for fields, buttons and other form elements is, for the most part, dynamically generated using our HTML template mechanism. This mechanism, commonly referred to as HTML element definitions, is overridable and extensible.

## Example 1

**Scenario:** You don't want fieldsets rendered on TDF pages to have legends. You need to go from something like this:

```
<fieldset class="" id="applicantInfo"><legend>Applicant Information</legend> <!-- ... omitted for brevity --> </fieldset>
```

To something like this:

```
<fieldset class="" id="applicantInfo"><!-- no more legend! --> <!-- ... ommited for brevity --> </fieldset>
```

This need can be accomplished by overriding the default HTML template definition that is used for all fieldsets on TDF pages.

1. The default template definitions all live in a file called `html_element_defintions.txt`, which is located inside the apwebapp JAR under the properties directory. Open this file.
2. Find the template definition with an ID of FIELDSET and copy it to your clipboard (where possible, always use an existing template definition as your starting point so you get all the structural elements you need).
3. Update `framework.properties` to bring in an additional HTML template file:  
`html_template_file=${my_context_path}WEB-INF/custom_html_definitions.txt`.
4. Create the file where you told the app to look for it (WEB-INF) and paste the FIELDSET definition that we copied in step 2.
5. Edit the custom template definition, as needed, and restart the application to see the changes take effect.

## Example 2

**Scenario:** There is a need to add a HTML anchor tag directly underneath a special text field. You are aware that the built-in `fieldHelpers` type script or balloon could potentially achieve the same goal, but the UI designer has indicated that a textual anchor tag is preferable.

Unlike the previous example, in this scenario we want our change to affect only one special text field in the TDF, not all text fields. In this case, we're not overriding/replacing an existing template, we're adding a brand new one that only one field will use.

1. The default template definitions all live in a file called `html_element_defintions.txt`, which is located inside the apwebapp JAR under the properties directory. Open up this file.
2. Find the template definition with an ID of TEXT and copy it to your clipboard (where possible, always use an existing template definition as your starting point so you get all the structural elements you need).
3. Update `framework.properties` to bring in an additional HTML template file:  
`html_template_file=${my_context_path}WEB-INF/custom_html_definitions.txt`.
4. Create the file where you told the app to look for it (WEB-INF) and paste the TEXT definition that we copied in step 2. The existing template should look something like this:  
`*BEGIN TEXT`
5. Modify the template as follows:
  1. Since you don't want your change to apply to all TEXT fields, you need to change the ID. Let's change it to `COOL_TEXT_WITH_HYPERLINK`.
  2. Add the anchor tag to the markup. Let's add the following markup at the bottom of the interior div:  
`<div class="cool-text-hyperlink-container">`  
 `<!-- hyperlink which opens a lightbox or something fancy`  
 `like that-->`  
 `<a href="javascript:void" id="aggregate-link">Aggregate</a>`  
`</div>`
6. Now you need to tell your field in the TDF to use the custom template definition.  
`<fieldElement type="text" htmlDefitionId="COOL_TEXT_WITH_HYPERLINK" uniqueId="annualExposure" id="WorkCompLineBusiness.WorkCompRateState.WorkCompLocInfo.WorkCompRateClass.Exposure" label="Est. Annual Exposure" size="10" maxLength="10">`  
 `<validation type="numeric" id="N343" />`  
`</fieldElement>`

## Styling the User Interface

There are several technologies at play on the client side to make the user interface look the way it does.

## LESS

Less is a CSS pre-processor, meaning that it extends the CSS language, adding features that allow variables, mixins, functions and many other techniques that allow you to make CSS that more maintainable, themable and extendable. You can make your own LESS files and use the GruntJS process to compile them into agencyportal.css. Otherwise, you can stick to branding with plain old CSS.

Learn more about Less [here](#).

## Bootstrap

Bootstrap is an HTML, CSS and JS framework for developing responsive, mobile first projects on the web. It works well with a CSS pre-processor like LESS and is full of layout and module features AgencyPortal relies on.

### GRID SYSTEM

Refer to the following on Grid System:

- [Grid System](#)
- [Examples](#)

Learn more about Bootstrap [here](#).

## Font Awesome

Font Awesome is a free library of scalable vector icons. An all-CSS/HTML alternative to images, Font Awesome renders scalable vector icons with its own font. It's equally important to AgencyPortal as Bootstrap is. This is how it's done:

- Font Awesome library: "/apwebcontent/Web Source/lib"
- Font Awesome variables: "/apwebcontent/Web Source/lib/font-awesome-4.2.0/less/variables.less"
- Font Awesome fonts: "/apwebcontent/Web Source/lib/font-awesome-4.2.0/fonts/"

We use the generic `<i class="fa"></i>` in the markup, centralizing actual font decisions in our own variables file.

- AP CSS: "/apwebcontent/Web Root/assets/themes/agencyportal/agencyportal.css", which is a compiled file of...
- AP LESS "/apwebcontent/Web Source/less"
- AP variables "/apwebcontent/Web Source/less/variables.less"

To get started, focus on the AP variables:

- line 150: `@font-path: "../../fonts";`
- line 151: `@fa-font-path: "@{font-path}/font-awesome/";`

## Note

The font path can be easily modified, in case you want to go with some other picture font vendor. The icons are defined as follows:

```
line 53: /* Font Awesome Icons ----- */

@editableIconName: @fa-var-fooIconName;
```

The left column has AP-defined icon names. Like CSS, these names can be edited. However, your names must point to existing names in the Font Awesome variables file.

## Example

```
@icon-workitem-workers-comp: @fa-var-wrench;
```

Individual icons are defined in section LESS files, using a parent class selector to define the icon.

## Example

```
from "/apwebcontent/Web Source/less/cards.less":
```

```
&.WORK .card-content.fa { &:before { content: @icon-workitem-workers-comp; } }
```

This is the structure that allows us to use the generic `<i class="fa"></i>` in the markup. For instance, what if you don't want the Workers Comp icon to be a wrench? If the specific icon was embedded, you'd have to edit a bunch of JSP and JavaScript files. This solution centralizes Font Awesome decisions in one file.

Learn more about Font Awesome from the following:

- [Font Awesome](#)
- [Icons](#)

## Branding AgencyPortal

There are two methods of creating customAgencyPortal themes: CSS and LESS. The CSS method is exactly what we've already been doing for previous AgencyPortal versions: duplicating and renaming our default theme. The LESS method pretty much does the same thing, but with AgencyPortal 5.2's new environment files.

### LESS Method

Remember, you should not modify any of the Web Source files that came with the SDK (with the exception of the Gruntfile.js). Start by creating your own main LESS file. This new LESS file should import the agencyportal.less file. You can update the LESS target of the Gruntfile.js build file to point to your new file as the main LESS file.

Less is a CSS preprocessor, meaning that it extends the CSS language, adding features that allow variables, mixins, functions and more. We use a compiler (NodeJS or Codekit) to push LESS values into CSS (specifically SDK web artifact's "assets/themes/agencyportal/agencyportal.css" file).

Our Less framework is controlled in /apwebcontent/Web Source/less/agencyportal.less, a collection of imported third-party dependencies (Bootstrap, Font Awesome, etc.) and our core styles (which overwrite Bootstrap defaults in "Web Source/lib/bootstrap-3.1.1/less/").

We need to make sure specific styles don't accidentally apply to the entire app.

### Example

```
.card { .row.card-actions { .card-button { .dropdown-menu { padding: 5px 0; } li > span { clear: both; display: block; color: @gray; font-weight: normal; padding: 3px 20px; white-space: nowrap; &:hover { color: #fff; background: @theme-active-color; } } } } }
```

Compiles as:

```
.card .row.card-actions .card-button .dropdown-menu { padding: 5px 0; } .card .row.card-actions .card-button .dropdown-menu > li > span { clear: both; display: block; color: @gray; font-weight: normal; padding: 3px 20px; white-space: nowrap; } .card .row.card-actions .card-button .dropdown-menu > li > span:hover { color: #fff; background: @theme-active-color; }
```

### Best Practice

As a best practice, any new Web Source files should be created under Web Source/custom/less.

### CSS Method

Using the CSS method is the simpler of the two methods, since you're not keeping track of LESS file compilations, library locations or versions of dependency assets. All you have to do is:

1. Find the default agencyportal theme. It is located along with the SDK's web artifacts under "assets/themes/agencyportal/agencyportal.css." (This CSS file is the compiled results of all our LESS files.)
2. Copy the folder /themes/agencyportal/ and paste it into /themes/ as a sibling of /themes/agencyportal/. Rename the copy and its enclosed CSS with the client name (e.g., /sdk/apwebcontent/Web Root/assets/themes/fooClientName/fooClientName.css).
3. Find our site shell /sdk/apwebcontent/Web Root/site/site\_shell.jsp. Copy the HEAD link to agencyportal.css:`<link href="${pageContext.request.contextPath}/assets/themes/agencyportal/agencyportal.css" rel="stylesheet">`and paste it underneath it (keeping track of the C in CSS). Rename the destination with our new client name (e.g., `<link href="${pageContext.request.contextPath}/assets/themes/fooClientName/fooClientName.css" rel="stylesheet">`).
4. Open fooClientName.css, select all, delete. Then, start overwriting the default styles one at a time. Modify images as needed, but retain their names ("logo.png", etc.).

Test on our supported browsers while working. Artistically, thread the needle between (a) matching the client's brand and (b) making our app look terrific. Study the client's website for web-specific style (page design, navigation, link color, font, headers, TABLEs, button and FORMs, etc.).

## Client-Side Programming

Unlike the Less compilation process that outputs one CSS file, the JavaScript aggregation and minification process outputs two JS files:

- agencyportal.js
- agencyportal-body.js

The difference between the two files is that agencyportal.js is added to the DOM by site\_shell.js under the <head> section and agencyportal-body.js is added to the DOM as the last child of the <body> section. The following is a snippet of site\_shell.jsp. For performance reasons, most, if not all, JavaScript code you write on a custom basis should be minified and appended to the agencyportal-body.js.

```
<html lang="en"> <head> <!-- other content omitted for brevity --> <link href="${pageContext.request.contextPath}/assets/themes/agencyportal/agencyportal.css" rel="stylesheet"><script src="${pageContext.request.contextPath}/assets/js/agencyportal.js" type="text/javascript"></script> </head> <body> <%= bodyClass %> <div class="agencyportal-5"> <!-- Navigation Bar --> <jsp:include page="../menu/menu.jsp" flush="true" /> <!-- Page Body (Editable Content) --> <jsp:include page="<%np%>" flush="true" /> </div> <script src="${pageContext.request.contextPath}/assets/js/agencyportal-body.js" type="text/javascript"></script> </body> </html>
```

SDK JavaScript and third party library files (including jQuery) are aggregated into these two files by Grunt and since we've distributed the source files and build process to project teams, you can get your custom JavaScript files into these aggregated files too.

### Tip

Review Gruntfile.js to see which files/libraries are being included into which aggregated JavaScript file and in what order.

As best practice, to optimize page load times in production, we recommend that as much custom JavaScript as is feasible should be included in these aggregated files. The challenge is that most custom JavaScript that project teams create needs to be in standalone JavaScript files that are dynamically included by TDF pages. Therefore, we came up with this standard to minimize the quantity of JavaScript files and the file size of those JavaScript files. The standard is comprised of two rules.

1. **Create a maximum of one JavaScript file per TDF page.** This is beneficial for two reasons:
  1. Fewer files for the browser to download improves page load times.
  2. Having only one file on a TDF page eliminates the risk of having multiple files conflict with each other when they both override the same hookpoint callback (such as ap.page.onInitialize - in this scenario, the ap.page object would only invoke the onInitialize function of whichever file gets loaded last. Therefore, one file would need to have special logic to invoke the hookpoint function belonging to the other file).
2. **Only initialize and delegate in your TDF page's JavaScript file.** All of the real heavy lifting should be done by helper objects, which are defined in files that get aggregated into agencyportal.js or agencyportal-body.js. Limiting the JavaScript file hanging off your TDF page to a simple delegator minimizes its file size. It also makes for very simple and readable boilerplate code.

### Example

**Scenario:** You're working on a new business transaction for commercial auto and you want to build a driver information page that is interactive. The UI design specification calls for an MVR modal along with other special client-side functionality that will need to be powered by JavaScript.

To follow the above pattern correctly, you'll first need a file (or set of files) that will do all of the heavy lifting. Let's create mvrHelper.js under Web Source/custom/js:

```
//All new objects should be defined under a namespace for our client, prestige worldwide. //Make sure this namespace is (conditionally) defined at the beginning of every file that //uses it so we dont need to worry about which file gets defined first var prestige = prestige || {};//Define our new MVRHelper constructor but dont construct an instance here since we dont //have an unnecessary instance taking up memory on every page prestige.MVRHelperCtor = function () { //Begin defining object... //Create some properties this.state = ""; this.driverName = ""; //Create the functions that do the work this.showLightbox = function () { //TODO - implement this }; this.makeMVRCall = function () { //TODO - imagine lots of code here }; this.calculateDriverAge = function () { //TODO - dont you love working with dates in JavaScript?? }; };
```

Since this file will live under Web Source/custom/js, we need to modify the Grunt build to get it as part of the aggregation process. In Gruntfile.js, simply add the filepath to the headJSFiles array:

```
var headJSFiles = ['lib/es5-shim/es5-shim.js', 'lib/jquery/jquery-1.10.2.js', /* several lines omitted for brevity */ 'lib/amcharts/jspdf.js', 'lib/amcharts/jspdf.plugin.addimage.js', 'custom/js/mvrHelper.js'];
```

Now that prestige.MVRHelperCtor construction will be available on every page, you just need to add code to construct an object and use it on your driver information page. Under Web Root/assets/js/custom, create a driverInfo.js file:

```
//Initialize our object during the ap.page.onInitialize hookpoint ap.page.onInitialize = function () { //we add the new object to the prestige namespace so that it will be scoped appropriately //for other functions to access it prestige.mvrHelper = new prestige.MVRHelperCtor; }
```

If we add this script to the TDF page (pageElement type "javascript") it will be included when the page loads. The SDK's JavaScript will invoke the ap.page.onInitialize hookpoint during page initialization (refer to the [Hookpoints](#) section for more information). Now that the presitge.mvrHelper is ready to use, a fieldHelper type "script" that uses it could look something like the following:

```
<fieldElement type="date" id="CommAutoLineBusiness.CommlDriver.DriverInfo.PersonInfo.BirthDt" label="Date of Birth" size="12" maxLength="10" groupId="CommAutoLineBusiness.CommlDriver.DriverInfo.PersonInfo"> <fieldHelper type="script" src="presitge.mvrHelper.showLightbox();"></fieldHelper> </fieldElement>
```

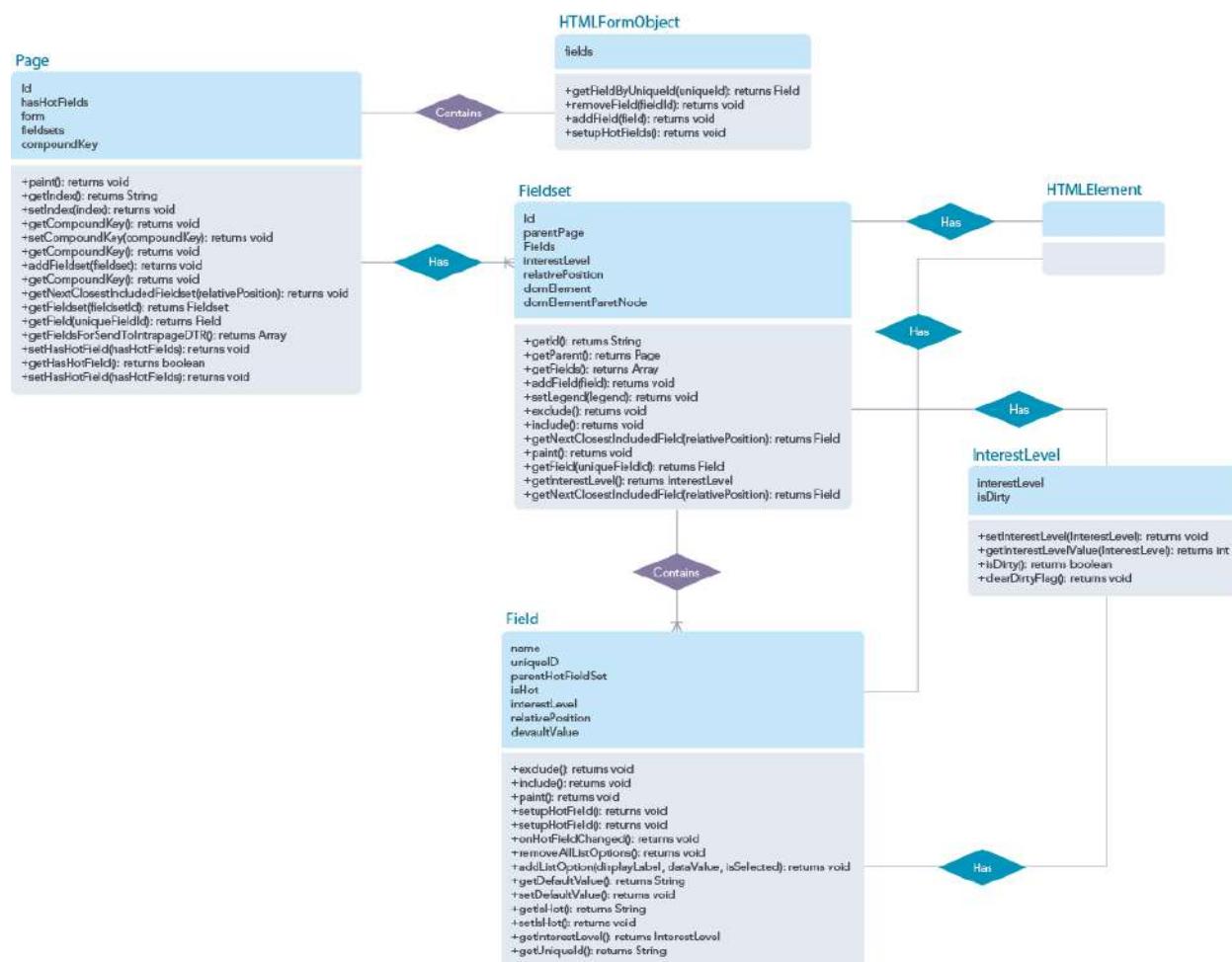
## Base Classes

The AgencyPortal framework creates a handful of core JavaScript base objects for each dynamically generated page. These files are defined in ap\_javascript\_objects.js. Refer to [JSDoc](#) for specifics.

### Note

Runtime information about these objects is available on the JavaScript panel of the Debug Console.

#### [AP JavaScript Page Class Model](#)



## Base Functions

The framework contains a set of utility functions commonly used by framework JS components and custom application code. These functions can be found in the ap\_javascript\_base.js file. An index of its methods follows:

### **addLoadEvent**

`addLoadEvent` can be used to associate the execution of a function with the `window.onload` intrinsic event.

## **Arguments**

{func} - the function that should be attached to window.onLoad

## **Example**

```
addLoadEvent(function() { alert("Hello World"); })
```

## **addSlashes**

Add slashes is a function that has been created to add escape characters to an Agencyport XPath like expression, which may contain single quote characters. If left unescaped, these characters can cause unexpected behavior within JavaScript code.

Add slashes can be used anytime that a developer is writing JavaScript, which creates additional lines of JavaScript for later execution and wishes to ensure that the JavaScript is sanitized for apostrophes.

## **Arguments**

string - a string whose apostrophes should be escaped.

## **Example**

```
var input = document.getElementById("Producer.PhoneInfo[PhoneTypeCd='Phone']");
var a = document.createElement('a');
a.href = "javascript:document.getElementById('" + addSlashes(input.id) + "').value = 'hello!';"
```

## **getElementsByClassName**

getElementsByClassName returns all requested elements given a search root, a desired tag name, and a string 'class name'.

## **Arguments**

object - the root element at which point to start the search. Expects a DOM Element.

string (tag name) - the desired element types to check.

object [class name(s)] - the desired class names to find.

## **Example**

```
var tips = getElementsByClassName(document, "div", "tip_container")
```

## **Additional Reference**

<http://www.snook.ca/jonathan>

## **getElementX**

getElementX is a function that will return the x coordinate of the top left most corner of a given element relative to the absolute corner of the browser window.

getElementX can be used anytime a developer desires to position content absolutely (often relative to a triggering element) in a z-index greater than the majority of the content on the page.

## **Arguments**

element - a valid DOM element who's position should be evaluated

## **Example**

```
//get the coordinate position of a link on the page so that content can be absolutely over top of it.
var menuLink = document.getElementById("main_menu_link");
var menuDiv = document.createElement("div");
var x = getElementX(menuLink);
var y = getElementY(menuLink);
menuDiv.style.position = 'absolute';
menuDiv.style.zIndex = '2';
menuDiv.style.left = x + "px";
menuDiv.style.top = y + "px";
```

## **getElementY**

getElementY is a function that will return the Y coordinate of the top left most corner of a given element relative to the absolute corner of the browser window.

`getElementY` can be used anytime a developer desires to position content absolutely (often relative to a triggering element) in a z-index greater than the majority of the content on the page.

#### **Arguments**

`element` - a valid DOM element who's position should be evaluated

#### **Example**

```
//get the coordinate position of a link on the page so that content can be absolutely over top of it. var menuLink =
document.getElementById("main_menu_link"); var menuDiv = document.createElement("div"); var x = getElementX(menuLink); var y =
getElementY(menuLink); menuDiv.style.position = 'absolute'; menuDiv.style.zIndex = '2'; menuDiv.style.left = x + "px"; menuDiv.style.top = y
+ "px";
```

### **getFieldsetByLegend**

`getFieldsetByLegend` is a function that can be used to return the first instance in the document of a fieldSet with a given legend.

`getFieldsetByLegend` is helpful to developers writing code that directly impacts the user interface around the HTML Form. This can be used for example to round up a fieldSet on the page and show it or hide it contingent on some action.

#### **Arguments:**

A string

#### **Example**

```
var optionalCoverages = getFieldsetByLegend("optionalCoverages"); var toggle = document.getElementById("oc_toggle"); if (toggle == 'yes')
optionalCoverages.style.display = 'block'; else optionalCoverages.style.display = 'none';
```

### **getParentByClassName**

`getParentByClassName` is a function that can be used to return the first parent instance with a given class attribute. It is meant to augment the standard DOM methods providing a shortcut for developers.

`getParentByClassName` should be used anytime a developer needs to gain access to the first parent instance of an element with a given class attribute. That parent may or may not be the immediate parent of the child element in the DOM hierarchy.

#### **Arguments**

`element` - a valid DOM element who's parentage should be evaluated.

`className` - the value of the class attribute on the sought after parent.

#### **Example**

```
//find the master DIV element for the buttons on a page var continue = document.getElementById('continueButton'); var mainButtonContainer
= getParentByClassName(continue, "buttons");
```

### **getParentByTagName**

`getParentByTagName` is a function that can be used to return the first parent instance of a specified entity type given a child. It is meant to augment the standard DOM methods providing a shortcut for developers.

`getParentByTagName` should be used anytime a developer needs to gain access to an element's parent of a given tag name, where that parent may or may not be the immediate parent of the child element in the DOM hierarchy.

#### **Arguments**

`element` - a valid DOM element who's parentage should be evaluated.

`parentType` - the tag name of the parent being sought.

#### **Example:**

```
//find the fieldSet to which an HTML input element belongs var input = document.getElementById('myInputId'); var fieldset =
getParentByTagName(input, "FIELDSET");
```

## getUniqueId

getUniqueId can be used to get a unique ID for programmatically generated DOM nodes. getUniqueId will make up random numbers, see if they're in use, and return the first one that is not.

getUniqueId should be used anytime a developer is appending a node to an in memory DOM and needs to insure that their appended node has a unique id with which they can refer to it.

### Arguments

No arguments are required. Returns an ID string unique to the DOM.

### Example

```
var myNode = document.createElement("div"); var newId = getUniqueId(); myNode.id = newId;
```

## getXMLHTTP

Commonly known as Ajax, getXMLHTTP returns an XML HTTP request object from which requests can be made of the server from within a JavaScript program. Usage of the getXMLHTTP function for formulation of XML HTTP Requests isolates the developer from having to deal with differences in implementation across browser types.

getXMLHTTP should be used anytime a JavaScript developer requires a connection to the server be created so that additional data can be provided on-demand to their script.

### Arguments

No parameters are required by getXMLHTTP. It returns a single request object.

### Example

```
var requestURL = "POLKQuery.jsp"; var polkResult = "something"; function polkLookupHelper(polkResults, fieldElement) { xCoordinate = getElementX(fieldElement); yCoordinate = getElementY(fieldElement); overlib(polkResults, FIXX, xCoordinate, FIXY, yCoordinate, CAPTION, ", STICKY, CLOSECLICK, TIMEOUT, 5000); } function getPolkResults(fieldElement){ var serverRequest = getXMLHTTP(); serverRequest.open("GET", requestURL + "?vin=" + fieldElement.value, true); serverRequest.onreadystatechange=function() { if (serverRequest.readyState==4 && serverRequest.status == 200){ alert(serverRequest.responseText); polkLookupHelper(serverRequest.responseText, fieldElement); } } serverRequest.setRequestHeader('Accept','message/x-formresult'); serverRequest.send(null); }
```

### Additional Reference – Ajax Resources

To familiarize yourself with the general concept of Ajax, see: <http://en.wikipedia.org/wiki/AJAX>

General discussion on the topic can be found at <http://ajaxian.com/>

Blogger Alex Bosworth has authored a piece titled "AJAX Mistakes" which can be very helpful to developers considering implementing functionality in their applications that make use of Ajax. The Article can be found at: <http://alexbosworth.backpackit.com/pub/67688>

Also helpful on this front is a piece titled "Places to use Ajax", which can be found [here](#).

## isArray

Often during execution of a script it can be helpful to understand the type of variables that you're working with. isArray is a function that has been included for this purpose, and will test if a given variable is an array or not.

Is array should be used anytime execution of your script is contingent on knowing if a variable is an array.

### Arguments

isArray requires one argument and returns true or false based on it's evaluation of that argument.

### Example

```
var bar = 0; var foo = new Array(1,2,3); //evaluates to false if (isArray(bar)) alert('bar!'); //evaluates to true if (isArray(foo)) alert('foo!');
```

## registerEvent

`registerEvent` allows a JavaScript developer to associate an action with an intrinsic event related to an element. It is a programmatic way of accomplishing what can be done through custom HTML with attributes like "onblur" or "onclick". Usage of the `registerEvent` function isolates the JavaScript author from differences between W3C DOM compliant browser syntax, and MSIE custom syntax. For more information on intrinsic events see: <http://www.w3.org/TR/html4/interact/scripts.html#events>

`registerEvent` should be used when an action needs to be hinged off of an intrinsic event related to an element that has already been rendered, and thusly that the developer does not have the ability to add an intrinsic event to it directly as an attribute.

### Arguments:

- `element` - a valid DOM element to which the event should be registered.
- `eventType` - the intrinsic event type (i.e., 'blur', 'submit', 'select', 'keypress', etc) on which the event should fire.
- `event` - the name of the function to be called when the `eventType` occurs for the element. A word of caution - `event` does not accept parameters. If arguments are required for the event, the author must use other means of conveying them.

### Example:

```
function helloWorld() { alert("hello world"); } foo = document.getElementById("foo"); registerEvent(foo, "mouseover", helloWorld);
```

### Additional Reference – HTML Intrinsic Events

<b>load</b>	The load event occurs when the useragent finishes loading a window or all frames within a FRAMESET. This may be used with BODY and FRAMESET elements.
<b>unload</b>	The unload event occurs when the useragent removes a document from a window or frame. This may be used with BODY and FRAMESET elements.
<b>click</b>	The click event occurs when the pointing device button is clicked over an element. This may be used with most elements.
<b>dblclick</b>	The dblclick event occurs when the pointing device button is double clicked over an element. This may be used with most elements.
<b>mousedown</b>	The mousedown event occurs when the pointing device button is pressed over an element. This may be used with most elements.
<b>mouseup</b>	The mouseup event occurs when the pointing device button is released over an element. This may be used with most elements.
<b>mouseover</b>	The mouseover event occurs when the pointing device is moved onto an element. This may be used with most elements.
<b>mousemove</b>	The mousemove event occurs when the pointing device is moved while it is over an element. This may be used with most elements.
<b>mouseout</b>	The mouseout event occurs when the pointing device is moved away from an element. This may be used with most elements.
<b>focus</b>	The focus event occurs when an element receives focus either by the pointing device or by tabbing navigation. This may be used with the following elements: A, AREA, LABEL, INPUT, SELECT, TEXTAREA, and BUTTON.
<b>blur</b>	The blur event occurs when an element loses focus either by the pointing device or by tabbing navigation. It may be used with the same elements as onfocus.
<b>keypress</b>	The keypress event occurs when a key is pressed and released over an element. This may be used with most elements.
<b>keydown</b>	The keydown event occurs when a key is pressed down over an element. This may be used with most elements.
<b>keyup</b>	The keyup event occurs when a key is released over an element. This may be used with most elements.
<b>submit</b>	The submit event occurs when a form is submitted. It only applies to the FORM element.

<b>reset</b>	The reset event occurs when a form is reset. It only applies to the FORM element.
<b>select</b>	The select event occurs when a user selects some text in a text field. This may be used with the INPUT and TEXTAREA elements.
<b>change</b>	The change event occurs when a control loses the input focus and its value has been modified since gaining focus. This applies to the following elements: INPUT, SELECT, and TEXTAREA.

## Hookpoints

### Running Custom JS when the Page Loads

TDF:

```
<page id="generalInfo" title="Agency/Applicant Information" type="dataEntry" ><pageElement type="script" contentType="text/javascript" src="sayHello.js" /></page>
```

JS (sayHello.js):

```
page.onInitialize = function() { alert("Hello World!"); }
```

### Running Custom JS when the Page is Submitted

TDF:

```
<page id="generalInfo" title="Agency/Applicant Information" type="dataEntry" ><pageElement type="script" contentType="text/javascript" src="sayHello.js" /></page>
```

JS (sayHello.js):

```
page.onSubmit = function() { alert("Hello World!"); }
```

### Running Custom JS when the Roster Add Section is Opened

TDF:

```
<page id="generalInfo" title="Agency/Applicant Information" type="dataEntry" ><pageElement type="script" contentType="text/javascript" src="sayHello.js" /></page>
```

JS (sayHello.js):

```
page.onAddOpen = function() { alert("Hello World!"); }
```

### Running Custom JS when the Roster Add Section is Closed

TDF:

```
<page id="generalInfo" title="Agency/Applicant Information" type="dataEntry" ><pageElement type="script" contentType="text/javascript" src="sayHello.js" /></page>
```

JS (sayHello.js):

```
page.onAddClose = function() { alert("Hello World!"); }
```

### Running Custom JS when a Fields Value is Set

TDF:

```
<page id="generalInfo" title="Agency/Applicant Information" type="dataEntry" ><pageElement type="script" contentType="text/javascript" src="sayHello.js" /></page>
```

JS (sayHello.js):

```
//get one of the field objects that's on the page var foo = page.form.getFieldByName("bar"); //register an event to run when code runs
bar.setValue() bar.onSetValue = function() { alert("Hello World!"); }
```

## Note

This code will only be invoked if the setter is updating the value using the `Field.setValue()` method.

## Note

The roster action hook points described below are available in Framework versions 3.6.000019 and later.

### Running Custom JS when the Roster Delete Button is Pressed

TDF:

```
<page id="vehicles" title="Vehicles" type="roster"><pageElement type="script" contentType="text/javascript" src="sayHello.js" /> ... </page>
JS (sayHello.js):

//register an event to run when the user hits the Delete button next to a roster item page.onRosterDelete = function() { alert("Hello World!"); }
```

## Note

If `onRosterDelete()` returns false, the roster delete action will be canceled.

At the time of user action, the `activeRosterActionAnchor` property is set on the page object. That property contains a reference to the HTML Anchor(A) that triggered the event. Application developers can evaluate and modify its properties (e.g., href or search) directly before the action is taken.

### Running Custom JS when the Roster Edit Button is Pressed

TDF:

```
<page id="vehicles" title="Vehicles" type="roster"><pageElement type="script" contentType="text/javascript" src="sayHello.js" /> ... </page>
JS (sayHello.js):

//register an event to run when the user hits the Edit button next to a roster item page.onRosterEdit = function() { alert("Hello World!"); }
```

## Note

If `onRosterEdit()` returns false, the roster edit action will be canceled.

At time of user action the `activeRosterActionAnchor` property is set on the page object. That property contains a reference to the HTML Anchor(A) that triggered the event. Application developers can evaluate and modify its properties (e.g., href or search) directly before the action is taken.

### Running Custom JS when the Roster Recover Button is Pressed

TDF:

```
<page id="vehicles" title="Vehicles" type="roster"><pageElement type="script" contentType="text/javascript" src="sayHello.js" /> ... </page>
JS (sayHello.js):

//register an event to run when the user hits the Recover button next to a roster item page.onRosterRecover = function() { alert("Hello World!"); }
```

## Note

If `onRosterRecover()` returns false, the roster recover action will be canceled.

At time of user action the property `activeRosterActionAnchor` is set on page object. That property contains a reference to the HTML Anchor(A) that triggered the event. Application developers can evaluate and modify its properties (e.g., href or search) directly before the action is taken.

## Third Party Libraries

The following is a list of the libraries included in AgencyPortal 5.2:

Library	Used for	Keep?	License	URL
angularjs	JavaScript framework used for client-side application development	Yes	MIT	<a href="https://angularjs.org/">https://angularjs.org/</a>
animate.css-master	Simplification of basic CSS animations	Probably	MIT	<a href="http://daneden.github.io/animate.css/">http://daneden.github.io/animate.css/</a>
bootstrap-3.1.1	Front end framework	Yes	MIT	<a href="http://getbootstrap.com/">http://getbootstrap.com/</a>
jquery-ui	UNKNOWN; apparently dragged in by the jQuery File Upload as a dependency.	No (if can be avoided)	MIT	<a href="http://jqueryui.com/">http://jqueryui.com/</a>
jQuery-File-Upload	UNKNOWN; apparently brought in to support file upload for the Turnstile upload process.	Probably	MIT	<a href="http://blueimp.github.io/jQuery-File-Upload/">http://blueimp.github.io/jQuery-File-Upload/</a>
jquery-visible-master	jQuery plugin used to check whether an element is visible to the user	Yes	MIT	<a href="https://github.com/teamdf/jquery-visible/">https://github.com/teamdf/jquery-visible/</a>
bootstrap-datepicker-master	Date pickers within the application	Yes	Apache 2.0	<a href="http://eternicode.github.io/bootstrap-datepicker">http://eternicode.github.io/bootstrap-datepicker</a>
jquery	JavaScript library	Yes	MIT	<a href="http://jquery.com/">http://jquery.com/</a>
bootstro.js-master	Welcome to AgencyPortal tour	Yes	MIT	<a href="http://clu3.github.io/bootstro.js">http://clu3.github.io/bootstro.js</a>
es5-shim	UNKNOWN		MIT	<a href="https://github.com/es-shims/es5-shim">https://github.com/es-shims/es5-shim</a>
modernizer	Browser feature detection	No (if can be avoided)	MIT	<a href="http://modernizr.com/">http://modernizr.com/</a>
font-awesome-4.0.3	Iconography	Yes	MIT	<a href="http://fortawesome.github.io/Font-Awesome/">http://fortawesome.github.io/Font-Awesome/</a>
respond	min/max-width CSS3 media queries for IE 6-8	Yes	MIT	<a href="https://github.com/scottjehl/Respond">https://github.com/scottjehl/Respond</a>
formatmask	Input mask library for jQuery	Yes	MIT	<a href="http://digitalbush.com/projects/masked-input-plugin">http://digitalbush.com/projects/masked-input-plugin</a>
spin.js	UNKNOWN; apparently brought in for Turnstile upload. Unclear why Bootstrap progress meters were insufficient.	No (if can be avoided)	MIT	<a href="http://fgnass.github.io/spin.js/">http://fgnass.github.io/spin.js/</a>
handlebars	Page templating	Yes	MIT	<a href="http://handlebarsjs.com/">http://handlebarsjs.com/</a>
timelinejs	UNKNOWN	No	MPL 2.0	<a href="https://github.com/NUKnightLab/TimelineJS">https://github.com/NUKnightLab/TimelineJS</a>
html5shiv	Adds HTML5 elements to IE 8	Yes	MIT	<a href="https://code.google.com/p/html5shiv/">https://code.google.com/p/html5shiv/</a>

Library	Used for	Keep?	License	URL
typeahead	Typeahead find for search boxes	Yes	MIT	<a href="http://twitter.github.io/typeahead.js/">http://twitter.github.io/typeahead.js/</a>

## REST API

In AgencyPortal 5.1, we began the process of opening up the SDK's inner working with a REST (Representational State Transfer) API to help developers create their applications more easily and provide additional maintainability via a clear separation between the "front-end" and "back-end" sides of the application.

REST is an architecture style that uses HTTP to make calls between machines in networked applications. An application that has implemented a REST API can use HTTP requests to perform CRUD (create, read, update, delete) operations. For more detailed information on REST, refer to Dr. Elkstein's [Learn REST: A Tutorial](#) site.

Implementing this type of technology helps pave the way for a more open architecture to expose portal core functions over an open service oriented platform. This type of platform can support other applications from other channels, such as self-service or customer applications. In Phase I of this initiative, we've revamped the AgencyPortal work list by using a set of data services based on REST. Refer to the [Work List Architecture](#) topic for more information.

The response data structures associated with these REST data service calls conform to a consistent and standard data structure and support both JSON and XML formats. All REST resources internally inter-operate with provider interfaces that provide the build of the implementation. Project teams can extend/implement these provider interfaces to enhance, augment or alter default product behavior due to their factory design pattern. Extension of REST resources is not supported since the annotation discovery mechanisms in Jersey (or other JAX-RS implementations) do not correctly disambiguate identifiable annotations between a base class and its possible extension.

## AngularJS

AngularJS is an extensible JavaScript framework that allows developers to create world-class responsive user interfaces and super fast applications that are completely in sync with today's modern technologies. Incorporating an AngularJS framework makes pages load faster, more responsive, easier to modify and more maintainable. For more information on AngularJS, refer to the [AngularJS](#) website.

In 5.1, we introduced AngularJS into AgencyPortal within the My Accounts and My Work pages.

# PDF Generation

AgencyPortal provides two PDF Generation utilities to map and generate PDF forms:

- [Legacy PDF Generation Utility](#)
- [PDF Generation Utility](#)

## Note

We recommend that you use the PDF Generation utility to map and generate PDF forms.

# Legacy PDF Generation Utility

The AgencyPortal SDK includes a PDF core facility with support for an open source PDFBox library.

ACORD forms are filled out with ACORD XML for a work item per page layout. A page layout (a proprietary XML) contains one or more field layout elements; one for each field on the ACORD PDF. Each file element has its own unique ID attribute, which is used for identification purposes. The PDF x, y coordinates, font size and size are contained on each field layout element as shown in the following XML for the ACORD 130 form:

```
<pdf>
 <layout name="ACORD_130_LAYOUT">
 <page pageNum="1" scaleFactor="0.12">
 <field name="FormDate" font="3" x="7.25" y="60" type="2" maxWidth="10" maxHeight="10" pdfDesc="Form date"/>
 <!-- Producer Box Fields-->
 <field name="AgencyInfo" font="3" x=".35" y=".92" type="2" maxWidth="40" maxHeight="40" pdfDesc="agency name"/>
 <field name="Producer.ProducerName" font="3" x="1.12" y="1.8" type="2" maxWidth="40" maxHeight="10" pdfDesc="Producer Name"/>
 <field name="Producer.CSRepresentativeName" font="3" x="1.26" y="1.95" type="2" maxWidth="40" maxHeight="10" pdfDesc="CS Representative Name"/>
 <field name="Form.Agency.Phone" font="3" x="1.38" y="2.13" type="2" maxWidth="20" maxHeight="10" pdfDesc="agency phone"/>
 <field name="Producer.CellPhone" font="3" x="1.73" y="2.29" type="2" maxWidth="40" maxHeight="10" pdfDesc="Agency Mobile Phone"/>
 <field name="Form.Agency.Fax" font="3" x="1.73" y="2.46" type="2" maxWidth="20" maxHeight="10" pdfDesc="agency fax"/>
 <field name="Producer.Party.Communications.EmailAddress" font="3" x="1.75" y="2.62" type="2" maxWidth="30" maxHeight="10" pdfDesc="agency email"/>
 <field name="AgencyCode" font="3" x="0.61" y="2.78" type="2" maxWidth="10" maxHeight="10" pdfDesc="agency name"/>
 <field name="Agency/SubCode" font="3" x="2.36" y="2.78" type="2" maxWidth="10" maxHeight="10" pdfDesc="agency name"/>
 <field name="Agency.CustomerID" font="3" x="1.32" y="2.96" type="2" maxWidth="40" maxHeight="10" pdfDesc="agency name"/>
 </page>
 </layout>
</pdf>
```

The coordinates (x,y) start with the upper left corner as the origin in inch.

## Example

The AgencyInfo field on the PDF starts at 0.35 inches from the left and 0.92 inches from the top, and has a maximum of 40 characters, as shown in the green box of the PDF template.

The form is a "WORKERS COMPENSATION APPLICATION" document. The "AGENCY NAME AND ADDRESS" section is highlighted with a green box. The "COMPANY:" field is empty. The "UNDERWRITER:" field is empty. The "APPLICANT NAME:" field is empty. The "OFFICE PHONE:" and "MOBILE PHONE:" fields are empty. The "MAILING ADDRESS (including ZIP + 4 or Canadian Postal Code)" field is empty. The "YRS IN BUS:" field is empty. The "SIC:" field is empty. The "NAICS:" field is empty. The "WEBSITE:" field is empty. The "ADDRESS:" field is empty. The "PRODUCER NAME:" field is empty. The "CS REPRESENTATIVE NAME:" field is empty. The "OFFICE PHONE:" field is empty. The "FAX:" field is empty. The "E-MAIL ADDRESS:" field is empty. The "CODE:" and "SUB CODE:" fields are empty. The "AGENCY CUSTOMER ID:" field is empty. The "SOLE PROPRIETOR" and "CORPORATION" checkboxes are empty. The "PARTNERSHIP" and "SUBCHAPTER 'S' CORP" checkboxes are empty. The "LLC" and "TRUST" checkboxes are empty. The "JOINT VENTURE" and "OTHER" checkboxes are empty. The "CREDIT BUSINESS NAME:" field is empty. The "FEDERAL EMPLOYER ID NUMBER" field is empty. The "NCCI RISK ID NUMBER" field is empty. The "ID NUMBER:" field is empty. The "OTHER RATING BUREAU ID OR STATE EMPLOYER REGISTRATION NUMBER" field is empty.

The PDF internal structures maintain a media box. The media box is measured in points, where an inch equals 72 points.

## Example

The page size of a document with a letter layout is 8.5x11 inches. This has an equivalent media box with coordinates (0,0) at the lower left corner and coordinates (612,792) at the upper right corner (8.5 inches \* 72 = 612 points, 11 inches \* 72 = 792 points).

Application developers can print text and insert objects within the media box. The SDK library seamlessly converts the coordinates in inches in the layout to points on the media box.

## Example

The AgencyInfo field prints at the coordinate (25.2, 66.24) on the media box.

## Transformation Matrix

A PDF page can use a matrix to scale, rotate or skew elements with a transformation matrix to determine the coordinate system. If text shows up in wrong coordinates and incorrect font size, the transformation matrix is most likely in play since PDFBox library does not automatically apply the transformation when texts or objects are rendered.

## Example

A transformation matrix (0.12 0 0 0.12 0 0) means the rest of the page will be set at a scale of 12/100ths at horizontal and vertical directions. At that scale, the font sized "70" scaled comes out at 8.4 pts.

To get text at a size of 12pt, use  $12/0.12 = 100$  (scaled units). You must also position coordinates scaled at the same rate. With such a transformation matrix in play, the coordinate of the upper right corner on the media box seems to be magnified to (5100, 6600). If the AgencyInfo field is printed at the coordinates (25.2, 66.24) on the magnified media box, the texts does not show up in the correct position or font. The AgencyInfo field should show up at coordinates (210, 552) on the magnified media box ( $25.2/0.12=210$ ,  $66.24/0.12=552$ ).

How can you apply the transformation matrix? There are two options:

- Replace the ACORD PDF templates with the transformation matrix with templates that do not have the transformation. Most of the ACORD PDF templates do not have the transformation matrix
- Apply the transformation matrix via ACORD PDF layout configurations. AgencyPortal SDK (apwebapp.jar) includes a command line utility, LayoutTester, which you can use to extract the matrix.

## Example

Run the following command:

```
LayoutTester -pdf C:/temp/acord131.pdf -pdf_layout C:/temp/ACORD_131_LA YOUT.xml -extract_scale 1 -out c:/temp/out.pdf
```

The console printouts indicate that three pages of Acord131.pdf have the same transformation matrix (0.12, 0.0, 0.0, 0.12, 0.0, 0.0).

```
***Transformation matrix (cm) (Acord131.pdf) Page 1:(0.12, 0.0, 0.0, 0.12, 0.0, 0.0)
```

```
***Transformation matrix (cm) (Acord131.pdf) Page 2:(0.12, 0.0, 0.0, 0.12, 0.0, 0.0)
```

```
***Transformation matrix (cm) (Acord131.pdf) Page 3:(0.12, 0.0, 0.0, 0.12, 0.0, 0.0)
```

This requires the PDF layout configuration to be adjusted with a scale factor to account for the transformation matrix.

A new attribute for layout page elements, `scaleFactor`, is added with a default value of 1.0. For example, `scaleFactor="0.12"` for page 1, 2 and 3 for Acord131.

```
<pdf_layout name="ACORD_131_LA YOUT"> <page pageNum="1" scaleFactor="0.12"> <page pageNum="2" scaleFactor="0.12"> <page pageNum="3" scaleFactor="0.12">
```

The `scaleFactor` is automatically applied by the AgencyPortal SDK when texts are rendered. Application developers or analysts still work with the coordinates of fields in inch and regular font sizes as if there was no transformation matrix.

# PDF Generation Utility

AgencyPortal's legacy PDF utility required a significant amount of work to configure and maintain. It required developers to map native PDFs to ACORD policy data using two dimensional coordinates. This configuration also required layout remapping when there were changes to a form.

ACORD now provides eForms, which are electronic fillable ACORD forms that are capable of collecting data. To provide significant improvement to the configuration and maintenance of PDF forms, we've added a new PDF Generation utility to AgencyPortal that will utilize these eForms.

This PDF Generation utility enables delivery teams to shorten their development cycles and eliminates the possibility of needing to re-map page layouts when changes are made to the form.

## Note

It is important to note the following assumptions about this feature:

- The legacy PDF core library will remain in the SDK; however, we suggest that you use the new PDF Generation feature to generate PDF forms.
- XML based configuration structures will be created to represent how an ACORD form and its overflow schedules are mapped and built.
- If you choose to use this new PDF Generation feature, all forms, ACORD or custom, must be in the FDF format.
- The revamped PDF library utilizes the latest released open source PDFBox library, version 1.8.8. PDFBox 1.8.8 has issues supporting multiple line text fields, but the SDK tries its best to wrap around the text for multiple line text fields.
- The length of a text may exceed the maximum length of a text field on an Adobe Acrobat form. AgencyPortal will print the whole content onto the form field. Project teams can decide how to process the data (truncate or move the overflow to another place).
- All interfaces will have performance objects included.
- PDF form files are significantly larger than their native PDFs. This will cause form creation and download to be slower.

## Form Data Mapping

The Data Mapping component is where the majority of this feature's configuration takes place. It refers to the process of reviewing form eLabel and eLabelCounter data to make sure that it complies with the feature's pre-defined schema.

Each field on the form must be mapped using an XPath, TDF source or custom Java class and referenced in the corresponding pdfDefinition configuration file. Refer to the Map Form Data section in the [PDF Generation Utility Configuration](#) topic for complete information and all steps needed to configure this portion of the feature.

You will need to create a pdfDefinition for each LOB. Each pdfDefinition file and forms must then be referenced in the productDatabase file. Refer to the Add PDF Definition Files to Product Database section in the [PDF Generation Utility Configuration](#) topic for more information.

## PDF Form Processing

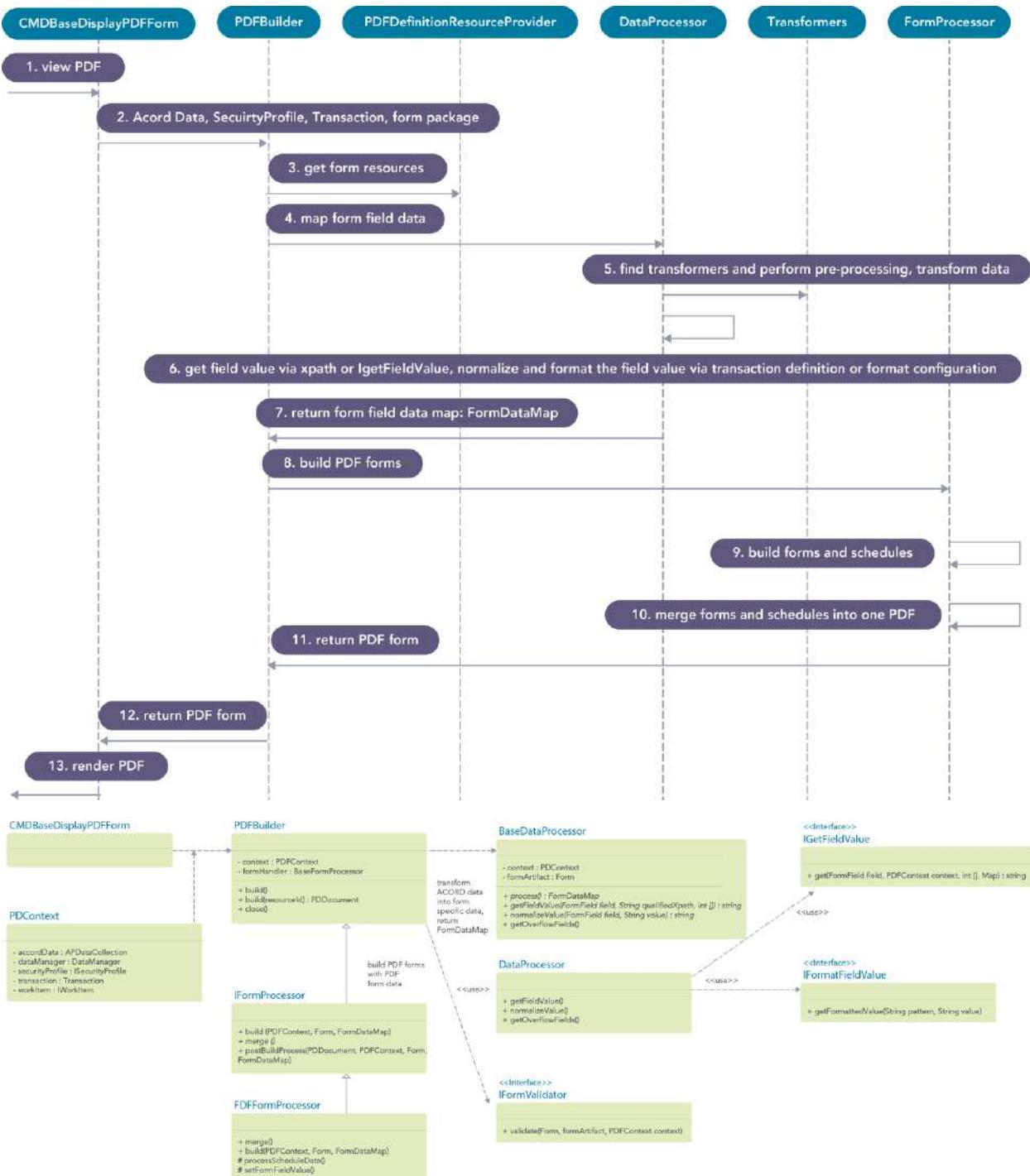
After forms are mapped and configured, ACORD policy data is transformed into PDF form specific data before the PDF generation engine can parse the form data and populate PDF forms. Each pdfDefinition file and forms you define in the ProductDatabase file are compiled during systembootup, serialized out to disk and fetched when needed.

When a PDF download request is sent via display commands, PDFBuilder takes the required context, which includes policy ACORD data, the user's security profile, the transaction definition of the policy work item, DataManager and IWorkItem. PDFBuilder then retrieves resource definitions from PDFDefinitionResourceProvider and determines which PDF forms to build.

If there are valid forms to be built, PDFBuilder hands the tasks over to DataProcessor, which executes data pre-processing via transformers and then retrieves data from ACORD forms with the XPaths defined in the pdfDefinition file for each form.

DataProcessor then hands back a FormDataMap to PDFBuilder. FormDataMap contains all of the fields and their properly formatted values.

The following diagrams illustrate this process:



After policy ACORD data is pre-processed, the form data is extracted per the configurations you determine in the definition files. XPath mapped data is extracted directly from the XPath; TDF mapped data is extracted directly from the transaction definition field.

A form's field data can also be retrieved with a Java class that implements `IGetFieldValueGetter`. Refer to the Create a Custom `IFieldValueGetter` Interface section in the [PDF Generation Utility Configuration](#) topic for more information.

## Form Population

After all field values are extracted, `PDFBuilder` instantiates and passes the form data and PDF context to `IFormProcessor`. The default form processor is `com.agencyport.pdf.FDFFormProcessor`. Project teams can use their own form processor by implementing `com.agencyport.pdf.IFormProcessor`.

`FDFFormProcessor` then goes through the transformed data, iterates the form fields, looks up the field value from the form data and sets the field value.

When `FDFFormProcessor` detects that more entries than the form can handle, the data is then printed on corresponding overflow schedules if schedules are available. For example, ACORD 127 can print as many as 13 drivers. If there are any additional drivers, they are printed on ACORD 163, which is the corresponding schedule that can hold 24 drivers.

If more than one schedule form is needed for additional entries, additional overflow schedules are created. If an overflow schedule has pagination form fields as indicated in the `pdfDefinition` configuration file, the page number is updated with every addition of overflow schedule forms.

## Form Hyperlinks

`CMDBaseDisplayPDFForms` is the base command used to download PDFs. This command can be used across LOBs. It looks for the forms you establish in a group and validates any criteria before merging them into one PDF.

A `view_acord_forms` tag on the policy summary page tells the application to generate the hyperlink that contains the PDF forms. Refer to the Add JSP Tag to Policy Summary Pages section in the [PDF Generation Utility Configuration](#) topic for information on adding this tag to summary pages.

## Resources and Artifacts

This section details the resources and artifacts included as part of this feature and ones you need to create.

### What's included?

The following resources are included in the SDK as part of this feature.

#### Schemas

<code>pdfDefinitions</code>	This is the <a href="#">pre-determined schema</a> that is used to validate the data mappings added to the <code>pdfDefinition</code> files that you create. All <code>eLabel</code> and <code>eLabelCount</code> mappings in each form field must match what is outlined in this file.
-----------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

#### Java Classes

<code>BaseDataProcessor</code>	Handles the transformation of ACORD data into form field data based on the FDF definitions.
<code>CMDBaseDisplayPDFForm</code>	Base command to download PDFs. This command can be used across LOBs. It expects a PACKAGE parameter. All form groups with a form group name of PACKAGE within the validation criteria are created and merged into one PDF form. If the PACKAGE parameter is missing, all forms for the LOB are created within the validation criteria.
<code>DataProcessor</code>	Parses ACORD XML data into form data. It executes data pre-processing via transformers and then retrieves data from ACORD data using the XPaths defined in the <code>pdfDefinition</code> file for each form. It then hands back a <code>FormDataMap</code> to <code>PDFBuilder</code> , which contains all the fields and their properly formatted values.
<code>FDFFormProcessor</code>	Handles form merging and data population for ACORD FDF forms.
<code>FormDataMap</code>	Contains all the fields and their properly formatted values.

FormValidator	Default mechanism used to validate forms.
OverflowSchedule	Holds the form overflow artifacts.
PDFBuilder	Handles the PDF generation from work item ACORD data.
PDFContext	Context information, including transactions, ACORD data and security profile.
PDFDefinitionParser	Parses PDF XML definitions that has embedded <include> markup.
PDFDefinitionRepository	The repository for PDF form definitions
PDFDefinitionResourceProvider	Reads PDF definition resources.
PDFFormResourceProvider	Serves up PDF instances. The names of the PDF are assumed to be the same as the artifact's title.
PDFFormUtility	Provides help for the population of PDF forms.

#### Interfaces

IFieldValueFormatter	Interface used for custom field formatting.
IFormProcessor	Interface used for custom population of PDF forms.
IFormValidator	Interface used for custom form validation.
IFieldValueGetter	Interface used to get field data using a customJava class.

#### What's needed?

##### Forms

You must have valid eForms to set up this feature. These can be ACORD forms downloaded from ACORD or customforms you create using Adobe Acrobat Pro (refer to the [Creating eForms Developer Guide](#) document for more information on creating your own customeForms).

ACORD eForms come in two formats:

- Forms Data Format (FDF) – This format is based on the PDF format and provides the most flexibility. It is used to submit form data to a server, receive a response and incorporate it into an interactive form. It can also be used to export form data that can be transmitted electronically and then imported back into the corresponding PDF interactive form. This format also supports the ability to merge multiple PDFs into one PDF document and individual field customization.
- XML Forms Architect (XFA) – This format is basically a “wrapper” around the XML form that allows it to display as a PDF. It requires a form template, form data and configuration information to describe how the form should appear and behave. XFA forms cannot be merged because they are not really PDFs.

AgencyPortal only supports the use of FDF formatted PDFs since they provide the most flexibility and support the ability to merge multiple PDFs into one document. It is important that you make sure that the ACORD forms and/or custom forms that you use are in this format.

##### Configuration Files

For this feature, you must create the following:

pdfDefinition.xml files	A pdfDefinition file is needed for each LOB (e.g., commlAutoPdfDefinition is for Commercial Auto). These files contain the meat of this feature. They are used to map your PDF forms and are validated against the pdfDefinitions schema. This file must also be referenced in the ProductDatabase.xml file.
-------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

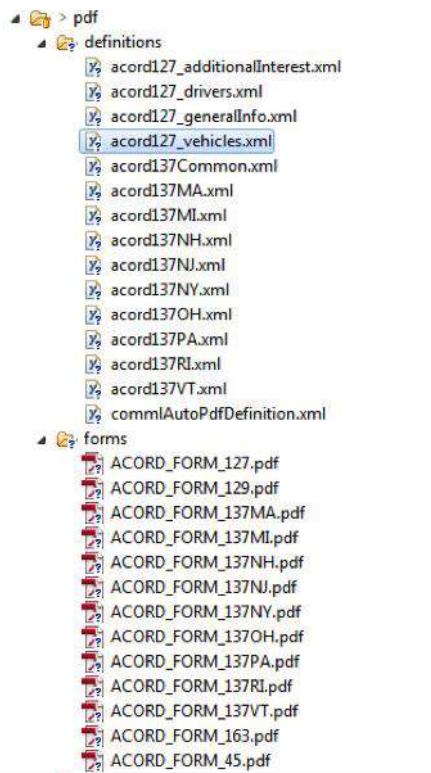
	Refer to the Create PDF Definition Files section in the <a href="#">PDF Generation Utility Configuration</a> topic for more information.
Resource files	Resource files (e.g., acord_generalInfo.xml or accord_formHeader.xml) are XML mapping files you create for common fields. These files can be reused for various forms if they reference the same fields. Some forms may include one or multiple resource files, depending on the number of common fields on the form. Refer to the When to Create Resource Files section in the <a href="#">PDF Generation Utility Configuration</a> topic for more information.

You must structure and organize your definition, resource and PDF files under the pdf directory within the product.

- All resource and definition files must be in a directory called definitions
- All PDF eForms must be in a directory called forms

The location of your directory should coincide with where you define your pdfDefinition and pdfForm resources in the ProductDatabase.xml file (i.e., as a shared resource or under an LOB):

The following is an example:



## Custom Java Classes

Depending on the level of customization you want to do for this feature, you may also need to create the following:

Custom Java classes	If you don't want to map fields using XPaths or TDF sources, you can create your own custom Java class that points to a custom <code>IFieldValueGetter</code> interface. Refer to Create a Custom <code>IFieldValueGetter</code> Interface for more information.
---------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

# PDF Generation Utility Configuration

The following steps are needed to implement this feature:

1. **Establish all forms needed for each LOB** – It is at the discretion of each project team to determine which forms are needed for each line of business (LOB). Forms can include ACORD forms or custom forms.
2. **Retrieve and review eForms** – AgencyPortal supports the use of ACORD or custom Adobe Acrobat Pro eForms in the FDF (Forms Data Format) format. The fields on each form must match what is defined in the PDF Generation schema (pdfDefinitions.xsd). Refer to the Retrieve and Review eForms section below for more information on this process.
3. **Add PDF Definition files and form references to the product database** – Each pdfDefinition file and PDF form must be referenced in the ProductDatabase.xml file in order for AgencyPortal to process the XML data mapping artifacts during application bootstrap time. Refer to the Add PDF Definition Files to Product Databases section below for more information.
4. **Create a PDF Definition file** – The pdfDefinition file servers as the main configuration component of this feature. This file houses all form mapping references for each LOB. Refer to the Create PDF Definition File(s) section below for more information.
5. **Map form fields** – All form fields must be mapped using XPaths, TDF or a custom Java class. Each form may have one or several mapping files (called resource files) depending on whether the fields are unique or common among other forms. Each resource file created must also be added to the PDF Definition file. Refer to the Map Form Fields section below for more information.
6. **Create any custom IFieldValueGetter interfaces** – If using custom Java classes to map field data, a custom IFieldValueGetter interface is needed to retrieve the data values. Refer to the Create a Custom IFieldValueGetter Interface section below for more information.
7. **Add any form variables** – Form variables allow you to identify XPath qualifiers if multiple forms use common fields. This is a helpful component to use for state forms. Refer to the Add Form Variables section below for more information.
8. **Add any form validation** – Form validation allows you to define certain criteria to only generate forms under a certain condition. This allows you to identify whether to only generate a form based on a transaction type or when a field equals a certain value. Refer to the Add Form Validation section below for more information.
9. **Add any schedules and overflows** – A schedule is used to source repeating roster information. An overflow is used to print additional items that do not fit on the form. Refer to the Add Schedules and Overflows section below for more information.
10. **Determine how you want forms to be grouped** – Forms print based on the group in which you place them; otherwise, all forms print on one PDF. This configuration is included in the pdfDefinition file. Refer to the Establish Form Groups section below for more information.
11. **Add the view\_acord\_forms tag to policy summary pages** – The view\_acord\_forms tag must be added to policy summary. This tag determines whether the form hyperlink displays on the policy summary page for a user to view the forms. Refer to the Add JSP Tag to Policy Summary Pages section below for more information.

## Establish All Forms Needed for Each LOB

It is at the discretion of each project team to determine which forms are needed for each line of business (LOB). You can include ACORD eForms or custom eForms created using Adobe Acrobat Pro (refer to the [Creating eForms Developer Guide](#) document for more information on this step).

To download ACORD eForms, you must have login credentials. Refer to the [ACORD website](#) for more information.

## Retrieve and Review eForms

You must review all of the forms you want to include for an LOB. This includes downloading the ACORD eForms and corresponding mapping spreadsheets from the ACORD website. You will need Adobe Acrobat Pro to view each field eLabel and their corresponding properties.

## Add PDF Definition Files to Product Database

All forms and form mapping configuration files (pdfDefinition files) must be validated against the PDF Generation schema (pdfDefinitions.xsd) and referenced in the productDatabase.xml file.

- **pdfDefinition** – Use this artifact type to reference a pdfDefinition file. You must reference the pdfDefinition file under the corresponding LOB.

### Example

```
<product type="AUTOB" version="5.0.0.0" title="Commercial Auto" path="commAuto" description="Commercial Auto" id="N167"> <artifact type="pdfDefinition" resourceId="commAutoPdfDefinition.xml" title="PDF definition" description="PDF definitions" id="N287" /> </product>
```

- **pdfForm** – Use this artifact type to reference PDF forms that will be loaded and shared as a resource. Forms can be referenced in the productDatabase.xml file as one of the following:
  - As a shared resource (e.g., a customform that is shared across all LOBs).

## Example

```
<product type="shared" shared="true" version="5.0.0.0" path="shared" title="Shared Product" id="N86"> <artifact type="pdfForm" resourceId="AP_WORKITEM_ASSISTANT_COMMENTS.pdf" title="AP_WORKITEM_ASSISTANT_COMMENTS" description="AP work item assistant comments PDF" id="N13" /> </product>
```

- Grouped under a corresponding LOB.

## Example

```
<product type="AUTOB" version="5.0.0.0" title="Commercial Auto" path="commAuto" description="Commercial Auto" id="N167"> <artifact type="pdfForm" resourceId="ACORD_FORM_127.pdf" title="ACORD_127_FORM" description="Acord 127 PDF" id="N288" /> <artifact type="pdfForm" resourceId="ACORD_FORM_129.pdf" title="ACORD_129_FORM" description="Acord 129 PDF" id="N289" /> <artifact type="pdfForm" resourceId="ACORD_FORM_163.pdf" title="ACORD_163_FORM" description="Acord 163 PDF" id="N290" /> </product>
```

The productDatabase.xml file is launched during application bootstrap time. The above resource providers process the XML artifacts registered in this file. It is important that you make sure all pdfDefinition files and PDF forms used in your mapping configurations are referenced in this file.

## Map Form Data

Each form must be mapped in a corresponding pdfDefinition file. The following tasks must be completed to map each form and each section includes various scenarios that can exist:

1. Create corresponding pdfDefinition files. Each LOB must have its own pdfDefinition file with all corresponding forms configured and mapped.
2. Map each form field. Fields can be mapped using XPaths, TDF sources or via a customJava class.
3. Configure any schedules and/or overflows. Schedules are form attachments that print additional roster item data; overflows are additional forms (usually custom) that print additional data that does not print on the ACORD form or schedule. A form can include both a schedule and an overflow. Various scenarios can exist for overflow data. Be sure to review your forms thoroughly to account for overflow situations that may arise. It is up to your project teams whether an overflow form is needed to print overflow data.
4. Add any form variables.
5. Add any form validations.

## Create PDF Definition File(s)

The pdfDefinition file serves as the main configuration component of this feature. Each LOB must have a corresponding pdfDefinition file (e.g., commAutoPdfDefinition file can be used for the Commercial Auto LOB) if you intend to generate PDF forms using this feature. This file allows you to configure the following:

- Forms to include
- Field mappings (either directly in the file or via a linked resource file)
- Form validation
- Form variables
- Schedules and overflows
- Form grouping

Each pdfDefinition file can contain multiple forms for an LOB. For example, Commercial Auto can include the following forms:

- ACORD 125
- ACORD 127
- ACORD 137
- A custom Work Item Assistant form

The following is an example of a pdfDefinition file for Workers Comp, which includes ACORD 130 and a customWork Item Assistant form:

```
<forms lob="WORK" xmlns:xi="http://www.w3.org/2001/XMLSchema-instance" xmlns:namespaceSchemaLocation="http://reference.agencyport.com/schemas/5.1/pdf/pdfDefinitions.xsd"> <formGroups> <group name="ACORD130" title="label.Document.AcordForm130"> <formResource>ACORD130</formResource> </group> </formGroups> <form name="ACORD130" resourceId="ACORD-0130-2013-09r2.pdf"> <include resourceId="../../commercial/pdf/definitions/acord_formHeader.xml" /> <include resourceId="../../commercial/pdf/definitions/acord125_producerNameInsured.xml" /> <fields> <field> <eLabel>NamedInsured_InBusiness YearCount_A</eLabel> <df tdfFieldId="yrsInBusiness" /></field> <field type="checkbox" javaclass="com.agencyport.workerscomp.pdf.eform.OtherLegalEntity Value Getter">
```

```

<eLabel>NamedInsured_LegalEntity_OtherIndicator_A</eLabel> <tdf tdfFieldId="legalEntity" /> </field> <field
javaclass="com.agencyport.workerscomp.pdf.form.OtherLegalEntity Value Getter">
<eLabel>NamedInsured_LegalEntity_OtherDescription_A</eLabel> <tdf tdfFieldId="legalEntity" /> </field> <field>
<eLabel>Producer_CustomerServiceRepresentative_FullName_A</eLabel> <tdf tdfFieldId="CSRName" /> </field> <field>
<eLabel>NamedInsured_CreditBureauName_A</eLabel> <tdf tdfFieldId="creditBureauName" /> </field> <field>
<eLabel>NamedInsured_CreditBureauIdentifier_A</eLabel> <tdf tdfFieldId="creditBureauID" /> </field> <field>
<eLabel>NamedInsured_NCCIriskIdentifier_A</eLabel> <tdf tdfFieldId="NCCIid" /> </field> <field>
<eLabel>NamedInsured_RatingBureauIdentifier_A</eLabel> <tdf tdfFieldId="stateEmployerNo" /> </field> <field>
<eLabel>Policy_NormalAnniversaryRatingDate_A</eLabel> <format type="date" /> <tdf tdfFieldId="anniversaryRatingDate" /> </field>
<field type="checkbox" yes Value="1"> <eLabel>Policy_ParticipatingIndicator_A</eLabel> <tdf tdfFieldId="participatingPlanInd" /> </field>
<field type="checkbox" yes Value="0"> <eLabel>Policy_NonParticipatingIndicator_A</eLabel> <tdf tdfFieldId="participatingPlanInd" />
</field> <field> <eLabel>Policy_RetrospectiveRatingPlan_A</eLabel> <tdf tdfFieldId="retrospectiveRatingPlanCd" /> </field> <field>
<eLabel>WorkersCompensationEmployersLiability_EmployersLiability_EachAccidentLimitAmount_A</eLabel>
<xpath>WorkCompLineBusiness.CommlCoverage[CoverageCd='WCEL'].Limit[LimitAppliesToCd='PerAcc'].FormatInteger</xpath> <format
type="number" pattern="#,###,###.##" /></field> <field>
<eLabel>WorkersCompensationEmployersLiability_EmployersLiability_DiseasePolicyLimitAmount_A</eLabel>
<xpath>WorkCompLineBusiness.CommlCoverage[CoverageCd='WCEL'].Limit[LimitAppliesToCd='DisPol'].FormatInteger</xpath> <format
type="number" pattern="#,###,###.##" /></field> <field>
<eLabel>WorkersCompensationEmployersLiability_EmployersLiability_DiseaseEachEmployeeLimitAmount_A</eLabel>
<xpath>WorkCompLineBusiness.CommlCoverage[CoverageCd='WCEL'].Limit[LimitAppliesToCd='Dis EachEmpl'].FormatInteger</xpath>
<format type="number" pattern="#,###,###.##" /></field> <field type="checkbox">
<eLabel>WorkersCompensation_DeductibleType_MedicalIndicator_A</eLabel>
<xpath>WorkCompLineBusiness.CommlCoverage[CoverageCd='WCEL'].Deductible[DeductibleTypeCd='MED'].FormatInteger</xpath>
</field> <field type="checkbox"> <eLabel>WorkersCompensation_DeductibleType_IndemnityIndicator_A</eLabel>
<xpath>WorkCompLineBusiness.CommlCoverage[CoverageCd='WCEL'].Deductible[DeductibleTypeCd='IND'].FormatInteger</xpath>
</field> <field type="checkbox"> <eLabel>WorkersCompensation_DeductibleType_OtherIndicator_A</eLabel>
<xpath>WorkCompLineBusiness.CommlCoverage[CoverageCd='WCEL'].Deductible[DeductibleTypeCd='MEDIND'].FormatInteger</xpath>
</field> <field type="checkbox"> <eLabel>WorkersCompensation_DeductibleType_OtherIndicator_A</eLabel>
<xpath>WorkCompLineBusiness.CommlCoverage[CoverageCd='WCEL'].Deductible[DeductibleTypeCd='OT'].FormatInteger</xpath> </field>
<field> <eLabel>WorkersCompensation_DeductibleType_OtherDescription_A</eLabel>
<xpath>WorkCompLineBusiness.CommlCoverage[CoverageCd='WCEL'].Deductible[DeductibleTypeCd='MEDIND'].DeductibleTypeCd</xpath>
<tdf tdfFieldId="deductibleTypeCd" /> </field> <field> <eLabel>WorkersCompensation_DeductibleType_OtherDescription_A</eLabel>
<xpath>WorkCompLineBusiness.CommlCoverage[CoverageCd='WCEL'].Deductible[DeductibleTypeCd='OT'].DeductibleTypeCd</xpath>
<tdf tdfFieldId="deductibleTypeCd" /> </field> <field> <eLabel>WorkersCompensation_DeductibleAmount_A</eLabel> <format
type="number" pattern="#,###,###.##" /> <tdf tdfFieldId="deductible" /> </field> <field type="checkbox">
<eLabel>WorkersCompensation_Coverage_USLHIndicator_A</eLabel> <tdf tdfFieldId="policyUSLH" /> </field> <field type="checkbox">
<eLabel>WorkersCompensation_Coverage_VoluntaryCompensationIndicator_A</eLabel> <tdf
tdfFieldId="WorkCompLineBusiness.CommlCoverage.VOL" /> </field> <field type="checkbox">
<eLabel>WorkersCompensation_Coverage_ForeignCoverageIndicator_A</eLabel> <tdf tdfFieldId="policyForeign" /> </field> <field
type="checkbox"> <eLabel>WorkersCompensation_Coverage_ManagedCareOptionIndicator_A</eLabel> <tdf
tdfFieldId="policyManagedCare" /> </field> <field> <eLabel>WorkersCompensation_DividendOrSafetyPlan_A</eLabel> <tdf
tdfFieldId="dividendPlan" /> </field> <field> <eLabel>WorkersCompensation_AdditionalCompanyInformation_A</eLabel> <tdf
tdfFieldId="additionalCompanyInfo" /> </field> <field javaclass="com.agencyport.shared.pdf.form.FullNameInfoGetter">
<eLabel>NamedInsured_AccountingContact_FullName_A</eLabel>
<xpath>CommlPolicy.MiscParty[MiscPartyRoleCd='AC'].GeneralPartyInfo.NameInfo</xpath> </field> <field>
<eLabel>NamedInsured_AccountingContact_PhoneNumber_A</eLabel> <tdf tdfFieldId="ACPhone" /> </field> <field>
<eLabel>NamedInsured_AccountingContact_CellPhoneNumber_A</eLabel> <tdf tdfFieldId="ACMobile" /> </field> <field>
<eLabel>NamedInsured_AccountingContact_EmailAddress_A</eLabel> <tdf tdfFieldId="ACEmail" /> </field> <field>
javaclass="com.agencyport.shared.pdf.form.FullNameInfoGetter"> <eLabel>NamedInsured_InspectionContact_FullName_A</eLabel>
<xpath>CommlPolicy.MiscParty[MiscPartyRoleCd='AC'].GeneralPartyInfo.NameInfo</xpath> </field> <field>
<eLabel>NamedInsured_InspectionContact_PhoneNumber_A</eLabel> <tdf tdfFieldId="ICPhone" /> </field> <field>
<eLabel>NamedInsured_InspectionContact_CellPhoneNumber_A</eLabel> <tdf tdfFieldId="ICMobile" /> </field> <field>
<eLabel>NamedInsured_InspectionContact_EmailAddress_A</eLabel> <tdf tdfFieldId="ICEmail" /> </field> <field>
javaclass="com.agencyport.shared.pdf.form.FullNameInfoGetter"> <eLabel>NamedInsured_ClaimContact_FullName_A</eLabel>
<xpath>CommlPolicy.MiscParty[MiscPartyRoleCd='CC'].GeneralPartyInfo.NameInfo</xpath> </field> <field>
<eLabel>NamedInsured_ClaimContact_PhoneNumber_A</eLabel> <tdf tdfFieldId="CCPhone" /> </field> <field>
<eLabel>NamedInsured_ClaimContact_CellPhoneNumber_A</eLabel> <tdf tdfFieldId="CCMobile" /> </field> <field>
<eLabel>NamedInsured_ClaimContact_EmailAddress_A</eLabel> <tdf tdfFieldId="CCEmail" /> </field> <field>
<eLabel>CommercialPolicy_OperationsDescription_A</eLabel> <tdf tdfFieldId="operationsDesc" /> </field> <field type="question">
<eLabel>WorkersCompensationLineOfBusiness_Question_ABCode_A</eLabel>
<xpath>WorkCompLineBusiness.QuestionAnswer[QuestionCd='WORK07'].YesNoCd</xpath> </field> <field>
<eLabel>WorkersCompensationLineOfBusiness_ApplicantOwnLeaseAircraftOrWatercraftExplanation_A</eLabel>
<xpath>WorkCompLineBusiness.QuestionAnswer[QuestionCd='WORK07'].Explanation</xpath> </field> <field type="question">
<eLabel>WorkersCompensationLineOfBusiness_Question_ACDCode_A</eLabel>
<xpath>WorkCompLineBusiness.QuestionAnswer[QuestionCd='WORK16'].YesNoCd</xpath> </field> <field>
<eLabel>WorkersCompensationLineOfBusiness_PastPresentDiscontinuedOperationsHazardousMaterialExplanation_A</eLabel>
<xpath>WorkCompLineBusiness.QuestionAnswer[QuestionCd='WORK16'].Explanation</xpath> </field> <field type="question">
<eLabel>WorkersCompensationLineOfBusiness_Question_AADCode_A</eLabel>
<xpath>WorkCompLineBusiness.QuestionAnswer[QuestionCd='BOP21'].YesNoCd</xpath> </field> <field>

```

<eLabel>Workers CompensationLineOfBusiness\_WorkPerformedUndergroundOrAboveFifteenFeetExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='BOP21'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_KARCode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='CUMBR09'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_WorkPerformedOnVesselsDocksBridgesExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='CUMBR09'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_KASCode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='BOP11'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_ApplicantEngagedAnyOtherTypeOfBusinessExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='BOP11'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_KATCode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='WORK06'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_SubcontractorsUsedExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='WORK06'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_KAUCode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='BOP15'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_AnyWorkSubletWithoutCertificatesOfInsuranceExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='BOP15'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_ABCCode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='GENRL47'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_WrittenSafetyProgramInOperationExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='GENRL47'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_ABICode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='WORK11'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_AnyGroupTransportationProvidedExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='WORK11'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_AAECode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='BOP22'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_AnyEmployeesUnderSixteenOrOverSixtyYearsExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='BOP22'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_KAVCode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='WORK13'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_AnySeasonalEmployeesExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='WORK13'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_AAFCode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='BOP23'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_VolunteerOrDonatedLabourExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='BOP23'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_ABJCode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='WORK12'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_EmployeesWithPhysicalHandicapsExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='WORK12'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_ABHCode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='WORK10'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_EmployeesTravelOutOfStateExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='WORK10'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_AABCode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='BOP10'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_AthleticTeamsSponsoredExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='BOP10'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_ACBCode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='WORK14'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_PhysicalsRequiredAfterOffersOfEmploymentAreMadeExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='WORK14'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_ABACode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='GENRL22'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_OtherInsuranceWithThisInsurerExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='GENRL22'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_AAICode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='GENRL06'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_PriorCoverageDeclinedCancelledNonRenewedLastThreeYearsExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='GENRL06'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_ABFCODE\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='WORK08'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_EmployeeHealthPlansProvidedExplanation\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='WORK08'].Explanation</xpath> </field> <field type="question">  
 <eLabel>Workers CompensationLineOfBusiness\_Question\_KAWCode\_A</eLabel>  
 <xpath>WorkCompLine Business.QuestionAnswer[QuestionCd='CGL05'].YesNoCd</xpath> </field> <field>  
 <eLabel>Workers CompensationLineOfBusiness\_AnyEmployeesPerformWorkForOtherBusinessesExplanation\_A</eLabel>

```

<xpath>WorkCompLineBusiness.QuestionAnswer[QuestionCd='CGL05'].Explanation</xpath> </field> <field type="question">
<eLabel>WorkersCompensationLineOfBusiness_Question_AACCode_A</eLabel>
<xpath>WorkCompLineBusiness.QuestionAnswer[QuestionCd='CGL04'].YesNoCd</xpath> </field> <field>
<eLabel>WorkersCompensationLineOfBusiness_LeaseEmployeesToOrFromOtherEmployersExplanation_A</eLabel>
<xpath>WorkCompLineBusiness.QuestionAnswer[QuestionCd='CGL04'].Explanation</xpath> </field> <field type="question">
<eLabel>WorkersCompensationLineOfBusiness_Question_ABGCode_A</eLabel>
<xpath>WorkCompLineBusiness.QuestionAnswer[QuestionCd='WORK09'].YesNoCd</xpath> </field> <field>
<eLabel>WorkersCompensationLineOfBusiness_EmployeesPredominantlyWorkAtHomeExplanation_A</eLabel>
<xpath>WorkCompLineBusiness.QuestionAnswer[QuestionCd='WORK09'].Explanation</xpath> </field> <field type="question">
<eLabel>WorkersCompensationLineOfBusiness_Question_KAXCode_A</eLabel>
<xpath>WorkCompLineBusiness.QuestionAnswer[QuestionCd='GENRL14'].YesNoCd</xpath> </field> <field>
<eLabel>WorkersCompensationLineOfBusiness_TaxLiensOrBankruptcyWithinLastFiveYearsExplanation_W</eLabel>
<xpath>WorkCompLineBusiness.QuestionAnswer[QuestionCd='GENRL14'].Explanation</xpath> </field> <field type="question">
<eLabel>WorkersCompensationLineOfBusiness_Question_KAYCode_A</eLabel>
<xpath>WorkCompLineBusiness.QuestionAnswer[QuestionCd='WORK17'].YesNoCd</xpath> </field> <field>
<eLabel>WorkersCompensationLineOfBusiness_UndisputedUnpaidWorkersCompensationPremiumDueExplanation_A</eLabel>
<xpath>WorkCompLineBusiness.QuestionAnswer[QuestionCd='WORK17'].Explanation</xpath> </field> <fields> <schedule name="location"> <repeat size="3" source="Location"> <include resourceId="locations.xml" /> </repeat> </schedule> <schedule name="policyStates"> <repeat size="10" source="WorkCompLineBusiness.WorkCompRateState"> <fields> <field>
<eLabel>WorkersCompensation_PartOne_StateOrProvinceCode</eLabel>
<xpath>WorkCompLineBusiness.WorkCompRateState.StateProvCd</xpath> </field> </fields> </repeat> </schedule> <schedule name="otherStates"> <repeat size="10" source="WorkCompLineBusiness.OtherCoveredStateProvCd"> <fields> <field>
<eLabel>WorkersCompensation_PartThree_StateOrProvinceCode</eLabel>
<xpath>WorkCompLineBusiness.OtherCoveredStateProvCd</xpath> </field> </fields> </repeat> </schedule> <schedule name="includedExcludedIndividuals"> <repeat size="4" source="WorkCompLineBusiness.WorkCompIndividuals"> <fields> <field indexRef="WorkCompLineBusiness.WorkCompIndividuals.@LocationRef">
<eLabel>WorkersCompensation_Individual_StateOrProvinceCode</eLabel> <xpath>Location[@id='${indexRef}'].Addr.StateProvCd</xpath>
</field> <field indexRef="WorkCompLineBusiness.WorkCompIndividuals.@LocationRef" />
<eLabel>WorkersCompensation_Individual_LocationProducerIdentifier</eLabel>
<xpath>Location[@id='${indexRef}'].ItmldInfo.AgencyId</xpath> </field> <field javaclass="com.agencyport.shared.pdf.eform.FullNameInfoGetter"> <eLabel>WorkersCompensation_Individual_FullName</eLabel>
<xpath>WorkCompLineBusiness.WorkCompIndividuals.NameInfo</xpath> </field> <field>
<eLabel>WorkersCompensation_Individual_BirthDate</eLabel> <format type="date" /> <tdf tdfFieldId="WorkCompIndividualsDOB" />
</field> <field> <eLabel>WorkersCompensation_Individual_TitleRelationshipCode</eLabel> <tdf tdfFieldId="WorkCompIndividualsTitle" />
</field> <field> <eLabel>WorkersCompensation_Individual_OwnershipPercent</eLabel> <tdf tdfFieldId="WorkCompIndividualsOwnershipPercent" />
</field> <field>
<eLabel>WorkersCompensation_Individual_DutiesDescription</eLabel> <tdf tdfFieldId="WorkCompIndividualsDuties" /> </field> <field>
<eLabel>WorkersCompensation_Individual_IncludedExcludedCode</eLabel> <tdf tdfFieldId="WorkCompIndividualsIncExcCd" /> </field>
<field> <eLabel>WorkersCompensation_Individual_RatingClassificationCode</eLabel>
<xpath>WorkCompLineBusiness.WorkCompIndividuals.RatingClassificationCd</xpath> </field> <field>
<eLabel>WorkersCompensation_Individual_RemunerationAmount</eLabel> <format type="number" pattern="####,###,###" /> <tdf tdfFieldId="WorkCompIndividualsRemuneration" /> </field> </fields> </repeat> </schedule> <schedule name="stateRating" source="WorkCompLineBusiness.WorkCompRateState" overflow="stateRating"> <fields> <field>
<eLabel>WorkersCompensation_RateState_StateOrProvinceName_A</eLabel>
<xpath>WorkCompLineBusiness.WorkCompRateState.StateProvCd</xpath> </field> <field>
<eLabel>WorkersCompensationStateCoverage_ExperienceOrMerit_ModificationFactor_A</eLabel>
<xpath>WorkCompLineBusiness.WorkCompRateState.CreditOrSurcharge[CreditSurchargeCd='AREM'].NumericValue.FormatModFactor</xpath> </field> </fields> <repeat size="14" source="WorkCompLineBusiness.WorkCompRateState.WorkCompLocInfo.WorkCompRateClass" resourceId="ACORD-0130-2013-09r2-page2.pdf"> <include resourceId="stateRating.xml" /> </repeat> </schedule> <overflow name="stateRating" size="14" resourceId="ACORD-0130-2013-09r2-page2.pdf"> <pagination pageLabel="WorkersCompensation_RateState_PageNumber_A" totalPageLabel="WorkersCompensation_RateState_TotalPageNumber_A" /> </overflow> <schedule name="priorcarrier"> <repeat size="3" source="WorkCompLineBusiness.WorkCompLossOrPriorPolicy" /> <fields> <field> <eLabel>PriorCoverage_EffectiveYear</eLabel> <xpath>WorkCompLineBusiness.WorkCompLossOrPriorPolicy.EffectiveDt</xpath> <format type="date" pattern="yyyy" /> </field> <field> <eLabel>PriorCoverage_InsurerFullName</eLabel> <tdf tdfFieldId="LossCarrier" /> </field> <field> <eLabel>PriorCoverage_PolicyNumberIdentifier</eLabel> <tdf tdfFieldId="LossPolicyNumber" /> </field> <field>
<eLabel>PriorCoverage_TotalPremiumAmount</eLabel> <format type="number" pattern="####,###,###" /> <tdf tdfFieldId="LossAnnualPremium" /> </field> <field> <eLabel>PriorCoverage_ModificationFactor</eLabel> <tdf tdfFieldId="LossExpMod" /> </field> <field> <eLabel>LossHistory_ClaimCount</eLabel> <tdf tdfFieldId="LossNoOfClaims" /> </field> <field>
<eLabel>LossHistory_PaidAmount</eLabel> <format type="number" pattern="####,###,###" /> <tdf tdfFieldId="LossAmountPaid" /> </field> <field> <eLabel>LossHistory_ReserveAmount</eLabel> <format type="number" pattern="####,###,###" /> <tdf tdfFieldId="LossAmountReserved" /> </field> </fields> </repeat> </schedule> </form> <form name="workItemAssistantData" resourceId="AP_WORKITEM_ASSISTANT_COMMENTS.pdf" > <include resourceId="../../commercial/pdf/definitions/acord_formHeader.xml" /> <include resourceId="../../shared/pdf/definitions/workItemAssistantComments.xml" /> </form> </forms>

```

The following are the various elements that can be used to configure the pdfDefinition file:

resourceId	The resource file to use. This includes the form file, which you designated as a <code>pdfForm</code> in the <code>ProductDefinition.xml</code> file and any resource files you create to map the fields on the form.
javaclass	The custom Java class, which implements <code>BaseDataProcessor</code> . This class is used for the custom transformation of ACORD XML data into form specific data. The default is <code>com.agencyport.pdf.DataProcessor</code> .
transformer	The transformer is used as pre-processing facilities. The transformer will be executed at the very first step during the data transformation. The step allows delivery teams to transform the data into a certain format before the data is transformed into a form specific format.
variables	A list of variables with name-value pairs. These values will replace the matching variables in the XPaths of form fields.
fields	Represents an aggregate of form fields.
schedule	The form contains repeated fields, such as Driver. If the number of entries exceeds the form capacity, the rest of the entries may be moved to schedule attachments. Non-repeat entries can also be marked as overflow and moved to schedule attachments.
overflow	The ACORD attached schedules, such as ACORD 163.
criteria	Forms are built only under certain conditions. This element provides an opportunity to specify the transaction types, XPath or customJava class to validate the form before it is built.

## Map Form Fields

Review and take inventory of the fields on each form. Are there any common or repeat fields that can be used across multiple forms or are they unique to that form?

### Ways to Map Form Fields

The field artifact is the basic form element configuration that specifies how the field value is retrieved and formatted. The following are the various elements that can be used to map fields:

type	Designates the type of the field: check box or question. The default is text.
tdf	Designates the TDF field per transaction, which is the data source for the eLabel.
indexRef	<p>Indicates that the data source is out of overflow data source (i.e., in a different XPath, but references the overflow XPath).</p> <p><b>Example</b></p> <pre>&lt;field indexRef="CommAutoLineBusiness.CommlRateState.CommlVeh.@LocationRef"&gt; &lt;eLabel&gt;Vehicle_PhysicalAddress_LineOne&lt;/eLabel&gt; &lt;xpath&gt;Location[@id='\${indexRef}'].Addr.Addr1&lt;/xpath&gt; &lt;/field&gt;</pre> <p>The <code>locationRef</code> is used to find the vehicle location.</p>
javaclass	Specifies the Java class to get the field value. The Java class must implement <code>com.agencyport.pdf.IFieldValueGetter</code> .
eLabel	The eLabel of the form field.
xpath	The XPath for the field value.

format	Designates the formatting mechanism for data printing. The type can be date, number or a custom Java class, which implements <code>com.agencyport.pdf.IFieldValueFormatter</code> .  <b>Example</b>  <code>&lt;field&gt; &lt;eLabel&gt;LossHistory_ReservedAmount&lt;/eLabel&gt; &lt;format type="number" pattern="#,##,##" /&gt;&lt;tdf tdfFieldId="LossAmountReserved" /&gt; &lt;/field&gt;</code>
multiValue	Indicates that the field holds multiple values for multiple PDF form fields.
eLabelCounters	The eLabel counter for the field.

Field data can be retrieved using XPaths from ACORD XML or via a custom Java class. You can also use TDF sources to map the form field to the transaction definition file.

## Best Practice

As a best practice, we recommend that you map fields using TDF sources whenever possible. This will make mapping easier since XPaths can be longer and more redundant. Note, however, there are a few scenarios when using TDF sources are not the best option, such as:

- Using an indexRef to cross link data
- When the field data requires reverse lookup, such as the form only fits the state abbreviation but the TDF value is the state name

## Note

Any text field can include overflow text if the data is longer than the field on the form. By default, this information will not display on the printed form. However, you can view the data in the electronic version of the form by selecting the field and scrolling through the field data. It is up to project teams to determine how they want to handle this data (i.e., truncate it or cut it off at the overflow).

### XPath

XPath fields provide direct mapping to ACORD XML data. However, they can be lengthy and repetitive. We recommend you only use XPath mapping when TDF mapping is not available. XPath mapping is configured using the `<xpath>` attribute.

## Example

```
<field> <eLabel>Driver_ProducerIdentifier</eLabel> <xpath>CommAutoLineBusiness.CommlDriver.ItemInfo.AgencyId</xpath> </field>
```

### TDF Source

Mapping fields using TDF sources means that any views, code list lookups and format masks will be applied as defined by the transaction definition field. XPaths are also automatically inherited. A TDF field must have a unique ID to use it as a source (referenced in the pdfDefinition file as `<tdfFieldId>`). If the field does not have a unique ID, you must map the field using the XPath.

## Example

```
<field> <eLabel>WorkersCompensation_AdditionalCompanyInformation_A</eLabel> <tdf tdfFieldId="additionalCompanyInfo" /> </field>
```

### Custom Java Class

If you do not want to map a field using an XPath or TDF source, you can add a custom Java class that points to a custom `IFieldValueGetter` interface. This interface can be used to grab custom field data or non-ACORD data.

Refer to the Creating a Custom `IFieldValueGetter` Interface section below for more information. This can be especially useful if you're attempting to map a custom Adobe Acrobat form or you want to link to "if else" else scenario.

## Example

The following configuration points to the `OtherLegalEntityValueGetter` interface to retrieve the value for the Legal Entity value on the ACORD 130 form.

```
<field javaClass="com.agencyport.workerscomp.pdf.eform.OtherLegalEntityValueGetter">
<eLabel>NamedInsured_LegalEntity_OtherDescription_A</eLabel> <tdf tdfFieldId="legalEntity" /> </field>
```

#### **Creating a Custom IFFieldValueGetter Interface**

When using custom Java classes to map field data, the class must be pointed to a custom `IFFieldValueGetter` interface to retrieve the data values. The following is the `IFFieldValueGetter` interface template provided in the SDK:

```
/** * implement this interface to get the field for the field. */ public interface IFFieldValueGetter { /** * get the field value * @param field The field artifact * @param context The context with ACORD xml, security profile * @param indices the index array * @param customParams custom parameters * @return the value for the field * @throws APException if the attempt to get the fields value ran into a catastrophic issue. */ FieldValue get(FormField field, PDFContext context, int[] indices, Map<String, String> customParams) throws APException; }
```

The `IFFieldValueGetter` interface can be configured to grab certain field values based on defined criteria. This can be used to grab specific data that is defined differently on the screen than on the form or non-ACORD data.

#### **Example**

The following `IFFieldValueGetter` interface looks to see if the Other Legal Entity field value is any of the options that display on the form. If the value is not an option that displays on the form, then Other is checked on the form. This is a unique scenario since the form only provides nine available field options and the values in AgencyPortal include up to 30 available options.

```
/** * The OtherLegalEntityGetter class supports the rendering of the other checkbox and description * for the Legal Entity field. */ public class OtherLegalEntityValueGetter extends OtherCheckboxValueGetter { /** * The <code>VALUES_OTHER_THAN_OTHER</code> contains the values that are accommodated * by the check boxes previous to the other check box for the legal entity. */ private static final Set<String> VALUES_OTHER_THAN_OTHER = Collections.unmodifiableSet(new HashSet<String>(ArrayHelper.createImmutableList("SolePrp", "PT", "CP", "SS", "LL", "JV", "TR", "IN", "UA"))); /** * Constructs an instance. */ public OtherLegalEntityValueGetter() { } /** * {@inheritDoc} */ @Override protected boolean shouldRenderValue(String value, FormField field, PDFContext context, int[] indices, Map<String, String> customParams){ return !VALUES_OTHER_THAN_OTHER.contains(value); } }
```

#### **Example**

The following is an example of a custom `IFFieldValueGetter` interface that retrieves non-ACORD data. It checks to see if there are any Work Item Assistant comments and, if so, print them on the form.

```
/** * WorkItemAssistantCommentFieldValueGetter will get all the remarks */ public class WorkItemAssistantCommentFieldValueGetter implements IFFieldValueGetter { /** * construct a new instance */ public WorkItemAssistantCommentFieldValueGetter() { } /** * {@inheritDoc} */ @Override public FieldValue get(FormField field, PDFContext context, int[] indices) { DatabaseResourceAgent dra = new DatabaseResourceAgent(this); WorkitemAssistantOperationProvider provider = new WorkitemAssistantOperationProvider(); StringBuilder builder = new StringBuilder(); try { List<BaseEntry> comments = provider.listComments(dra, context.getWorkItem(0).getWorkItemId(), intValue()); for(BaseEntry entry : comments){ builder.append(((CommentEntry)entry).getComment()); builder.append("\n"); } } catch (APException e) { } finally{ dra.closeDatabaseResources(this); } FieldValue value = new FieldValue(); value.addValue(builder.toString()); return value; } }
```

#### **How to Handle Unique Form Fields**

Fields that are unique to only one form can be mapped directly in the `pdfDefinition` file under the corresponding form.

#### **Example**

The following fields are only present on the ACORD 130:

```
<form name="ACORD130" resourceId="ACORD-0130-2013-09r2.pdf"> <fields> <field>
<eLabel>NamedInsured_InBusiness_YearCount_A</eLabel> <tdf tdfFieldId="yrsInBusiness" /></field> <field type="checkbox">
javaclass="com.agencyport.workerscomp.pdf.eform.OtherLegalEntity Value Getter">
<eLabel>NamedInsured_LegalEntity_OtherIndicator_A</eLabel> <tdf tdfFieldId="legalEntity" /></field> <field>
javaclass="com.agencyport.workerscomp.pdf.eform.OtherLegalEntity Value Getter">
<eLabel>NamedInsured_LegalEntity_OtherDescription_A</eLabel> <tdf tdfFieldId="legalEntity" /></field> <field>
<eLabel>Producer_CustomerServiceRepresentative_FullName_A</eLabel> <tdf tdfFieldId="CSRName" /></field> </fields> </form>
```

#### **How to Handle Common Form Fields**

Fields that are common among multiple forms can be mapped in a resource file that is then shared among corresponding forms. Link the resource file in the `pdfDefinition` file under the forms that share the common fields.

#### **Example**

The following forms share common fields with other forms:

```
<form resourceId="AP_WORKITEM_ASSISTANT_COMMENTS.pdf"> <include
resourceId="../../../../commercial/pdf/definitions/acord_formHeader.xml"/> <include
```

```
resourceId="../../shared/pdf/definitions/workItemAssistantComments.xml"/> </form> <form resourceId="ACORD_FORM_137MA.pdf"><include resourceId="../../commercial/pdf/definitions/acord_formHeader.xml"/> <include resourceId="acord137Common.xml"/> </form><form resourceId="ACORD_FORM_137NJ.pdf"> <include resourceId="../../commercial/pdf/definitions/acord_formHeader.xml"/> <include resourceId="acord137Common.xml"/> </form>
```

The AP\_WORKITEM\_ASSISTANT\_COMMENTS.pdf form is a shared form that has a common header with other forms. The acord\_formHeader.xml resource file contains common header fields that are common to a variety of ACORD forms; thus, it can be reused for many different forms. For state forms, a common resource file can be created to include fields that are common among the state forms. In the above example, ACORD 137 MA and NJ share a set of common fields.

#### When to Create Resource Files

A resource file can be used when you have a group of fields that are common across various forms.

#### Best Practice

We recommend that you name resource files with a generic name that can be easily used for different forms. For example, accord\_producerNameInsured.xml is a resource file that includes producer named insured information that can be used with ACORD 125 and ACORD 130.

After a resource file is created, you must include it under the corresponding form in the pdfDefinition file using the <include> attribute. The resource ID is the location where the form is saved within your project directory.

#### Example

```
<form resourceId="ACORD_FORM_125.pdf"> <include resourceId="../../commercial/pdf/definitions/acord_formHeader.xml"/> <include resourceId="../../commercial/pdf/definitions/acord125_natureOfBusiness.xml"/> </form>
```

A form can have one or several resource files, depending on whether the fields are unique or common among other forms.

#### How to Handle Repeating Fields

##### Simple Repeat

ACORD forms tabulate information. For example, ACORD 127 lists driver information. Since the number of drivers is unknown at the time of data mapping configuration, it is not feasible to map all the drivers. The effort to map all the drivers can be tremendous if a policy involves a great number of drivers. To simplify the mapping, schedules are introduced.

Fields in Adobe Acrobat forms are uniquely identified by field names. Field names are separated into eLabels and eLabelCounters in ACORD eForms. The combination of an eLabel and eLabelCounter makes a field name. This naming convention is very helpful for repeating form fields since you only need to map the fields once. However, repeating fields have varying eLabelCounters even though they share eLabels. This naming scheme allows us to create the field names on the fly for the repeating fields.

Schedules vary in their complexity. The schedule, such as driver information on ACORD 127, simply loops through each driver. For this type of schedule, map the fields for one driver and let the SDK pull out all the drivers.

#### Example

```
<schedule name="Driver" overflow="driverSchedule" > <repeat size="13" source="Comm1AutoLineBusiness.CommlDriver"> <include resourceId="acord127_drivers.xml"/> </repeat> </schedule>
```

In the above example, acord127\_drivers.xml contains all the mappings for one driver. The SDK will go through ACORD XML and pull out all drivers from the Comm1AutoLineBusiness.CommlDriver source. ACORD 127 can print 13 drivers. If there are more than 13 drivers, where will the rest of the drivers go? The schedule designates an overflow to print the additional drivers using ACORD 163, which is specifically designed for additional drivers.

#### Example

```
<overflow name="driverSchedule" size="24" resourceId="ACORD_FORM_163.pdf" scheduleIndicator="CommercialVehicleLineOfBusiness_Attachment_CommercialAutoDriverInformationScheduleIndicator_A" > <include resourceId="../../commercial/pdf/definitions/acord_formHeader.xml"/> </overflow>
```

ACORD 163 can print 24 drivers. Additional ACORD 163 forms are added if needed. What if there is no ACORD form for the overflow? For example, ACORD 130 has room for three locations; however, no overflow ACORD form is available. Decisions must be made to indicate where to print the rest of the locations. The following options are available:

- Option 1: Create a custom PDF form for location overflow

- Option 2: Implement `IFormProcessor.postBuildProcess()` to print out the overflow location. The data entries for schedules are removed as they are emitted out. When `IFormProcessor.postBuildProcess()` is executed, the rest of the data entries for schedules are still available.

It is up to a project team's discretion whether to create their own custom overflow form for forms that have schedule attachments. Refer to the Add Schedules and Overflows section below for more information.

## Double Repeats

The repeating schedule may become more complicated. For example, page 2 of ACORD 130 lists all the rating classes by state. A separate page 2 is created for each state. To accommodate the scenario of repeating within repeating, the SDK allows the schedule itself to repeat. The following is a sample configuration of this type of schedule.

## Example

```

<schedule name="stateRating" source="WorkCompLineBusiness.WorkCompRateState" overflow="stateRating"> <fields> field>
<eLabel>WorkersCompensation_RateState_StateOrProvinceName_A</eLabel>
<xpath>WorkCompLineBusiness.WorkCompRateState.StateProvCd</xpath> </field> </fields> <repeat size="14"
source="WorkCompLineBusiness.WorkCompRateState.WorkCompLocInfo.WorkCompRateClass" resourceId="ACORD-0130-2013-09r2-
page2.pdf"> <include resourceId="stateRating.xml" /> </repeat> </schedule> <overflow name="stateRating" size="14" resourceId="ACORD-
0130-2013-09r2-page2.pdf"> <pagination pageLabel="WorkersCompensation_RateState_PageNumber_A" /> </overflow>
totalPageLabel="WorkersCompensation_RateState_TotalPageNumber_A" /> </overflow>

```

The SDK will iterate through the schedule source, `WorkCompLineBusiness.WorkCompRateState` in the above example, to catch all the states and then iterate through all the rating classes for each state based on the source in repeat (`WorkCompLineBusiness.WorkCompRateState.WorkCompLocInfo.WorkCompRateClass` in the above example). For each state, a new page 2 using the resource on repeat will be added. If more than one sheet is needed for the state, additional sheets with the resource on overflow will be added. The support repeating within repeating tremendously simplifies the data mapping.

### **Field Repeat on Repeat**

In this example, a field on a row may repeat itself. For example, multiple drivers may drive a vehicle with various use percent. ACORD 90ME includes the following table:

Within the table, each vehicle can list 6 Driver Use %. The use % fields have the same eLabel, `Vehicle_UsePercent`. Their eLabelCounters vary from A to F for the first row, G to L for the second row, and so on. The driver use % is retrieved from different aggregates in ACORD data with reference to the vehicle. In this case, `Vehicle_UsePercent` field can be classified as a field with multiple values. The following is an example of the mapping configuration for this scenario.

### Example

The table is classified as a repeating schedule and only defines the mapping for the fields of one row. The following is how this scenario would be coded:

```
<field multiValue="true" javaClass="com.agencyport.persauto.pdf.eform.DriverUsePercentFieldValue Getter">
<eLabel>Vehicle _UsePercent</eLabel> <eLabelCounters> <eLabelCounter>A</eLabelCounter> <eLabelCounter>B</eLabelCounter>
<eLabelCounter>C</eLabelCounter> <eLabelCounter>D</eLabelCounter> <eLabelCounter>E</eLabelCounter>
<eLabelCounter>F</eLabelCounter> </eLabelCounters> <eLabelCounters> <eLabelCounter>G</eLabelCounter>
<eLabelCounter>H</eLabelCounter> <eLabelCounter>I</eLabelCounter> <eLabelCounter>J</eLabelCounter>
<eLabelCounter>K</eLabelCounter> <eLabelCounter>L</eLabelCounter> </eLabelCounters> <eLabelCounters>
<eLabelCounter>M</eLabelCounter> <eLabelCounter>N</eLabelCounter> <eLabelCounter>O</eLabelCounter>
<eLabelCounter>P</eLabelCounter> <eLabelCounter>Q</eLabelCounter> <eLabelCounter>R</eLabelCounter> </eLabelCounters>
```

```
<eLabelCounters> <eLabelCounter>S</eLabelCounter> <eLabelCounter>T</eLabelCounter> <eLabelCounter>U</eLabelCounter>
<eLabelCounter>V</eLabelCounter> <eLabelCounter>W</eLabelCounter> <eLabelCounter>X</eLabelCounter> </eLabelCounters> </field>
```

The `Vehicle_UsePercent` is mapped as one field with multiple values, which are received via Java class `DriverUsePercentFieldValue` Getter:

```
/** * The class will retrieve the driver use percent for each vehicle */ public class DriverUsePercentFieldValue implements
IFieldValueGetter { /** * <code>PATH</code> part of apxth */ private static String PATH = "PersPolicy.DriverVeh[@VehRef=""; /**
constructor*/ public DriverUsePercentFieldValue() { } /** * {@inheritDoc} */
@Override public FieldValue get(FormField field,
PDFContext context, int[] indices) throws APException { FieldValue fieldValue = new FieldValue(); String indexRef =
context.getAcordData().getFieldValue("Pers AutoLineBusiness.Pers Veh.@id", indices, ""); String path = PATH + indexRef + "]"; int count =
context.getAcordData().getCount(path); for(int i = 0; i < count; i++){ String fieldId = PATH + indexRef + "'&& position()=" + i +
"]".UsePct"; String percent = context.getAcordData().getFieldValue(fieldId, ""); fieldValue.addValue(percent); } return fieldValue; } }
```

### When to Use indexRef Fields

The `indexRef` attribute can be used to cross link data if the data source is reference in a different XPath. In other words, it's used to find field data from one aggregate (XPath entered for the `indexRef`) for another (XPath entered for the field mapping). To do this:

1. Enter the XPath of where you want `indexRef` to source the field data.
2. Enter the form eLabel where you want this data to print.
3. Enter the corresponding XPath of the field, except use `[@id=' {indexRef} ']` to reference the exact field ID of where you want `indexRef` to cross link the data. This uses the actual field ID to find the data source and enter it into the form field.

### Example

The following field mapping configuration uses `indexRef` to get the individual's location (`@LocationRef`). Since the location can include an entire address, the XPath says to only grab the state abbreviation from the location by referencing the field ID (`Location[@id=' {indexRef} ']`).

```
<field indexRef="WorkCompLineBusiness.WorkCompIndividuals.@LocationRef">
<eLabel>WorkersCompensation_Individual_StateOrProvinceCode</eLabel> <xpath>Location[@id='${indexRef}'.Addr.StateProvCd</xpath>
</field>
```

### When to Access Index and Count

Some schedules require you to print the current entry and total repeating count. The index and count are available via the `itemIndex` and `totalItems` variables.

### Example

```
<field defaultValue="${itemIndex}/${totalItems}"> <eLabel> Driver_ProducerIdentifier </eLabel> </field>
```

### How to Add Custom Field Formatting

Fields are formatted based on the TDF value. If you don't want a field to display using the default format, you can add your own formatting using one of the following:

- A format type, such as date or number.
  - A date format type defaults to the localized version of date (e.g., MM/DD/YYYY); otherwise, a pattern is needed to specify the format.
  - A number format requires a specific pattern.

### Example

The following example includes a default date configuration:

```
<field> <eLabel>Policy_NormalAnniversaryRatingDate_A</eLabel> <format type="date" /> <tdf
tdfFieldId="anniversaryRatingDate" /> </field>
```

The following example includes a date configuration with a defined pattern.

```
<field> <eLabel>Driver_LicensedYear</eLabel>
<xpath>CommlAutoLineBusiness.CommlDriver.DriverInfo.License.LicensedDt</xpath> <format type="date"
pattern="yyyy"/></field>
```

The following example includes a number configuration with a defined pattern:

```
<field> <eLabel>WorkersCompensation_Individual_RemunerationAmount</eLabel> <format type="number"
pattern="####,###,##" /><tdf tdfFieldId="WorkCompIndividualsRemuneration" /></field>
```

- A Java class to use custom formatting.

### Example

The following is an example of a custom format for include/exclude fields:

```
/** * Format Included/Excluded. Included is stored as '1', and Excluded is stored as '0' in Acord data *Print out INC or EXC on the PDF form *
*/ public class IncludedExcludedCodeFormatter implements IFieldValueFormatter { /** * constructor */ public
IncludedExcludedCodeFormatter() { } /** * {@inheritDoc} */ @Override public String getFormattedValue(FormField field, String pattern,
String value) { if("I".equals(value)){ return "INC"; }else if("E".equals(value)){ return "EXC"; } return value; } }
```

### How to Configure a Field Type (Check Box or Question)

If a field is a check box or question, you must add a field type to the field configuration.

### Example

The following is a sample of a check box field configuration:

```
<fields> <field type="checkbox"> <eLabel>Vehicle_CombinedSingleLimit_LimitIndicator_A</eLabel>
<xpath>CommlAutoLineBusiness.CommlCoverage[CoverageCd='CSL'].CoverageCd</xpath> </field> </fields>
```

The following is a sample of a question field configuration:

```
<fields> <field type="question"> <eLabel>CommercialPolicy_Question_AAICode_A</eLabel>
<xpath>CommlPolicy.QuestionAnswer[QuestionCd='GENRL34'].YesNoCd</xpath> </field> </fields>
```

There are some scenarios where there are two check boxes on the form and only one should be checked based on the question. This requires you to include a yesValue to indicate which field to check.

### Example

ACORD 130 has a question that asks whether the policy is participating or non-participating. If the field value is 1, Participating is checked; if the field value is 0, Non-Participating is checked.

```
<field type="checkbox" yesValue="1"> <eLabel>Policy_ParticipatingIndicator_A</eLabel> <tdf tdfFieldId="participatingPlanInd" /></field>
<field type="checkbox" yesValue="0"> <eLabel>Policy_NonParticipatingIndicator_A</eLabel> <tdf tdfFieldId="participatingPlanInd" />
</field>
```

### How to Configure a Check Box Group

ACORD forms have lots of check box "groups", which are driven off a single XPath in ACORD. Depending on the value in that XPath, a different check box should be checked. For example, the Additional Interest on ACORD 127 lists various interests. The value of interest selected in AgencyPortal determines which interest is checked. A checkBoxGroup can be used to simplify the configuration.

### Example

The following is a sample of a check box group configuration:

```
<checkBoxGroup>
<xpath>CommlAutoLineBusiness.CommlRateState.CommlVeh.AdditionalInterest.AdditionalInterestInfo.NatureInterestCd</xpath> <check kBox
trueValue="ADDIN"> <eLabel>AdditionalInterest_Interest_AdditionalInsuredIndicator</eLabel> </checkbox> <checkbox
trueValue="EMPL"> <eLabel>AdditionalInterest_Interest_EmployeeAsLessorIndicator</eLabel> </checkbox> <checkbox
trueValue="LIEN"> <eLabel>AdditionalInterest_Interest_LienholderIndicator</eLabel> </checkbox> <checkbox trueValue="LOSSP">
<eLabel>AdditionalInterest_Interest_LossPayeeIndicator</eLabel> </checkbox> <checkbox trueValue="OWNER">
<eLabel>AdditionalInterest_Interest_OwnerIndicator</eLabel> </checkbox> <checkbox trueValue="RG">
<eLabel>AdditionalInterest_Interest_RegistrantIndicator</eLabel> </checkbox> <checkbox>
<eLabel>AdditionalInterest_Interest_OtherIndicator</eLabel> <checkboxDetail>
<eLabel>AdditionalInterest_Interest_OtherDescription</eLabel> <tdf tdfFieldId="natureInterestCd"/></checkboxDetail> </checkbox>
</checkboxGroup>
```

## How to Aggregate Multiple ACORD Paths into One Form Field

There are cases where the data for a PDF form field is stored in multiple XPaths in ACORD data. A good example of this is a PDF field for full name where the first name, middle name and last name are stored separately in ACORD data.

### Example

```
<field> <eLabel>NamedInsured_AccountingContact_FullName_A</eLabel> <xpath order="1">
CommPolicy.MiscParty[MiscPartyInfo.MiscPartyRole Cd='A C'].GeneralPartyInfo.Name Info.PersonName.GivenName </xpath> <xpath
order="2"> CommPolicy.MiscParty[MiscPartyInfo.MiscPartyRole Cd='A C'].GeneralPartyInfo.Name Info.PersonName.OtherGivenName
</xpath> <xpath order="3">
CommPolicy.MiscParty[MiscPartyInfo.MiscPartyRole Cd='A C'].GeneralPartyInfo.Name Info.PersonName.Surname </xpath> </field>
```

### Add Form Variables

Form variables allow you to identify XPath qualifiers when multiple forms use common fields (e.g., state version of forms). The variable in the XPath can be replaced with a variable (e.g., state code) when state specific forms are built.

### Example

ACORD 137 fields for NH, NH, MA, NY, RI, OH, PA and VT have common fields. They are placed in the acord137common.xml file, which is included in the configuration for all state specific forms.

The following variable configuration is added for ACORD 137 MA:

```
<form resourceId="ACORD_FORM_137MA.pdf"> <variables> <variable name="stateCode" value="MA"/> </variables> <include
href="../../shared/pdf/definitions/acord_formHeader.xml"/> <include href="acord137Common.xml"/> <include href="acord137MA.xml"/>
<criteria> <condition xpath="CommAutoLineBusiness.CommlRateState[StateProvCd='MA'].StateProvCd" value="MA"/> </criteria> </form>
```

The variable attribute name="stateCode" value="MA" replaces {stateCode} in the mapped XPaths included in acord137common.xml.

### Add Form Validation

Form validation allows you to define certain criteria to only generate forms under a certain condition. This allows you to identify whether to only generate a form based on a transaction type or when a field equals a certain value (condition). A condition can be based on an XPath, TDF or external value (using a Java class).

Form validation includes the following elements:

javaclass	The customJava class name, which implements com.agencyport.pdf.IFormValidator. If evaluated result is true, the form will be built.
condition	Condition of the field. This can be an XPath or TDF equal to a certain value.
transactionType	List of transaction types with which the form will be built.

### Example

The following example indicates that ACORD 137 MA only generates if the value of the CommAutoLineBusiness.CommlRateState[StateProvCd='MA'].StateProvCd XPath is MA:

```
<form resourceId="ACORD_FORM_137MA.pdf"> <variables> <variable name="stateCode" value="MA"/> </variables> <include
resourceId="../../commercial/pdf/definitions/acord_formHeader.xml"/> <include resourceId="acord137Common.xml"/> <include
resourceId="acord137MA.xml"/> <criteria> <condition
xpath="CommAutoLineBusiness.CommlRateState[StateProvCd='MA'].StateProvCd" value="MA"/> </criteria> </form>
```

When the defined criteria is evaluated to true, the form is generated. The default validator (IFormValidator) is com.agencyport.pdf.FormValidator, which declares whether the form is valid if all of the defined conditions and transaction types are met. You can use a customJava class to point to a custom IFormValidator if you do not want to use an XPath or TDF configuration.

### Add Schedules and Overflows

Schedules and overflows are used to define and print additional, repeating roster item data, such as a list of drivers or vehicles.

- A schedule is used to reference repeating roster items.
- An overflow is an additional schedule attachment form that prints items that do not fit on the form. It can be a designated ACORD form or a custom form.

Some forms have schedule attachments designated, but others do not. In some cases, you may need to create your own overflow form using Adobe Acrobat Pro. It is up to each project team's discretion how they handle overflow data.

There are also a variety of scenarios that can exist based on the data presented on the form. Some forms may have multiple overflow attachments while others may only have one. The following are some scenarios that may exist where you must determine how to handle overflow data:

- Commercial Auto ACORD 127 provides pre-determined schedule attachments to print overflow roster data. These are ACORD 163 to print up to 24 additional drivers on each attachment, ACORD 45 to print an additional four additional interests, and ACORD 129 to print up to five additional vehicles. The following is the configuration that relays this information:

```
<schedule name="Driver" overflow="driverSchedule" ><repeat size="13" source="CommAutoLineBusiness.CommlDriver">
<include resourceId="acord127_drivers.xml"/> </repeat></schedule> <overflow name="driverSchedule" size="24"
resourceId="ACORD_FORM_163.pdf"
scheduleIndicator="CommercialVehicleLineOfBusiness_Attachment_CommercialAutoDriverInformationScheduleIndicator_A">
<include resourceId="../../commercial/pdf/definitions/acord_formHeader.xml"/> </overflow> <overflow name="vehicleSchedule"
size="5" resourceId="ACORD_FORM_129.pdf"
scheduleIndicator="CommercialVehicleLineOfBusiness_Attachment_VehicleScheduleIndicator_A"> <include
resourceId="../../commercial/pdf/definitions/acord_formHeader.xml"/> </overflow> <overflow name="additionalInterestSchedule"
size="4" resourceId="ACORD_FORM_45.pdf"
scheduleIndicator="CommercialVehicleLineOfBusiness_Attachment_AdditionalInterestIndicator_A"><include
resourceId="../../commercial/pdf/definitions/acord_formHeader.xml"/> </overflow>
```

- Workers Comp ACORD 130 includes a rating information roster; however, the form only supplies enough room to enter up to 14 items for one state. To include additional items and/or items for another state, you must create a separate overflow attachment form that is a copy of page 2 of ACORD 130 (e.g., ACORD-0130-2013-09r2-page2.pdf). The following is the configuration that relays this information:

```
<schedule name="stateRating" source="WorkCompLineBusiness.WorkCompRateState" overflow="stateRating" ><fields> <field>
<eLabel>WorkersCompensation_RateState_StateOrProvinceName_A</eLabel>
<xpath>WorkCompLineBusiness.WorkCompRateState.StateProvCd</xpath> </field> <field>
<eLabel>WorkersCompensationStateCoverage_ExperienceOrMerit_ModificationFactor_A</eLabel>
<xpath>WorkCompLineBusiness.WorkCompRateState.CreditOrSurcharge[CreditSurchargeCd='AREM'].NumericValue.FormatModifier</xpath> </field> </fields> <repeat size="14"
source="WorkCompLineBusiness.WorkCompRateState.WorkCompLocInfo.WorkCompRateClass" resourceId="ACORD-0130-2013-09r2-page2.pdf" > <include resourceId="stateRating.xml" /> </repeat></schedule> <overflow name="stateRating" size="14"
resourceId="ACORD-0130-2013-09r2-page2.pdf" > <pagination pageLabel="WorkersCompensation_RateState_PageNumber_A"
totalPageLabel="WorkersCompensation_RateState_TotalPageNumber_A" /> </overflow>
```

- There may be overflow data that does not have a corresponding schedule attachment. In this case, you will have to create your own custom form to print the data (if your project team determines that is what you want to do). The following is an example of how this can be configured:

```
<schedule name="additionalData" overflow="additionalData" ><repeat size="0"
source="CommAutoLineBusiness.CommlRateState.CommlVeh" > <include resourceId="additionalData125.xml"/> </repeat>
</schedule> <overflow name="additionalData" size="2" resourceId="ADDITIONAL_DATA_FORM125.pdf" > <include
resourceId="../../commercial/pdf/definitions/acord_formHeader.xml"/> <pagination pageLabel="Current_Page_A"
totalPageLabel="Total_Page_A" /> </overflow>
```

## Schedules

A schedule is used to reference repeating roster information. Roster entries are differentiated in ACORD forms using the eLabelCounter. For example, a Commercial Auto policy may have a number of drivers. The schedule configuration allows you to only map one driver and the SDK will collect all of the drivers, place them into the designated ACORD form and move any extra drivers to a specified overflow attachment schedule.

## Note

Any custom data or custom forms added to a pdfDefinition file must comply with the pdfDefinition schema.

A schedule assumes that the form's eLabelCounters use consecutive alpha letters from A to Z. For example, ACORD 127 has a Driver\_GivenName field with eLabelCounters A to M. The overflow schedule ACORD 163 has a Driver\_GivenName field with eLabelCounters A to X. If an overflow field uses a different eLabelCounter scheme, custom eLabelCounters can be injected into the configuration. Refer to the How to Handle Repeating Fields section for more information.

The above example shows that the additional interest on ACORD 127 is mapped to the overflow. ACORD 127 can print two additional interests and the rest will be moved to the additionalInterestSchedule, which is ACORD 45. The field values will be retrieved from the field source (e.g., CommlAutoLineBusiness.CommlRateState.CommlVeh.AdditionalInterest) within the configuration (using the `<repeat>` element). This source sample has three repeated elements. The SDK will iterate through all states, vehicles and additional interests to pull out all additional interest.

This schedule/overflow sample for additional interests and drivers is a simple repeat. The schema also supports two level repeats. For example, page 2 of ACORD 130 lists the rating class codes by states. The mapping can be done one state at a time; however, the states and number of states in the policy are unknown. It is difficult to do the mapping with unknown factors. This will require a double repeat concept where both the schedule and repeat has XPath sources. At the schedule level, all of the rating states will be pulled and at the repeat level, the rating classes for the state are pulled.

The following metadata can be added when using a repeat element:

<code>name</code>	The name of the overflow and reference to the overflow schedule.
<code>size</code>	The number of overflow entries on the form.
<code>source</code>	The data source for the overflow repeating aggregates.
<code>resourceID</code>	The form file for additional schedule sheet.

### Example

The following example indicates that the states are pulled via `WorkCompLineBusiness.WorkCompRateState`. A new page will be added for every new state using the `ACORD_130_STATE_RATING_SHEET.pdf` template. If the number of rating classes for a state is more than the template sheet can hold, additional sheets for the state can be added using the `ACORD_130_STATE_RATING_SHEETS.pdf` template (as indicated in the overflow configuration).

```
<schedule name="ratingInfo" source="WorkCompLineBusiness.WorkCompRateState" overflow="ratingInfo"> <include
resourceId="premium.xml"/> <repeat source="WorkCompLineBusiness.WorkCompRateState.WorkCompLocInfo.WorkCompRateClass"
size="14" resourceId="ACORD_130_STATE_RATING_SHEET.pdf"> <include resourceId="ratingClass.xml"/> </repeat> </schedule>
<overflow name="ratingInfo" size="14" resourceId="ACORD_130_STATE_RATING_SHEET1.pdf"> <pagination
pageLabel="WorkersCompensation_RateState_PageNumber_A" totalPageLabel="WorkersCompensation_RateState_TotalPageNumber_A" />
</overflow>
```

Schedule fields can be mapped directly in the `pdfDefinition` file if they are unique to the schedule only or by referencing a common resource file.

### Example

The following is an example of a schedule that includes field mapping directly in the `pdfDefinition` file:

```
<schedule name="includedExcludedIndividuals"> <repeat size="4" source="WorkCompLineBusiness.WorkCompIndividuals"> <fields> <field
indexRef="WorkCompLineBusiness.WorkCompIndividuals.@LocationRef">
<eLabel>WorkersCompensation_Individual_StateOrProvinceCode</eLabel> <xpath>Location[@id='${indexRef}'.Addr.StateProvCd</xpath>
</field> <field indexRef="WorkCompLineBusiness.WorkCompIndividuals.@LocationRef">
<eLabel>WorkersCompensation_Individual_LocationProducerIdentifier</eLabel>
<xpath>Location[@id='${indexRef}'.ItemInfo.AgencyId</xpath> </field> <field>
javaclass="com.agencyport.shared.pdf.eform.FullNameInfoGetter"> <eLabel>WorkersCompensation_Individual_FullName</eLabel>
<xpath>WorkCompLineBusiness.WorkCompIndividuals.NameInfo</xpath> </field> <field>
<eLabel>WorkersCompensation_Individual_BirthDate</eLabel> <format type="date"/><tdf tdfFieldId="WorkCompIndividualsDOB" />
</field> <field> <eLabel>WorkersCompensation_Individual_TitleRelationshipCode</eLabel> <tdf tdfFieldId="WorkCompIndividualsTitle" />
</field> <field> <eLabel>WorkersCompensation_Individual_OwnershipPercent</eLabel> <tdf
tdfFieldId="WorkCompIndividualsOwnershipPercent" /> </field> <field>
<eLabel>WorkersCompensation_Individual_DutiesDescription</eLabel> <tdf tdfFieldId="WorkCompIndividualsDuties" /> </field> <field>
<eLabel>WorkersCompensation_Individual_IncludedExcludedCode</eLabel> <tdf tdfFieldId="WorkCompIndividualsIncExcCd" /> </field>
<field> <eLabel>WorkersCompensation_Individual_RatingClassificationCode</eLabel>
<xpath>WorkCompLineBusiness.WorkCompIndividuals.RatingClassificationCd</xpath> </field> <field>
<eLabel>WorkersCompensation_Individual_RemunerationAmount</eLabel> <format type="number" pattern="#,,##,##" /><tdf
tdfFieldId="WorkCompIndividualsRemuneration" /> </field> </fields> </repeat> </schedule>
```

The following is an example of a schedule that includes a reference to a common resource file that includes field mapping:

```
<schedule name="location"> <repeat size="3" source="Location"> <include resourceId="locations.xml" /> </repeat> </schedule>
```

When mapping a schedule, you must include the following metadata:

<b>name</b>	Name of the overflow schedule.
<b>size</b>	The number of repeat entries that the form can hold. If there are any additional entries, they will print on an overflow attachment if one is configured.
<b>overflow</b>	The ACORD or custom attachment resource to use for entries more than the form can hold.
<b>source</b>	The source of the overflow. This must be an XPath. It's the source for the roster page. The field values are retrieved from the source and the SDK will iterate through the path to pull out the field data.
<b>include</b>	Reference to the XML resource (i.e., the resource file that includes the mapped fields to print on the schedule) to include.

### Example

The following example indicates that the schedule name is Driver, 13 drivers can print on the form, any additional items will be deferred to the driverSchedule overflow and the source of the schedule data comes from `CommlAutoLineBusiness.CommlDriver`.

```
<form resourceId="ACORD_FORM_127.pdf"> <schedule name="Driver" size="13" overflow="driverSchedule" source="CommlAutoLineBusiness.CommlDriver"> <include resourceId="acord127_drivers.xml"/> </schedule> </form>
```

### Overflows

An overflow represents an attachment, such as ACORD 165, 101, 45 or any customPDF forms. Attachments are referenced by the schedule. When entries do not fit onto ACORD forms, the rest of the entries are printed onto the overflow attachment schedule. Additional schedules are added if the number of overflow entries exceeds the schedule capacity.

When mapping overflow, you must include the following metadata:

<b>name</b>	Name of the overflow attachment.
<b>size</b>	The number of repeat entries that the form can hold.
<b>resourceID</b>	The PDF form file to use as a resource.
<b>scheduleIndicator</b>	ACORD forms usually have check boxes to indicate there are attachments. This element references the eLabel for the corresponding check box, such as <code>CommercialVehicleLineOfBusiness_Attachment_CommercialAutoDriverInformationScheduleIndicator</code> on ACORD 127.
<b>type</b>	The overflow type.

### Example

The following example indicates that the overflow name is driverSchedule, it can print up to 24 rosteritems and uses ACORD Form 163 to print this overflow data.

```
<schedule name="Driver" overflow="driverSchedule" > <repeat size="13" source="CommlAutoLineBusiness.CommlDriver"> <include resourceId="acord127_drivers.xml"/> </repeat> </schedule> <overflow name="driverSchedule" size="24" resourceId="ACORD_FORM_163.pdf" scheduleIndicator="CommercialVehicleLineOfBusiness_Attachment_CommercialAutoDriverInformationScheduleIndicator_A"> <include resourceId="../../commercial/pdf/definitions/acord_formHeader.xml"/> </overflow>
```

ACORD forms that include a schedule indicator check box must be configured in the overflow using the `scheduleIndicator` attribute.

### Example

The following example says to check the `CommercialVehicleLineOfBusiness_CommercialAutoDriverInformationScheduleIndicator_A` check box on ACORD 127 the since ACORD 163 will be attached.

```
<overflow name="driverSchedule" size="24" resourceId="ACORD_FORM_163.pdf"
scheduleIndicator="CommercialVehicleLineOfBusiness_Attachment_CommercialAutoDriverInformationScheduleIndicator_A"> <include
resourceId="../../commercial/pdf/definitions/acord_formHeader.xml"/> </overflow>
```

## Pagination

If you decide to use a custom overflow, you may want to add pagination to the form so that it coincides with the page numbering of the other forms included in the form group.

Pagination includes the following elements:

pageLabel	The eLabel for the current page on the form, such as Form_CurrentPageNumber_A for ACORD 101.
totalPageLabel	The eLabel for the total pages on the form, such as Form_TotalPageNumber_A for ACORD 101.

## Example

The following adds pagination to a customoverflow Additional Data form to use as an attachment to ACORD 127:

```
<overflow name="stateRating" size="14" resourceId="ACORD-0130-2013-09r2-page2.pdf"> <pagination
pageLabel="WorkersCompensation_RateState_PageNumber_A" totalPageLabel="WorkersCompensation_RateState_TotalPageNumber_A" />
</overflow>
```

## Establish Form Groups

After you add all of your forms to the pdfDefinition file, you must add form group configurations if you want forms the form to generate separately. This configuration tells AgencyPortal how to build the forms when the client asks for them. If no form groups are configured, all forms will generate in one PDF file.

You can group similar forms together into one group (e.g., state forms) or list them individually.

Form grouping includes the following elements:

name	The group name.
title	The group title.
formResource	The PDF form to include in the grouping.

## Example

You want all version of ACORD 137 to generate into one PDF but ACORD 125 and ACORD 127 individually. The following is what your configuration could look like:

```
<formGroups> <group name="ACORD137" title="label.Document.AcordForm137">
<formResource>ACORD_FORM_137MA.pdf</formResource> <formResource>ACORD_FORM_137MI.pdf</formResource>
<formResource>ACORD_FORM_137NH.pdf</formResource> <formResource>ACORD_FORM_137NJ.pdf</formResource>
<formResource>ACORD_FORM_137NY.pdf</formResource> <formResource>ACORD_FORM_137PA.pdf</formResource>
<formResource>ACORD_FORM_137RI.pdf</formResource> <formResource>ACORD_FORM_137VT.pdf</formResource> </group><group
name="ACORD125" title="label.Document.AcordForm125"> <formResource>ACORD_FORM_125.pdf</formResource> </group> <group
name="ACORD127" title="label.Document.AcordForm127"> <formResource>ACORD_FORM_127.pdf</formResource> </group>
</formGroups>
```

## Add JSP Tag to Policy Summary Pages

You must add the `view_acord_forms` tag to all policy summary pages that you want to display PDF form download hyperlinks. This tag will get a list of form groups available (based on those you defined in the pdfDefinition configuration file) and creates the applicable download hyperlinks.

There is no need to hardcode download links in the policy summary pages. You only need to add `<ap:view_acord_forms />` to the page.

The following is an example of a policy summary page for Workers Comp:

```

<!--begin workerscomp/policySummary.jsp --> <!--BEGIN STANDARD FRAMEWORK --> <%@page
import="com.agencyport.workitem.model.IWorkItemStatus"%> <%@page import="com.agencyport.workitem.model.IWorkItem"%> <%@
page import="java.util.*", com.agencyport.html.elements.BaseElement, com.agencyport.pagebuilder.Page,
com.agencyport.pagebuilder.BasePageElement, com.agencyport.webshared.IWebsharedConstants,
com.agencyport.connector.IConnectorConstants, com.agencyport.connector.MessageMap, com.agencyport.connector.Result,
com.agencyport.workerscomp.beans.WorkersCompPolicySummary, com.agencyport.webshared.URLBuilder, com.agencyport.jsp.JSPHelper"
%> <%@ taglib prefix="ap" uri="http://www.agencyportal.com/agencyportal"%> <%@ include file="../policychange/pageSetup.jsp" %> <%@
include file="../shared/framework_ui.jsp" %> <% IWorkItem workItem = jspHelper.getWorkItem(); IWorkItemStatus workItemStatus =
workItem.getWorkItemStatus(); String workItemStatusMnemonic = workItemStatus.getMnemonic(); boolean isAlreadySubmitted =
"SUBMIT".equals(workItemStatusMnemonic); String pageTitle = htmlPage.getTitle(); String pageName = htmlPage.getId(); boolean
errorMessagesExist = false; IResourceBundle policySummaryRB =
ResourceBundleManager.get().getHTMLEncodedResourceBundle(ILocaleConstants.POLICY_SUMMARY_BUNDLE);
WorkersCompPolicySummary policySummary = (WorkersCompPolicySummary) request.getAttribute("WORK_POLICY_SUMMARY");
String status = JSPHelper.prepareForHTML(policySummary.getWorkItemStatus().getAfterChangeStatusTitle()); String pdfUrl =
URLBuilder.buildFrontServletURL(jspHelper.getWorkItemId(), "PDFForm", null, "workersComp", URLBuilder.DISPLAY_METHOD, false);
MessageMap messageMap = (MessageMap) request.getAttribute(IWebsharedConstants.MESSAGES); if (messageMap != null) { List<Result>
errors = messageMap.getResultsByType(IConnectorConstants.MESSAGE_ERROR_LITERAL); if (errors.size() > 0) { errorMessagesExist =
true; } } %> <div class="container policy-summary"> <div class="messages"><jsp:include page="../shared/messageTemplate.jsp" flush="true"
/> </div> <h1 class="header"><%=policySummaryRB.getString("header.Title.Summary")%> <span class="label label-
<%=status.toLowerCase()%> ">%=status%></h1>
<p><%=policySummaryRB.getString("prompt.StandardTips.SaveAndExit")%></p> <div class="row padded-box animated fadeIn premiums">
<div class="col-md-4"> <div class="premium monthly">
<h3><%=JSPHelper.prepareForHTML(policySummary.getTotalPremium().toString())%></h3>
<h4><%=policySummaryRB.getString("header.Title.Monthly")%></h4> </div> </div> <div class="col-md-4"> <div class="premium bi-
annual"> <h3><%=JSPHelper.prepareForHTML(policySummary.getTotalPremium().toString())%></h3>
<h4><%=policySummaryRB.getString("header.Title.BiAnnual")%></h4> </div> </div> <div class="col-md-4"> <div class="premium full-
payment"> <h3><%=JSPHelper.prepareForHTML(policySummary.getTotalPremium().toString())%></h3>
<h4><%=policySummaryRB.getString("header.Title.Full")%></h4> </div> </div> </div> <% if (null == policySummary) { %> <div
class="padded-box"> <h3><%=policySummaryRB.getString("header.Title.Summary")%></h3>
<p><%=policySummaryRB.getString("label.Summary.Error")%></p> </div> <% } %> <div class="row animated fadeIn cards"><div
class="col-xs-12 col-lg-4"> <div class="card agency"><h3> <%=policySummary.getAgencyName()%> <i
class="fa fa-angle-down"></i> </h3>
<%=policySummaryRB.getString("label.Agency.ProducerCode")%><%=policySummary.getProducerCode()%>
<%=policySummaryRB.getString("label.Agency.AgencyCustomerID")%> <%=policySummary.getAgencyCustomerId()%>
</div> </div> <div class="col-xs-12 col-lg-4"> <div class="card applicant"><h3> <%=policySummary.getInsured()%> <span class="pull-
right"><i class="fa fa-angle-down"></i> </h3> <%=policySummaryRB.getString("label.Applicant.CoApplicant")%>
<%=policySummary.getCoinsured()%> <%=policySummaryRB.getString("label.Applicant.PolicyTerm")%>
<%=policySummary.getEffectiveDate()%> to <%=policySummary.getExpirationDate()%> </div> </div> <div class="col-xs-12 col-
lg-4"> <div class="card documents"><h3> <%=policySummaryRB.getString("header.Title.Documents")%> <i
class="fa fa-angle-down"></i> </h3> <p> <a href="<%=pdfUrl%>" target="new_window"><%=rb.getString("label.Document.AcordForms")%> </p> </div> </div> <div id="quote"><ap:page
/> </div> <form action="FrontServlet" id="<%="pageName%">" name="<%="pageName%">" method="post"><input type="hidden"
name="TRANSACTION_NAME" id="TRANSACTION_NAME" value="<%="tran.getId()%>" /> <input type="hidden"
name="PAGE_NAME" id="PAGE_NAME" value="<%="htmlPage.getId()%>" /> <input type="hidden" name="WORKITEMID"
value="<%="jspHelper.getWorkItemId()%>" /> <input type="hidden" name="NEXT" /> <input type="hidden" name="METHOD"
value="Process" /><ap:csrf> <script type="text/javascript"> ap.submitQuoteForm = function(action) { var form =
document.forms["<%=pageName%>"]; var next = form["NEXT"]; next.value = action; form.submit(); ap.pleaseWaitLB(true); } </script> <div
id="buttons"><div class="btn-group" role="group"><% if
(tran.getType().equals(IWebsharedConstants.QUICK_QUOTE_TRANSACTION_TYPE)) { %> <input name="convertToApplicationButton"
type="button" class="btn btn-primary" value="Convert to Application" onclick="ap.submitQuoteForm('ConvertToApplication');"
/> <% } %>
<input name="saveAndExitButton" type="button" class="btn btn-primary" value="Save and Exit" onclick="ap.submitQuoteForm('Save and
Exit');"
/> <a href="#" class="btn btn-default" onclick="javascript:window.history.back(-1);return
false;"><%=policySummaryRB.getString("action.Back")%> </div> </div> </form> </div> <jsp:include page="../home/footer.jsp"
flush="true" />

```

# Integration

This section provides information on the following products available to integrate with AgencyPortal:

- [Connect5](#)
- [Turnstile](#)
- [Integration Kits](#)

# Connect5

Connect5 is an enterprise integration solution using XML. It supports the following integration styles:

- File transfer
  - FTP
  - Directory Polling
- Shared Database
  - Database polling
  - SQL XML
- Remote Procedure Invocation
  - Web Services, Servlets and Java RMI exposed to provide request/reply type functionality
- Messaging
  - Connect to messaging system to exchange data and invoke operation

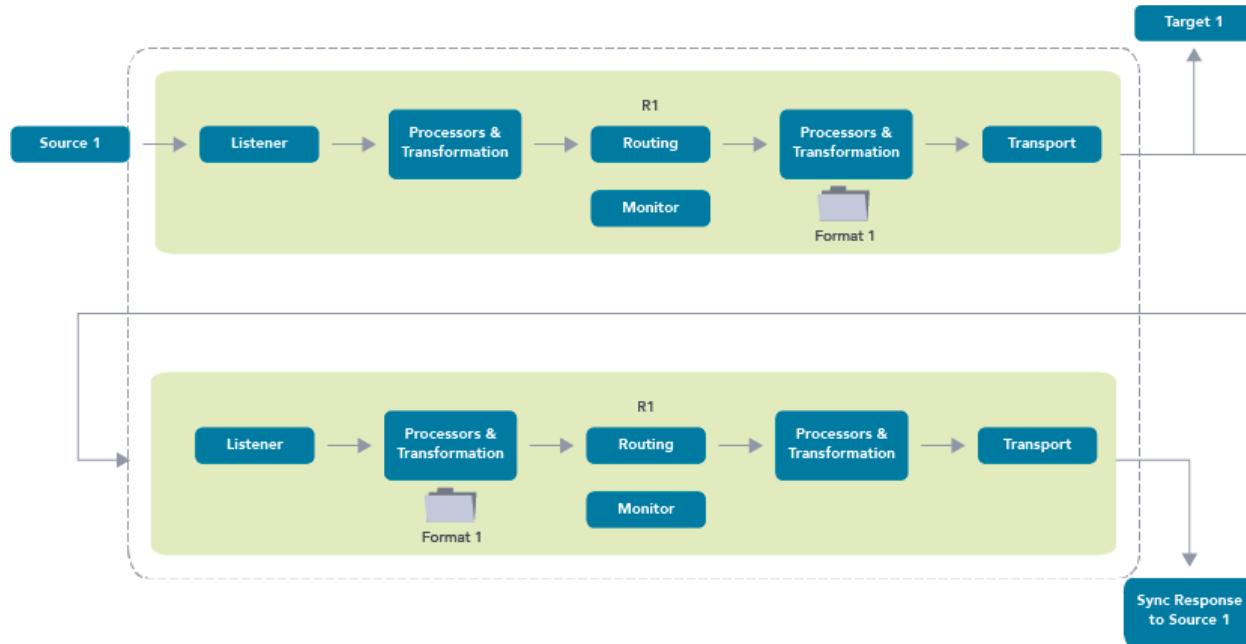
## Features and Architectures

- A variety of built-in listeners
  - Transport layer protocol support to get message into the system
- Multi-step delivery system
  - Independent and flexible step
- Routing capability
  - Route message based on certain criteria
  - Send messages simultaneously to multiple target system
- Message Transformation
  - GUI based data mapper to transform message using XML technology
- A variety of built-in transports
  - Transport layer support to get the message out to the target system
- Process orchestration
  - Fork and join messages using process manager
- Message Auditing
  - File system and database logging
- Integration development
  - Pilotfish variety of testing mode
- Heterogeneous message format (in/out bound)
  - Receive data in a wide variety of format from external systems

# Connect5 Best Practices

This topic outlines the best practices for designing solutions using Connect5. Even though this documentation is specific to request/response integration styles, the practices are applicable to all integration styles.

1. Solution architects should produce a Visio outlining the routes, formats and chaining of routes. Visio basic flow chart or EIP stencils can be used.
2. The number of routes in a given interface should be no more than N+1. Where N is the number of target system. With processor and XSLT transformation in places at the target and source, almost all the solutions can be designed by applying the rule.
3. When chaining routes, they should share formats. See flowchart below.
4. Source side of the route must have a processor to validate all incoming messages; if the messages don't conform, an exception must be thrown. Avoid having targets to route error conditions. Groovy processors can be used, which gives scripting ability to write adhoc validations.
5. If there is more than one target, avoid using default routes. Instead, specifically check for routing predicates in the source; if they are not available, throw exception.
6. All routes, except for error routes, must have transaction monitor registered. Make sure error routes don't throw an exception.
7. All transaction monitors should be configured to include attributes. Transaction data can be included if you think error route can consume the transaction data.
8. If the protocol supports timeouts, then it must be specified in the listener and transport configuration.
9. All modules – listeners, transports and processors - must have unique names and the names should be describing the task it performs.
10. All XSLT templates must be cached.
11. If applicable, try to re-use processors configuration and formats.
12. Leverage XSLT import/include feature instead of copying templates.
13. Populate message ID in the source of the very first route.
14. Xalan Java extensions are available with Xalan interpreted template engine; Xalan compiled template engine does not support Java extensions.
15. Avoid using Xalan Compiled engine; instead, use Saxon.



# Connect5 Server Configuration

This topic outlines the differences between each major JEE server and the customization necessary to run Connect5.

## All Servers

For optimal JVM tuning, please apply the following parameters:

```
-XX:+CMSClassUnloadingEnabled
-XX:+UseConcMarkSweepGC
-XX:MaxPermSize=256m
-XX:PermSize=128m
```

## WebSphere

If you are using the File Upload Widget on WebSphere, we found it had severe memory leaks due to the use of build-in implementation of JAX-WS/SAAJ (AXIS2). To resolve this issue, apply the following JVM parameters:

```
-Djavax.xml.soap.MetaFactory= com.sun.xml.internal.messaging.saaj.soap.SAAJMetaFactoryImpl -Djavax.xml.soap.SOAPConnectionFactory= com.sun.xml.internal.messaging.saaj.client.p2p.HttpSOAPConnectionFactory -Djavax.xml.soap.MessageFactory= com.sun.xml.internal.messaging.saaj.soap.ver1_2.SOAPMessageFactory1_2Impl
```

## Tomcat

If you are using any polling listener that uses Quartz scheduler under the hood, you will need an implementation of JTAs User Transaction. All JEE containers have transaction support except for Tomcat. To resolve this issue, add the jta.jar to your server's lib directory.

## Weblogic

If you are using the File Upload Widget, the default implementation of JAX-WS/SAAJ has several performance issues. To resolve this issue, apply the following JVM parameters:

```
-Djavax.xml.soap.MetaFactory= com.sun.xml.internal.messaging.saaj.soap.SAAJMetaFactoryImpl -Djavax.xml.soap.SOAPConnectionFactory= com.sun.xml.internal.messaging.saaj.client.p2p.HttpSOAPConnectionFactory -Djavax.xml.soap.MessageFactory= com.sun.xml.internal.messaging.saaj.soap.ver1_2.SOAPMessageFactory1_2Impl
```

In addition to the above change, you need to set the class loader policy to prefer Sun JARs. This is achieved by including the weblogic.xml file in the WEB-INF directory of the WAR/EAR.

If you continue to have problems, you may need to add the latest SAAJ jars to WEB-INF/lib and remove 'internal' from the JVM parameters above.

## JBOSS

Out of the box, JBoss does not provide class loader isolation. AgencyPortal was getting transaction manager class created by Connect5, which means any class level initialization specific to Connect5 was used by AgencyPortal. A jboss-classloading.xml and joboss-web.xml files were added to the web app so that it gets its own domain and that solved the issue.

Add the following JARS to endorsed lib in JBoss:

```
lib/endorsed/jbossws-native-jaxrpc.jar lib/endorsed/jbossws-native-jaxws-ext.jar lib/endorsed/jbossws-native-jaxws.jar lib/endorsed/jbossws-native-saaj.jar
```

Added the attribute in red below to /jboss-5.1.0.GA/server/default/conf/bootstrap/Profile.xml; otherwise, JBoss won't deploy our apps in the JDK 1.6:

```
<bean name="AttachmentStore" class="org.jboss.system.server.profileservice.repository.AbstractAttachmentStore"><constructor><parameter class="java.io.File" ><inject bean="BootstrapProfileFactory" property="attachmentStoreRoot" /></parameter></constructor> <property name="mainDeployer"><inject bean="MainDeployer" /></property> <property name="serializer"><inject bean="AttachmentsSerializer" /></property> <property name="persistenceFactory"><inject bean="PersistenceFactory" /></property></bean>
```

# Connect5 Eclipse Setup

This topic outlines the step needed to set up a deployable (Connect5) web project, install integration kits and a Java project to build and debug custom modules (processor/listener/transport).

We can debug modules by deploying them to the server, but it becomes cumbersome when running WebLogic or WebSphere. It is recommended that developers create and test modules using the eiConsole Test simulator/test route feature before deploying it to the servlet container.

## Web Project (Required)

1. Download the Connect5 WAR from the nightly folder and create a web project by importing the war file.
2. There is a sample Ant build file and application.xml file (in case you are building an EAR file) provided.
3. The build.xml file expects the web root to be Web Root and source folder to be src. It also requires servlet-api.jar.
  - Connect5 requires a systemproperty called "env" to load the appropriate property files. The deployment descriptor can be modified so that the initialization servlet can look for a different system property.
  - Connect5 requires a data source called "aplatform" by default - the DDL required to create tables can be found in the nightly build folder.

## Integration Kits (Optional)

1. Download the appropriate integration kits from your FTP site.
2. Extract the zipped C5 Interfaces file to the /WEB-INF/interface-root/interfaces folder.
3. Add the appropriate properties to /WEB-INF/properties/<env>/interface.properties.

## Module Project (Optional)

The primary purpose of this project is to create and debug custom Java modules.

This project depends on the libraries that are in the PilotFish eiConsole installation directory, so PilotFish eiConsole should be installed before creating this project. If you haven't installed it yet, run the appropriate DMG or EXE files.

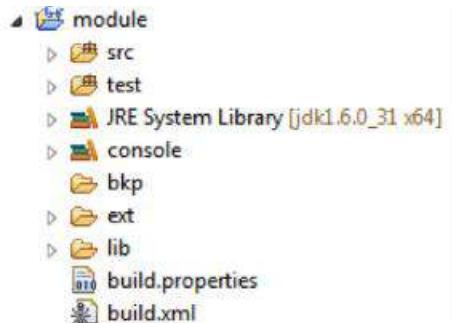
Create a Java project within Eclipse and then add a user library for the jars that are in /eiConsole/runtime and /eiConsole/runtime/lib folders.

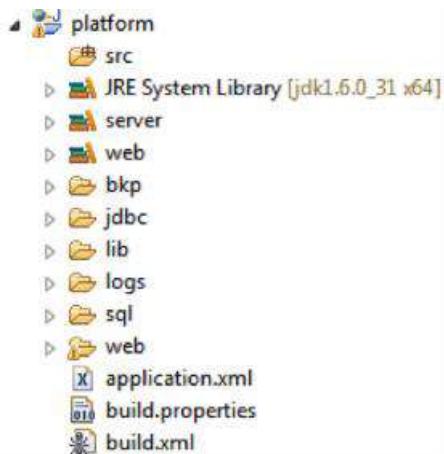
Steps to manually create user library:

1. When creating the user library, make sure to use "i4jruntime.jar" from eiConsole/.install4j directory.
  - Add the user library to the project build path.
  - Add "apmodule.jar" and "apsubstrate.jar" to the project build path.
2. After the project setup is complete, you can run eiConsole from Eclipse by executing the "com.PilotFish.eip.gui.console.Console" main class.
3. You can now set breakpoints in your custommodules and hit them when running eiConsole in Test simulator/Test route mode.
4. If you make any changes to the contents of "environment-settings.conf" (found in the root folder of your route configuration), they must be ported to "interface.properties" (found in the WEB-INF folder of your web project).

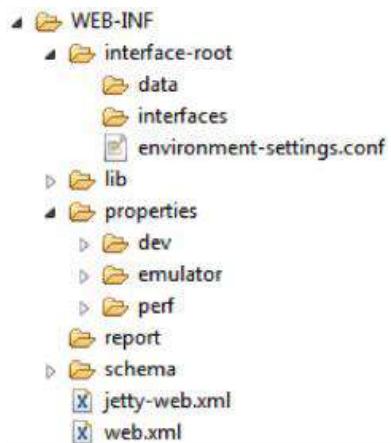
## Outcome

Your workspace setup is now complete and should look something like the following Package Explorer view:





The following is a sample of the WEB-INF folder and its contents. Pay special attention to the interface-root and properties folders:



# Connect5 Report Configuration

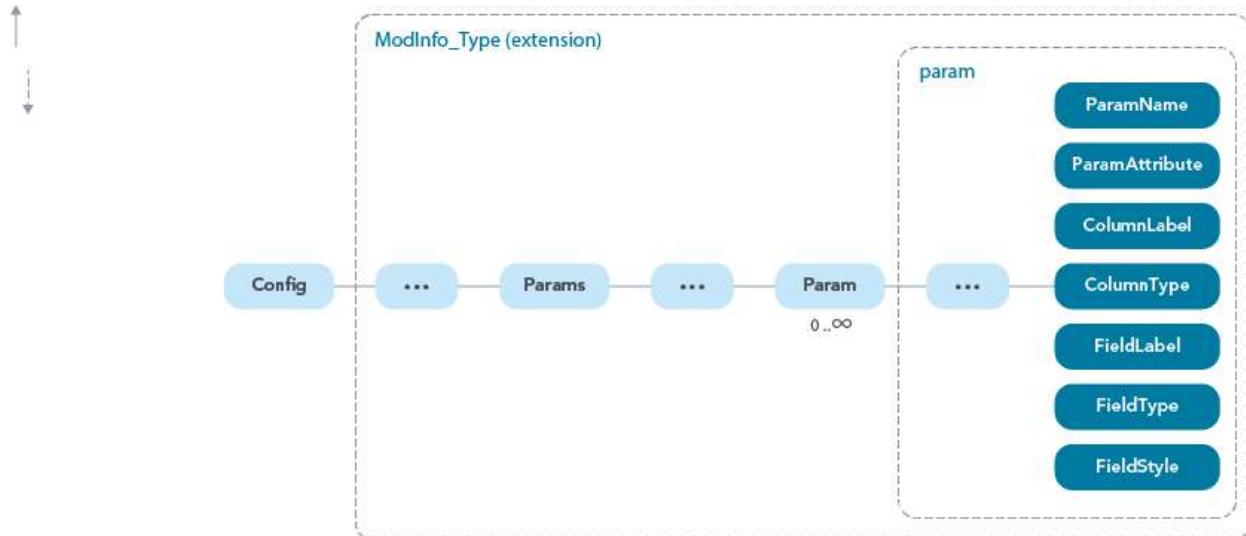
This topic details how to configure reports for your Connect5 project implementation.

## XML Report Parameter Model

The configurations for the dynamic report parameters are stored in an XML file. The following is a partial sample:

```
<Config>
 <Params>
 ...
 <Param>
 <ParamName>LOB</ParamName>
 <ParamAttribute>com.agencyport.lob</ParamAttribute>
 <ColumnLabel>LOB</ColumnLabel>
 <ColumnType>message</ColumnType>
 <FieldLabel>LOB</FieldLabel>
 <FieldType>checkbox</FieldType>
 <FieldStyle>triple</FieldStyle>
 </Param>
 ...
 </Params>
</Config>
```

The actual schema looks like the following (available in /web-inf/schema):



## Report Properties

The following properties are used by the application:

Property	Description	Default
report_param.definition_location	Specifies location of Param XML configuration file.	\${context_path}/WEB-INF/report/report_param.xml

## Web Reports

### HTML Report

The following is an HTML report sample.

- The columns in red are dynamic.
- Only the parameters with `Param.ColumnType` of "message" display.

- The Param.ColumnLabel is used as the header

Advanced Search

Previous | Rows: 0 to 50 | Next

Msg ID	Interface	Work Item	LOB	Vendor	Biz. Proc.	File Type	Request Time	Response Time	Duration	Complete	Errors
e1aafdf26-8cbf-455c-9e6e-d0c48fd5474e	ROOT	7895	CGL	Hawksoft	Upload Bridging	AL3	04/17/2013 02:53:24 PM	04/17/2013 02:54:32 PM	68	Yes	No
c1ad5a20-c7d9-4a20-9ff9-694c277c8c71	ROOT	7895	CGL	Hawksoft	Upload Bridging	AL3	04/17/2013 02:51:49 PM	04/17/2013 02:52:01 PM	11	Yes	No
33874868-30b0-4f9a-9efd-1bbb48763d34	ROOT	1234	CGL	Hawksoft	Upload Bridging	PDF	04/17/2013 02:50:02 PM	04/17/2013 02:50:15 PM	12	Yes	No
669af173-4177-4d99-bcfb-3653604b1cd7	ROOT		CGL	Hawksoft	Upload Bridging	PDF	04/16/2013 03:24:33 PM	04/16/2013 03:25:08 PM	34	Yes	No
083e4ea1-394c-46cf-b2a0-9692ae59463a	ROOT		CGL	Hawksoft	Upload Bridging	AL3	04/16/2013 03:24:32 PM	04/16/2013 03:25:06 PM	33	Yes	No
62297d49-08e5-45d9-9fbh-9654b7cb0538	ROOT		AUTOB	Applied	Upload Bridging	XML	04/16/2013 03:24:26 PM	04/16/2013 03:24:55 PM	28	Yes	No
a65f9ba9-c7a0-410c-9e87-d3d293d52fb8	ROOT		CGL	Hawksoft	Upload Bridging	AL3	04/16/2013 03:23:22 PM	04/16/2013 03:23:45 PM	23	Yes	No
1d1fa1d5-4411-4528-b15a-3bd727fe465	ROOT		AUTOB	Applied	Upload Bridging	XML	04/16/2013 03:23:15 PM	04/16/2013 03:23:39 PM	23	Yes	No
67b154c5-1c38-45a0-88b7-d542acf50a02c	ROOT		CGL	Hawksoft	Upload Bridging	PDF	04/16/2013 03:23:11 PM	04/16/2013 03:23:33 PM	21	Yes	No
11664c00-9692-453e-84c8-19fd5caf01	ROOT		CGL	Hawksoft	Upload Bridging	AL3	04/16/2013 03:22:55 PM	04/16/2013 03:23:06 PM	10	Yes	No

#### Report Parameters

## HTML Report Search

The following is a sample of a message search.

- Fields in red are dynamic.
- Only the parameters with a Param.FieldType of "text" or "checkbox" display.
- The Param.FieldName is used as the header and Param.FieldStyle is used to determine the number of columns.
- The values are generated from distinct values in the database.

Advanced Search

Previous | Rows: 0 to 50 | Next

LOB	Vendor	Biz. Proc.	File Type	Request Time
CGL	Hawksoft	Upload Bridging	AL3	04/17/2013 02:53:24 PM
CGL	Hawksoft	Upload Bridging	AL3	04/17/2013 02:51:49 PM
CGL	Hawksoft	Upload Bridging	PDF	04/17/2013 02:50:02 PM
CGL	Hawksoft	Upload Bridging	PDF	04/16/2013 03:24:33 PM
CGL	Hawksoft	Upload Bridging	AL3	04/16/2013 03:24:32 PM
AUTOB	Applied	Upload Bridging	XML	04/16/2013 03:24:26 PM
CGL	Hawksoft	Upload Bridging	AL3	04/16/2013 03:23:22 PM
AUTOB	Applied	Upload Bridging	XML	04/16/2013 03:23:15 PM
CGL	Hawksoft	Upload Bridging	PDF	04/16/2013 03:23:11 PM
CGL	Hawksoft	Upload Bridging	AL3	04/16/2013 03:22:55 PM

**Report Parameters**

**Work Item:**

**LOB:**  
 AUTOB    AUTOB    BOP  
 CGL    CPKG    HOME

**Vendor:**  
 AMS360    Hawksoft    Applied

**Business Process:**  
 Upload Bridging    Real Time Rating  
 Service Call

**File Type:**  
 AL3    PDF    XML

**Request Date Between:**  
 -

**Response Date Between:**  
 -

Completed    Incomplete  
 Errors    No Errors

## Runtime Logging

### Database Logging Model

The following table gets updated with the transaction data for each parameter. The NAME column is populated from the Param.ParamName field and the VALUE column is populated using the Param.ParamAttribute field to extract data from transaction attributes.

PF_EIP_PARAMLOG		
ID	INTEGER	NOT NULL
MESSAGEID	INTEGER	NOT NULL
TRANSACTIONID	INTEGER	NOT NULL
NAME	VARCHAR(10)	NOT NULL
VALUE	VARCHAR(10)	

## XML Route Configuration

The attributes referenced above need to be populated within your route.xml file. The following is a sample:

```
<Processor class="com.agencyport.substrate.module.APXPathAttributeProcessor" name="Bridging XPath Attributes"><ModuleConfig>
<ExecuteProcessor>true</ExecuteProcessor> <AttributeScope>Transaction</AttributeScope><XPathExpressions>
[eip_pair:com.agencyport.producer:eip_name://Producer/ProducerInfo/ContractNumber:eip_value]
[eip_pair:com.agencyport.insured:eip_name://InsuredOrPrincipalInfo/InsuredOrPrincipalRoleCd=Insured]/GeneralPartyInfo
/NameInfo/CommName/CommercialName:eip_value] [eip_pair:com.agencyport.lob:eip_name://CommPolicy/LOBCd:eip_value]
[eip_pair:com.agencyport.state:eip_name://CommPolicy/ControllingStateProvCd:eip_value]
[eip_pair:com.agencyport.premium:eip_name://CommPolicy/CurrentTermAmt/Amt:eip_value] </XPathExpressions> <Namespaces />
</ModuleConfig> </Processor>
```

## Logging Properties

The following properties are used by the application:

Property	Description	Default
com.pilotfish.eip.enableParamReporting	Enables logging to the ParamLog database table.	true

# Connect5 Database Definitions

This topic details the tables that are in place as part of the Connect5 implementation.

Not all tables are used by all implementations.

## Note

The diagrams included in this topic were generated from a Microsoft SQL Server instance, so note that some data types may not match exactly with your implementation.

Tables that are required by the framework start with "PF"; tables that were added for specific module (e.g., WebService Security) start with "AP"; carrier-specific tables should follow table naming conventions as set forth by the carrier.

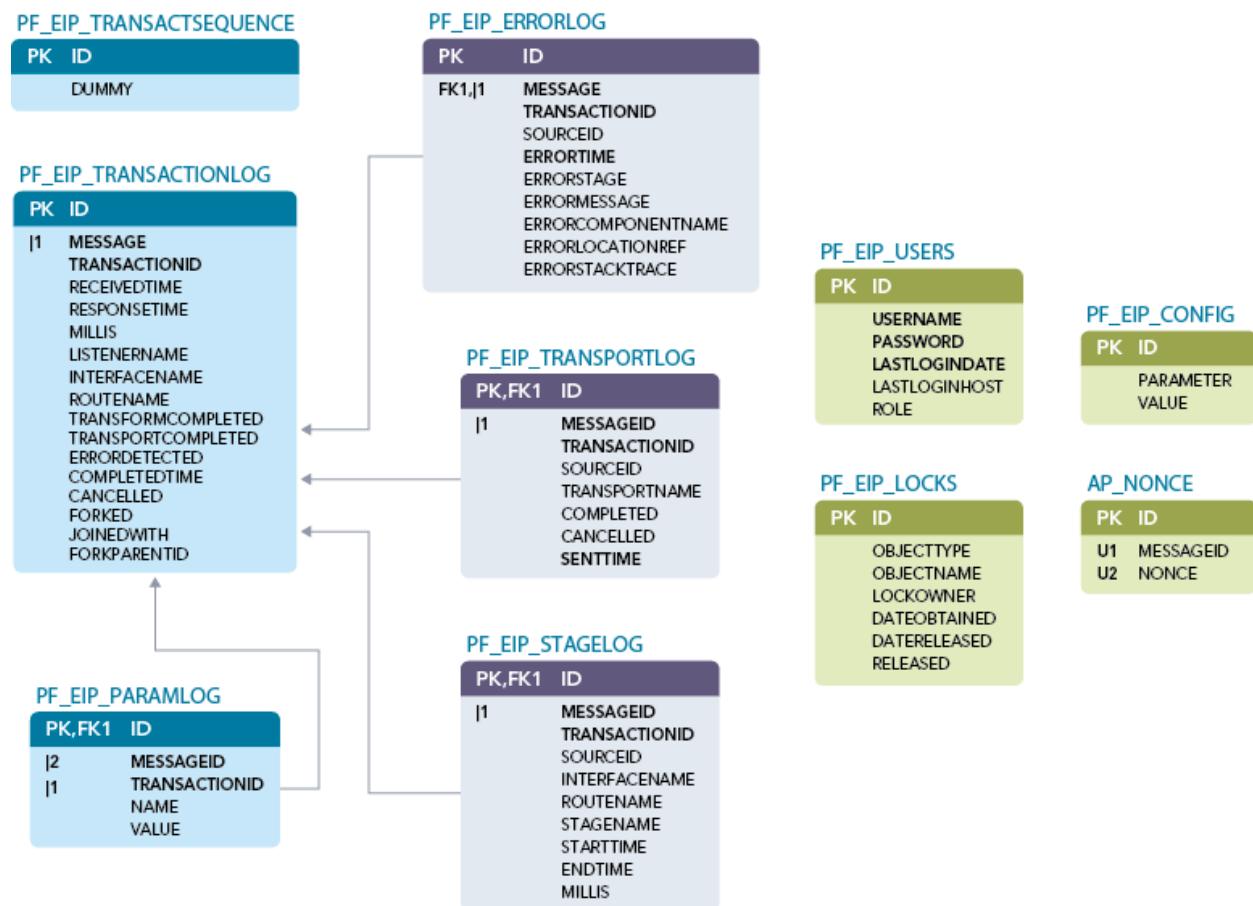
## General Technology

- Interface - an interface is a series of routes designed to represent a business process. Sometimes, interfaces are "named," whereas other times they are the "ROOT" interface (though this is discouraged).
  - Message - a message is an invocation of an interface.
- Route - a route is a dedicated business unit of work in an interface for the purpose of making a service call, or acting as a self-contained unit of work.
  - Transaction - a transaction is an invocation of a route.
- Format - contains all transformation logic and artifacts for inbound/outbound services.
- Module - a configurable component that is one of the following types:
  - Listener - entry point of a route (source).
  - Processor - a processor is an ad-hoc step within the route.
  - Transport - exit point of a route (target).
    - Stage - a stage is an invocation of a module that is part of a route.

## Example

When running an XSLT that's a straight "processor" stage, but when making a call out to HTTP or another route that would be a "transport" stage.

## SDK Tables



### PF\_EIP\_TRANSPORTLOG

The PF\_EIP\_TRANSPORTLOG table is responsible for capturing pertinent information about the transport invocations that happen as part of an interface execution.

Column Name	Data Type	Description
ID	int	Primary key identifier.
MESSAGEID	varchar(128)	ID of message that executed transaction.
TRANSACTIONID	varchar(128)	ID of transaction that executed transport.
SOURCEID	varcahr(128)	Origination transaction ID in the event that a fork happened, which would end up causing the original transaction ID to be lost.
TRANSPORTNAME	varchar(128)	Name of transport that was executed.
COMPLETED	int	Indicator that transport completed.
CANCELLED	int	Indicator that transport was cancelled.
SETTIME	datetime	Time transport was invoked.

## PF\_EIP\_TRANSACTSEQUENCE

The PF\_EIP\_PARAMLOG table is responsible for storing the parameters that the application determines are worth storing for the purpose of displaying those fields back as a part of the admin console. The fields that make it to this table are set up in the report\_param.xml file.

Column Name	Data Type	Description
ID	int	Primary key identifier.
MESSAGEID	varchar(128)	ID of message that executed transaction.
TRANSACTIONID	varchar(128)	ID of transaction that was last to update the parameter.
NAME	varchar(128)	Parameter name.
VALUE	varchar(128)	Parameter value.

## PF\_EIP\_STAGELOG

The PF\_EIP\_STAGELOG table is responsible for capturing the relevant information from the execution of a processor.

Column Name	Data Type	Description
ID	int	Primary key identifier.
MESSAGEID	varchar(128)	ID of message that executed transaction.
TRANSACTIONID	varchar(128)	ID of transaction that executed stage.
SOURCEID	varchar(128)	Original transaction ID in the event that a fork happened, which would end up causing the original transaction ID to be lost.
INTERFACENAME	varchar(128)	Name of interface that owns route.
ROUTENAME	varchar(128)	Name of route that owns stage.
STAGENAME	varchar(128)	Name of stage.
STARTTIME	datetime	Start time.
ENDTIME	datetime	End time.
MILLIS	int	Duration in milliseconds.

## PF\_EIP\_TRANSACTIONLOG

The PF\_EIP\_TRANSACTIONLOG table is responsible for auditing the execution of the routes in the interfaces.

Column Name	Data Type	Description
ID	int	Primary key identifier.
MESSAGEID	varchar(128)	ID of message that executed transaction.
TRANSACTIONID	varchar(128)	ID of transaction- logically this is the primary key.

Column Name	Data Type	Description
RECEIVEDTIME	datetime	Start time.
RESPONSETIME	datetime	End time.
MILLIS	int	Duration in milliseconds.
LISTENERNAME	varchar(128)	Name of listener on route that was used to "enter" the route.
INTERFACENAME	varchar(128)	Name of interface that owns route.
ROUTENAME	int	Name of route that was executed.
TRANSFORMCOMPLETED	int	Indicator that transformation was completed.
TRANSPORTCOMPLETED	int	Indicator that transport was completed.
ERRORDETECTED	int	Indicator that an error occurred.
COMPLETEDTIME	datetime	Completion time.
CANCELLED	int	Indicator whether transaction was cancelled.
FORKED	int	Indicator whether this transaction was forked.
JOINEDWITH	varchar(128)	If transaction was joined, the name of the transaction with which it was joined.
FORKPARENTID	varchar(128)	If this transaction was forked, the ID of the parent transaction.

## PF\_EIP\_USERS

The PF\_EIP\_USERS table is responsible for managing user information.

Column Name	Data Type	Description
ID	int	Primary key identifier.
USERNAME	varchar(128)	User's login.
PASSWORD	varchar(128)	User's password.
LASTLOGINDATE	datetime	Last login timestamp.
LASTLOGINHOST	varchar(128)	Host form which last login occurred.
ROLE	int	Role identifier.

## PF\_EIP\_CONFIG

The PF\_EIP\_CONFIG table is responsible for the product's internal configuration. This table serves as an extension point to the application's configuration.

Column Name	Data Type	Description
ID	int	Primary key identifier.
PARAMETER	varchar(128)	Application configuration property.
VALUE	varchar(128)	Application configuration value.

## AP\_NONCE

The AP\_NONCE table is responsible for storing unique Nonces as they're associated to a interface invocation.

Column Name	Data Type	Description
ID	int	Primary key identifier.
MESSAGEID	varchar(128)	ID of the message for which the NONCE is being stored.
NONCE	varchar(128)	Unique NONCE associated with message.

## PF\_EIP\_ERRORLOG

The PF\_EIP\_ERRORLOG table is responsible for capturing information around error generation.

Column Name	Data Type	Description
ID	int	Primary key identifier.
MESSAGEID	varchar(128)	ID of message that executed transaction.
TRANSACTIONID	varchar(128)	ID of transaction that executed stage (if applicable).
SOURCEID	varchar(128)	Original transaction ID in the event that a fork happened, which would end up causing the original transaction ID to be lost.
ERRORTIME	datetime	Timestamp of when error occurred.
ERRORSTAGE	varchar(128)	Type of activity that was happening when error occurred (e.g., transport or processor).
ERRORMESSAGE	varchar(128)	Specific error message.
ERRORCOMPONENTNAME	varchar(128)	Specific error message.
ERRORLOCATIONREF	varchar(128)	A combination of ERRORSTAGE and ERRORCOMPONENTNAME.
ERRORSTACKTRACE	varchar(4000)	Entire stack of the error.

## PF\_EIP\_LOCKS

The PF\_EIP\_LOCKS table is responsible for managing the dedicated locks that are needed by the application.

Column Name	Data Type	Description
ID	int	Primary key identifier.

<b>Column Name</b>	<b>Data Type</b>	<b>Description</b>
OBJECTTYPE	varchar(128)	Type of object that owns the lock.
OBJECTNAME	varchar(128)	Name of the object that owns the lock.
LOCKOWNER	varchar(128)	Owner of the lock.
DATEOBTAINED	datetime	Timestamp of when the lock was acquired.
DATERELEASED	datetime	Timestamp of when the lock was released.
RELEASED	int	Indicator as to whether the lock was released.

# Connect5 Deployment Configuration

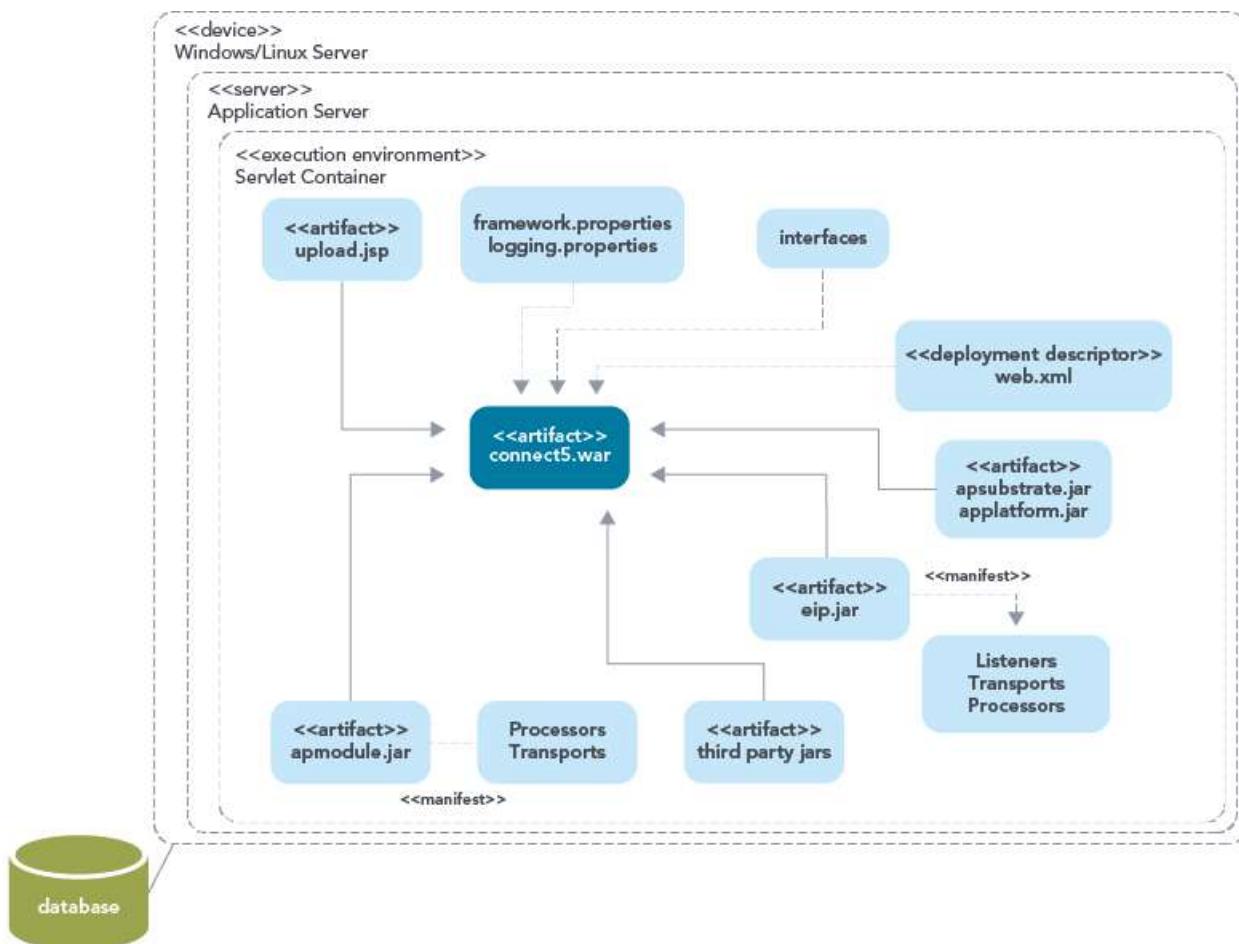
This topic outlines the artifacts that are produced during the Connect5 product build and how those artifacts should be used to set up projects within an Eclipse environment by the project team.

## Build Artifacts

The following are the build artifacts produced:

- apmodule.jar and source - These are the modules developed by Agencyport. Modules are descendent of the EIPModule interface.
- apsubstrate.jar and source - These are Java classes developed by Agencyport used by modules.
- Connect5.war and Connect5.ear - Deployable unit in a servlet container; referred to as 'Platform.'
- applatform.jar and source - Has initialization servlet and other Java classes used by Connect5.war/Connect5.ear.
- sql - SQL files to create tables used by platform authentication module.
- eip-root - Working directory that contains routes, formats, XSLT. These artifacts should be used as samples while devising solutions. At this time it's not advised to reuse them as is. The Connect5.war/Connect5.ear file has this.

## Deployment Diagram



## Properties

Properties are grouped into the following six files:

- `framework.properties`
- `pilotfish.properties`

- logging.properties
- interface.properties
- message.properties
- upload.properties

## framework.properties

The framework.properties file is specific to an environment. The application loads the appropriate properties file from the following location: \${context\_path}/WEB-INF/properties/<env>/framework.properties. The <env> is obtained from the 'env' systemJVM property.

The major properties control database, logging configuration and additional properties to load are:

Name	Description
database_agent_class_name	Class name of the AP Database Agent
datasource	JNDI URL of datasource
application.logging_config_file	The location of logging property file
additional_properties_to_load	The additional properties to load at bootstrap.

The following are the default values:

```
database_agent_class_name=com.agencyport.database.MySQLDatabaseAgent
datasource=java:comp/env/applatform
application.logging_config_file=${context_path}/WEB-INF/properties/<env>/logging.properties
additional_properties_to_load=${context_path}/WEB-INF/properties/<env>/pilotfish.properties; \
${context_path}/WEB-INF/properties/<env>/message.properties; \
${context_path}/WEB-INF/properties/<env>/upload.properties; \
${context_path}/WEB-INF/properties/<env>/interface.properties;
```

## pilotfish.properties

The pilotfish.properties file configures the PilotFish framework's logging, auditing, threading, etc.

The most important property is the location for interface root (eip-root) and relative path to interface properties.

Name	Description
com.pilotfish.eip.configDirectory	The location of eip-root directory.
com.pilotfish.eip.environmentSettings.file	The location of the interface setting file. The path is relative to the eip-root directory. By default, it is set to use the same framework property file.

The following are the default values:

```
com.pilotfish.eip.configDirectory=${context_path}/WEB-INF/interface-root/
com.pilotfish.eip.environmentSettings.file=../properties/<env>/interface.properties
```

Name	Description
com.pilotfish.eip.enableStageReporting	When set to true, stage processing time is recorded to database.
com.pilotfish.eip.enableTransactionReporting	When set to true, key transaction information is recorded to the database.
com.pilotfish.eip.enableParamReporting	When set to true, configured report parameters are recorded to the database.
com.pilotfish.eip.enableMessageLogging	When set to true, message data is written to the file system before and after exiting a route.
com.pilotfish.eip.messageLoggingDirectory	The location where message data will be logged.

Name	Description
com.pilotfish.eip.enableAttributeLogging	When set to true, message attributes are written to the file system before and after exiting a route.
com.pilotfish.eip.attributeLoggingDirectory	The location where message attributes will be logged.
com.pilotfish.eip.enableStageLogging	When set to true, stage data is written to the file system before and after exiting a route.
com.pilotfish.eip.stageLoggingDirectory	The location where stage data will be logged.

The following are the default values:

```
com.pilotfish.eip.enableTransactionReporting=true com.pilotfish.eip.enableStageReporting=false com.pilotfish.eip.enableParamReporting=true
com.pilotfish.eip.enableAttributeLogging=true com.pilotfish.eip.enableMessageLogging=true com.pilotfish.eip.enableStageLogging=false
com.pilotfish.eip.messageLoggingDirectory=${output_dir}/messages com.pilotfish.eip.attributeLoggingDirectory=${output_dir}/attributes
com.pilotfish.eip.stageLoggingDirectory=${output_dir}/stages
```

#### Note

'Stage' logging is turned OFF by default. This setting is useful for debugging, but should only be used locally as it is very resource intensive.

#### logging.properties

Platform uses JUL for logging. Logging level /loggers/filehandlers can be configured in the logging.properties file.

The following is a sample logging file:

```
com.agencyport.portal.handlers=com.agencyport.logging.PortalFileHandler,java.util.logging.ConsoleHandler,com.agencyport.logging.syslog.SyslogHandler com.agencyport.portal.loggers=com.pilotfish
com.agencyport.logging.PortalFileHandler.pattern=${output_log_dir}/framework%g.log
com.agencyport.logging.PortalFileHandler.limit=20000000 com.agencyport.logging.PortalFileHandler.count=3
com.agencyport.logging.PortalFileHandler.formatter=com.agencyport.logging.DefaultLogFormatter
com.agencyport.logging.PortalFileHandler.append=true com.agencyport.logging.DefaultLogFormatter.timestamp.format=yyyy-MM-dd
HH:mm:ss:SSSSS %z com.agencyport.logging.syslog.SyslogHandler.formatter=com.agencyport.logging.syslog.SyslogFormatter
com.agencyport.logging.syslog.SyslogHandler.level=SEVERE com.agencyport.logging.syslog.SyslogHandler.host=127.0.0.1
com.agencyport.logging.syslog.SyslogHandler.port=514 java.util.logging.ConsoleHandler.level=SEVERE
java.util.logging.ConsoleHandler.formatter=com.agencyport.logging.DefaultLogFormatter compilotfish.useParentHandlers=false
com.pilotfish.eip.transact.level=SEVERE com.pilotfish.eip.server.level=SEVERE com.pilotfish.eip.cache.level=SEVERE
com.pilotfish.eip.extend.level=SEVERE com.pilotfish.eip.admin.level=SEVERE com.pilotfish.eip.config.level=SEVERE
com.pilotfish.eip.modules.level=SEVERE
```

#### interface.properties

This file stores all environment properties used in your routes and interfaces (e.g., URLs). Its design time equivalent is the environment-settings.conf file.

Currently, these files have to be kept in sync manually. Note that the contents of the .conf file are escaped (e.g., http:\:) whereas .properties are not. Lastly, the use of \${property} referenced variables is not allowed in the .conf file.

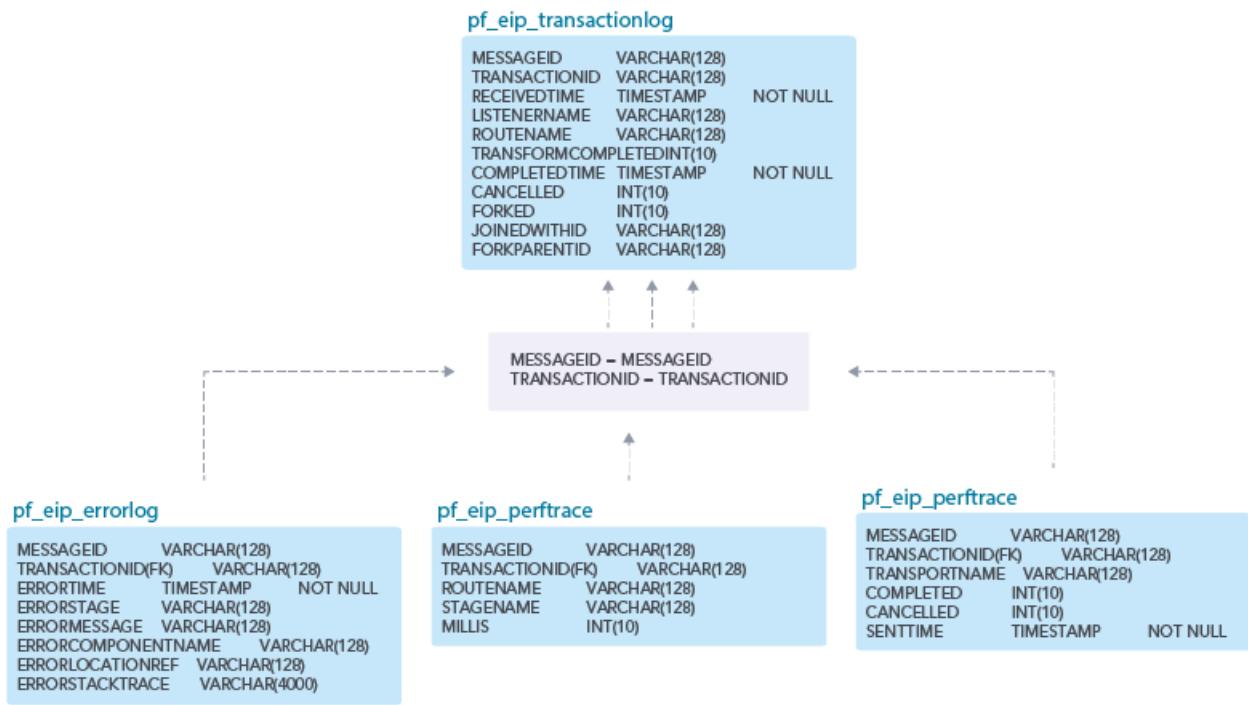
#### message.properties

This file is used much like a resource bundle without the localized content. It houses all user facing messages for easy editing.

This file is used only by the File Selector Widget.

#### Database

Use the reload.sql file found with the sql folder to create tables. It has SQL statements to drops and creates tables used by platform. When creating the database for the first time, remove the drop statements.



# Turnstile Integration

Many agents still attach ACORD forms as PDFs and email them to carriers to receive a quote. The carrier gets the quote by inputting the data by hand. This involves a burdensome, manual re-keying process of those forms for carriers to input the data into their agent portals and agent management systems, which can result in data errors, slow response time and high costs.

Agencyport's Turnstile provides a clean, simple and ultra-secure service that allows agents and carriers to take data from an ACORD form, transform it into XML and automatically import the information into AgencyPortal (or the carrier's agent portal). This functionality shortens intake time, accelerates decision making, reduces processing cost and eliminates lost opportunities.

As an additional benefit, no additional hardware is needed since Turnstile is hosted. This gives carriers access to superior data extraction capabilities directly within their own processes through a customer-unique code or API key. Just a simple to use, highly available and ultra-secure service that's implemented with AgencyPortal.

## History of Turnstile

AgencyConnect, an Agencyport product that uploaded data from agency management systems into an agent portal (i.e., it was the bridging upload tool between the agency management system and the carrier quoting system), was the beginning of what is now Turnstile. After speaking to agents on the Agencyport Agent Insight Tour in 2009, it was brought to our attention that agents weren't really using the upload feature in AgencyConnect as it was intended. Agents did not want to have to key in data multiple times and verify that each item uploaded the same in multiple portals; it was easier for them to simply email the PDF form to each of the carriers and have them input the data.

So that is when we decided to create a product that converts the ACORD form PDF data into XML, which could then be uploaded into AgencyPortal and make the process easier for everyone.

Agents and/or internal carrier staff can now simply upload an ACORD form into AgencyPortal.

## Integration with Connect5

Connect5 and Turnstile were built to go hand-in-hand. Connect5 is a single hub for agency interfaces, including inquiry, data bridging and comparative rating. When working with Turnstile and AgencyPortal, Connect5 receives data from Turnstile when an ACORD form is uploaded and automatically prefills the data into AgencyPortal. This, in other words, makes Connect5 the "middleman" who transports the form data to AgencyPortal.

## Turnstile Configuration

To enable Turnstile within your AgencyPortal application, you must enable it in the framework.properties file. The following is a list of configurable Turnstile integration properties to configure this feature:

Property	Description	Default Value	Recommended Setting (if any)
enable_turnstile_upload		false	
max_file_size	The maximum total filesize (in megabytes) allowed for a single upload to Turnstile via the Turnstile Integration Kit.	10	
APPEND_USER_CREDENTIALS_TO_RETURN_URL		false	
ENCRYPT_USER_CREDENTIALS		false	

## Using Turnstile with AgencyPortal

Turnstile's functionality within AgencyPortal is provided as a global and account level option, which allows agents and carrier staff to directly upload ACORD forms to create work items for new or existing accounts.

Refer to [Turnstile Forms](#) for a list of forms Turnstile accepts in PDF, TIFF or ZIP format.

## Note

Turnstile is only available for commercial lines of business. It is not available for personal lines of business.

Turnstile upload is used in one of the following ways:

- by carrier internal staff to upload PDF forms emailed from an agent
- by agents to upload PDF forms directly into portal

## Note

If there are any errors with the form, Connect5 returns these errors and displays them in AgencyPortal.

The work item workflow in AgencyPortal displays to enter any missing or correct any erroneous data to complete the item.

## PDF Spotlight

AgencyPortal provides the ability to quickly view an original PDF as a reference to determine whether the data uploaded into AgencyPortal correctly if it comes from a poor quality scanned form uploaded via Turnstile.

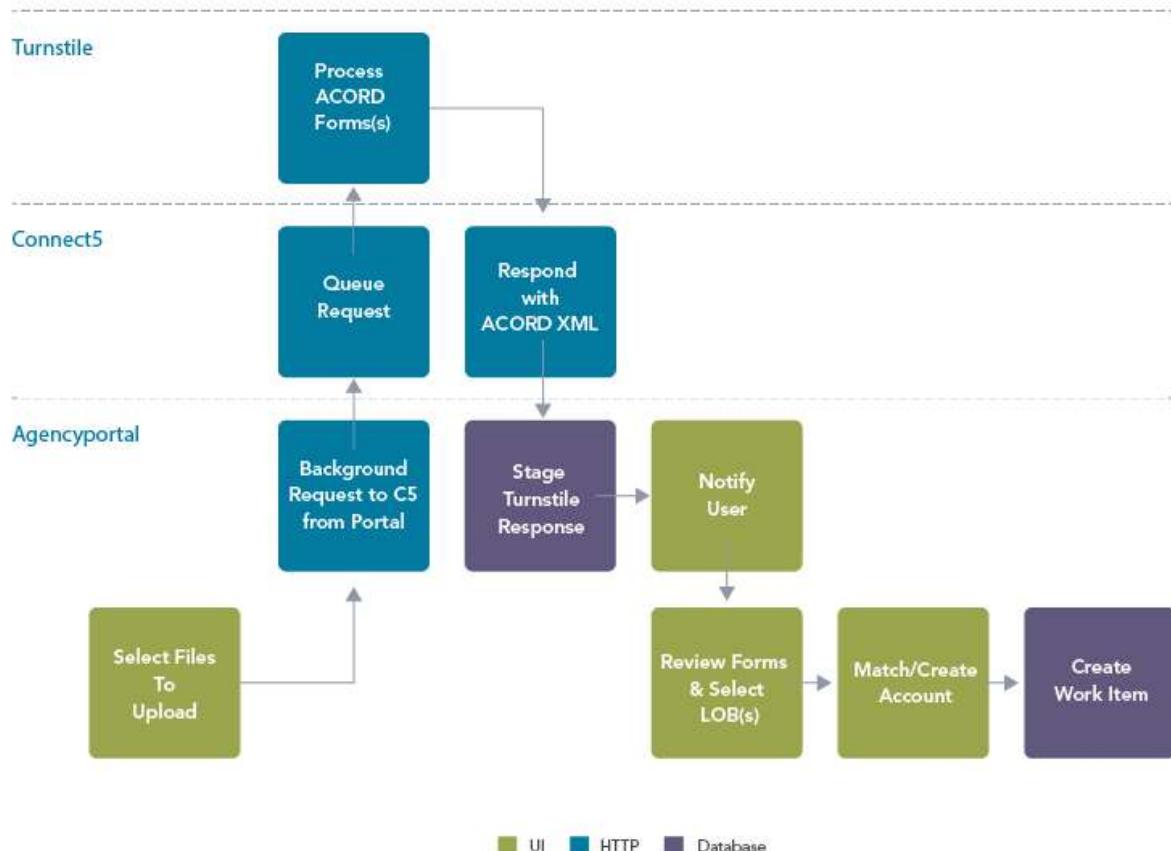
To use this feature within your application, the `enable_turnstile_snippets` property file in the `framework.properties` file must be set to true (the default value) and you must be set up to use Turnstile for your form uploads.

When a scanned PDF is uploaded, fields are identified based on document level confidence score or field level confidence score. The threshold of what is considered a low confidence score is configurable both at the field and document level within Turnstile. The rendering of the snippet image and the field's confidence score is set within Turnstile. Refer to the Turnstile team for more information on how to set this portion of the PDF Spotlight feature.

# Turnstile Technical Workflow

Turnstile allows you to upload completed ACORD forms into AgencyPortal to create new work items, add work items to existing accounts and add work items to new accounts.

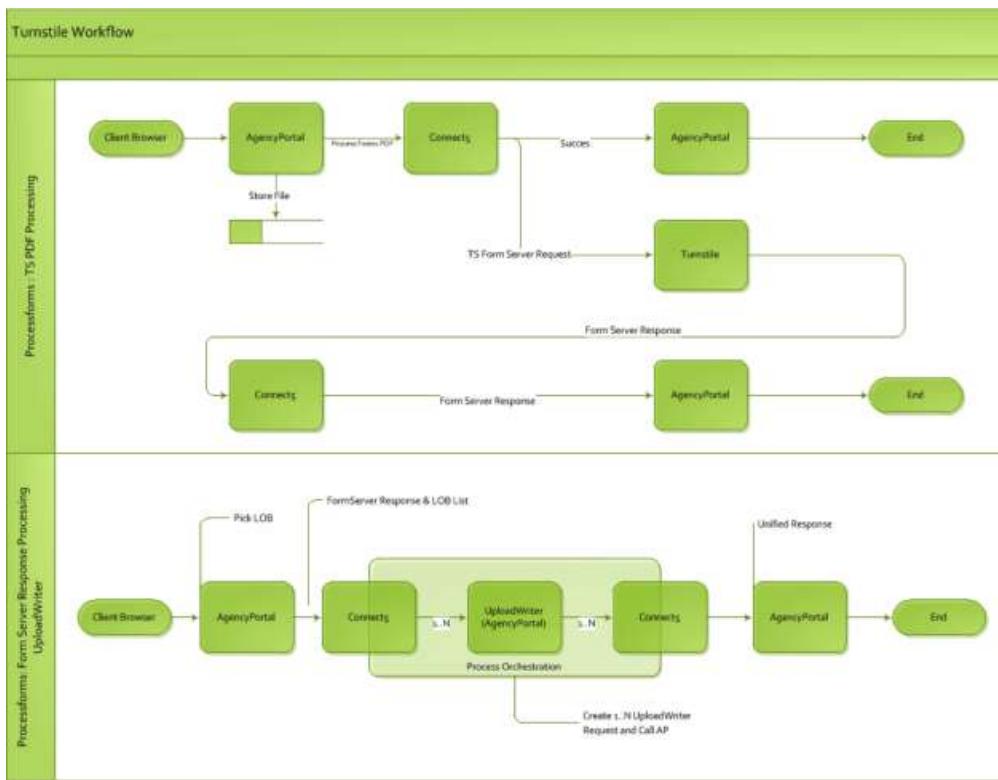
This simplified workflow is included with AgencyPortal, but how does it work?



## Note

A call to Turnstile for status updates is made when a user selects the Getting Started menu in AgencyPortal to view the status of uploaded items. Uploaded item statuses continue to update as long as the user has the upload window open. When the user closes out of the upload window, the call for updates ends.

## Turnstile PDF Processing



The following describes the steps of Turnstile PDF processing:

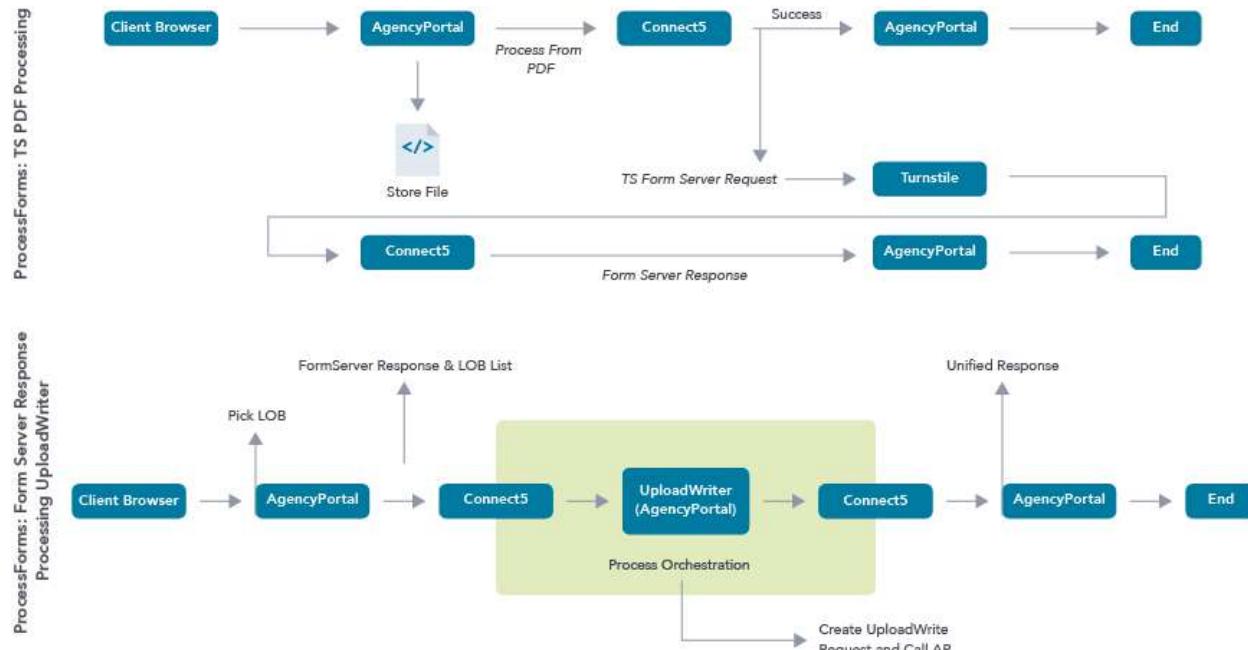
1. Connect5 receives PDF from AgencyPortal.
  1. Connect5 sends success message to AgencyPortal.
2. Connect5 sends PDF(s) to Turnstile.
3. Turnstile extracts data from PDF(s) and returns 'form server response' XML data to Connect5.
4. Connect5 sends the 'form server response' XML data to AgencyPortal.

## Form Sever Response Processing

The following describes the steps for form server response processing:

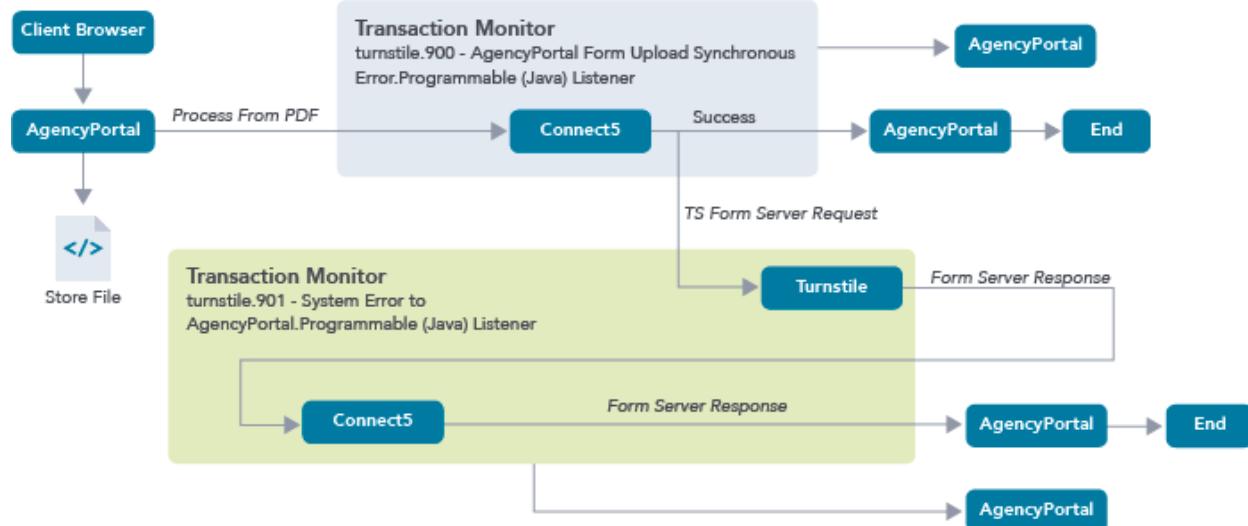
1. Connect5 receives 'form server response' and line of business to process.
2. Create line of business XML(s) from 'form server response'.
3. Connect5 normalizes the LOB XML and calls upload writer to create work item 1 to N work items.
4. Unifies the upload writer response and formulates the response AgencyPortal.
  1. Connect5 builds ACORD status and extend status message from AgencyConnect step data messages.

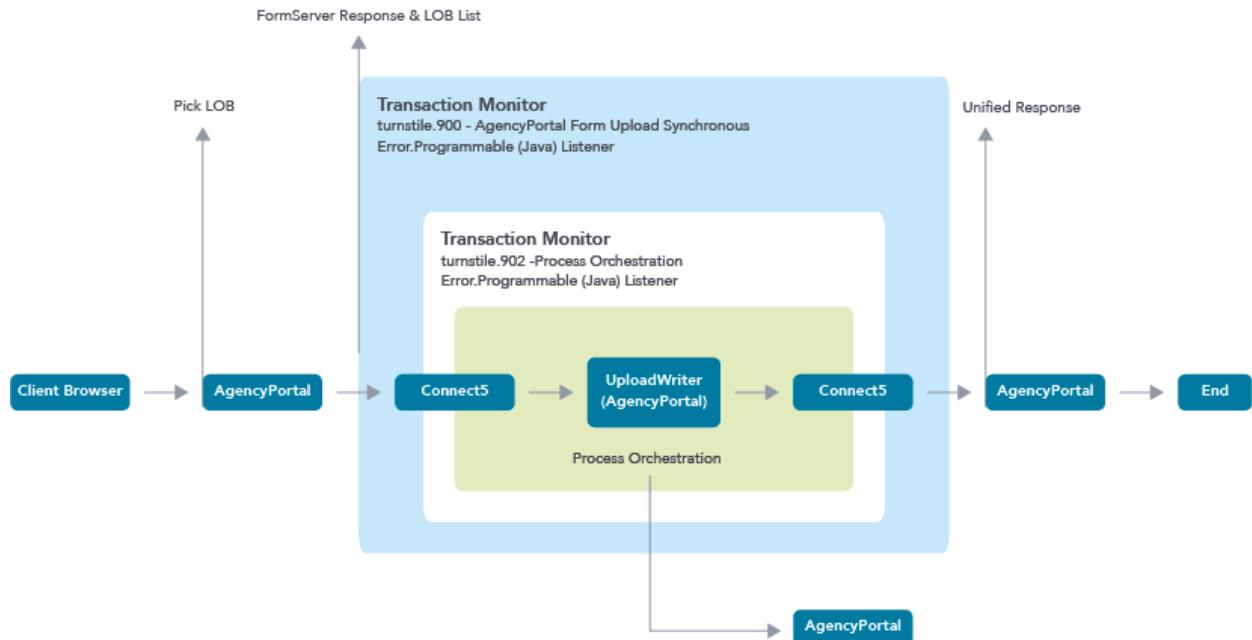
## Success



## Failure

There are three transaction monitors configured to trap an error condition and respond to AgencyPortal with an error message.

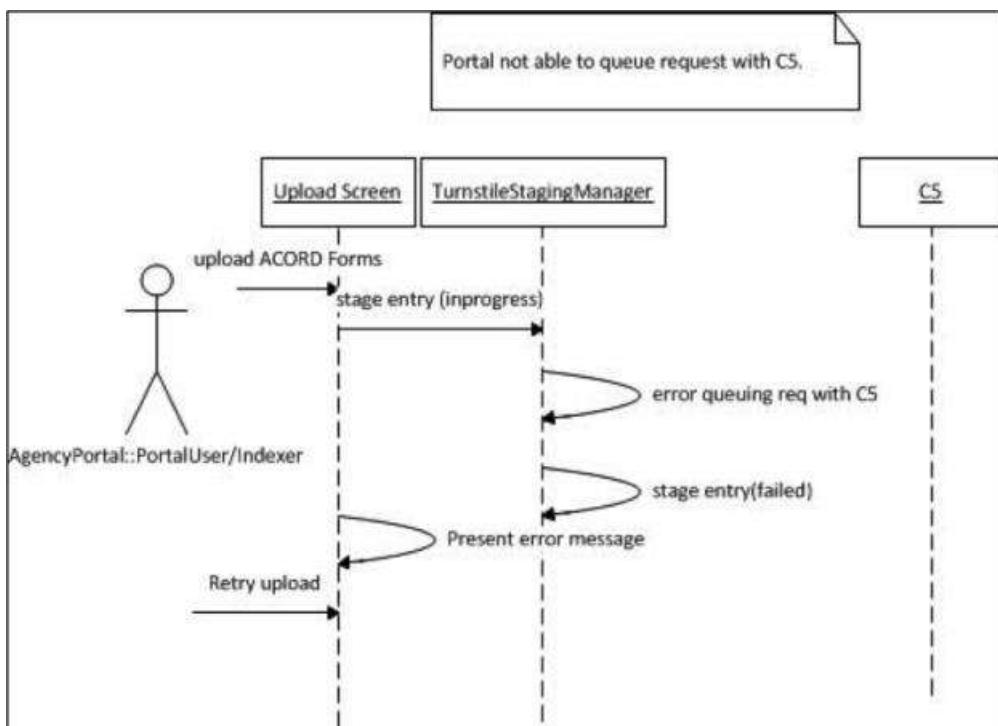




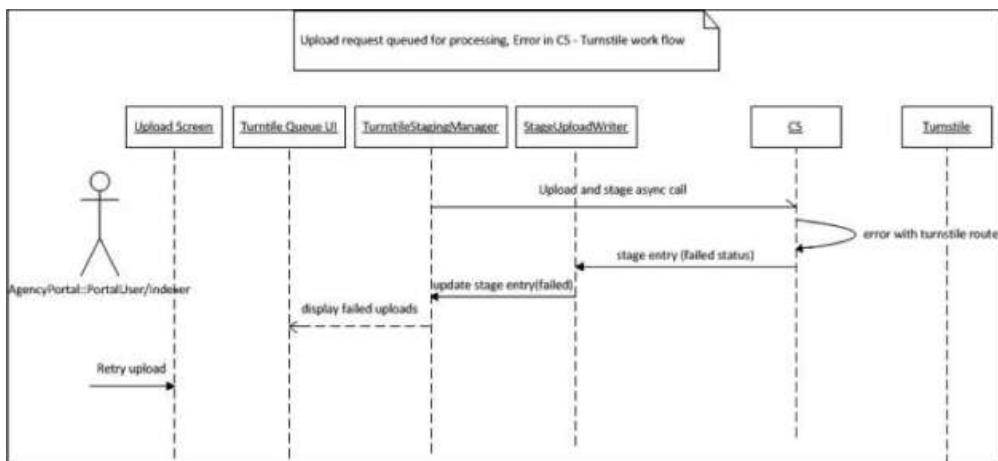
All validation and runtime errors are captured and routed to AgencyPortal as SOAP fault.

- Turnstile Unavailable (server down/unable to connect): *System Failure: We cannot process your message at this time. please try again later.*
- Invalid Turnstile Request (junk data/certificate error/authentication error): *System Failure: We cannot process your message at this time. Please try again later.*
- Generic/Unknown Error (generic AgencyPortal error message not listed above): *Invalid Attachments: Unable to process the attached forms. Please attach valid ACORD PDF forms and try again.*
- Cannot create LOB XMLs from 'Form Server Response'.
  - AgencyPortal Unavailable (server down/unable to connect): *System Failure: We cannot process your message at this time. Please try again later.*
  - Invalid Request (junk data/certificate error/authentication error): *System Failure: We cannot process your message at this time. Please try again later.*
  - Work Item not created (caused by junk data, invalid state, missing fields or relationships): *Invalid Upload: We could not create a valid application from your attachments.*
  - Invalid Work Item Status (rejected based on underwriter rules; declined due to clearance or referred to underwriter):
    - *Invalid Upload: We have rejected your application based on our business rules.*
    - *Invalid Upload: We have declined your application due to a clearance failure.*
    - *Invalid Upload: We have referred your application to an internal representative.*
  - Invalid Work Item Report (not enough fields were populated or too many fields are invalid):
    - *Invalid Upload: There was not enough data to create your application.*
    - *Invalid Upload: There was not enough valid data to create your application.*
  - Generic/Unknown Error (generic error message shown if an error that's not listed above): *Invalid Upload: We could not create a valid application from your attachments.*

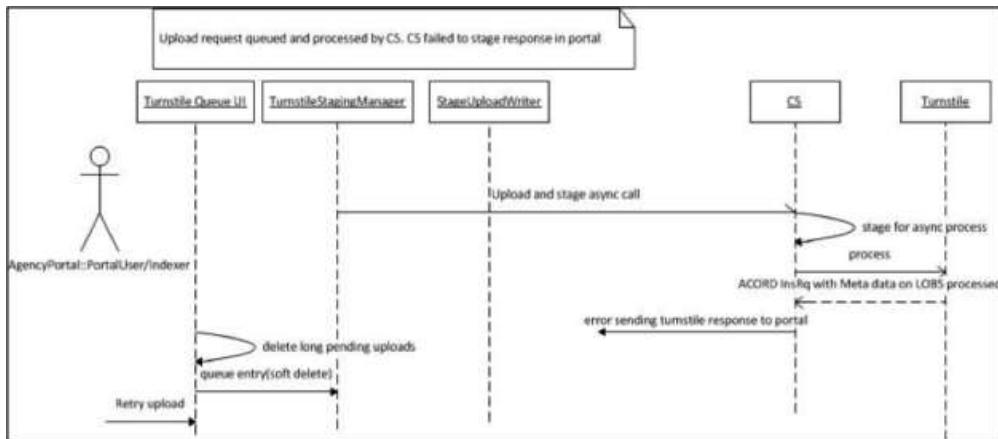
### Queuing Failure - Portal/C5



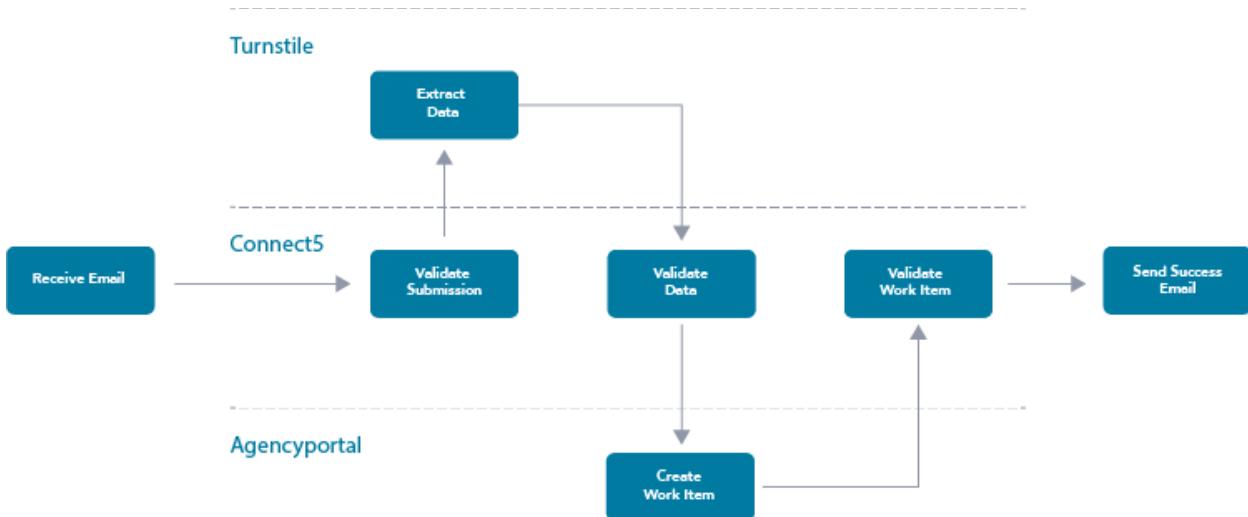
### Processing Failure – C5/Turnstile



## Staging Failure – C5/Portal



## Email



The following describes the steps of Turnstile email processing:

1. Connect5 receives the submission.
2. Connect5 validates the submission, then sends PDFs to Turnstile.
3. Turnstile extracts data from PDFs and returns data to Connect5.
4. Connect5 validates the data, transforms the data and sends to AgencyPortal.
5. AgencyPortal creates a work item and returns it to Connect5.
6. Connect5 validates the work item and deems it valid.
7. Connect5 sends a success email or redirects user automatically.

## Rules

The following rules are in place to send a response:

- Valid submission: [Success Message].
- Valid authentication: [Success Message].
- Some supported forms: [Success Message]+ some of the attached forms were excluded.
- Some supported LOBs: [Success Message]+ some of the lines-of-business are not valid.
- Some supported states: [Success Message]+ some of the states are not valid.
- Enough valid date to create work item: [Success Message].

[Success Message]= We successfully uploaded information from the following forms: <Form1>, <Form2>...

## Results

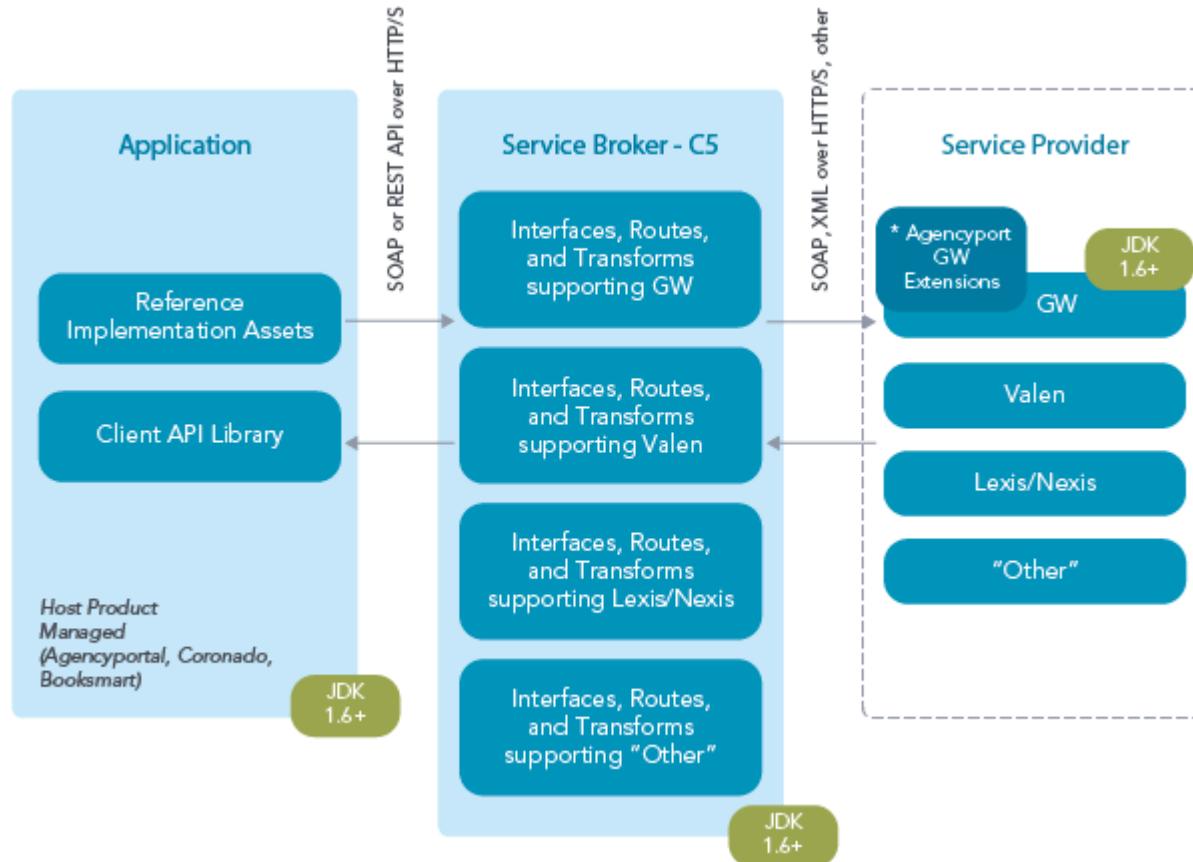
In the results message, link to work item emailed to user or work item is automatically opened. When opening the work item, appropriate messages display based on the rules above.

# Integration Kits

The integration kit initiative, which formalized and productized the notion of pre-built integrations that were previously positioned as 100% time and materials, was introduced in 2013. These pre-built integration kits are based on AgencyPortal templates with the out of the box functionality available from the service provider.

# Integration Kit Architecture

The following describes the integration kit architecture:



## Application Tier

### Client API Library

A Java based client API runtime library supporting the various integration business functions. The goals of the client API library is to:

- provide a unifying common interface between the application and service broker tier
- be a simple API
- be lightweight
- be Agencyport product independent
- not be reliant on communication libraries, such as Apache Axis.

### Example

The following is a sample of a client API library:

- Account level
  - addAccount()
  - updateAccount()
  - getAccount()
- LOB level
  - evaluate()
  - rate()
  - submit()

- `getPolicy()`
- Product Definition Extraction
  - `getClassCodes()`
  - `getQuestionSets()`
  - `getCoverage()`

## Reference Implementation Assets (RIA)

The Reference Implementation Assets (RIA) is a set of source level artifacts (Java source files and other supporting materials) that bridges between Agencyport's portal framework and the client API library, and is Agencyport product dependent.

### Example

Examples include:

- Display and process side connectors
- List builders
- TDPs
- Support for inbound SOAP web services
- Data synchronization handlers (inbound and outbound), such as account management

## Service Broker Tier

### Connect5

The service broker tier serves as the abstraction layer between the application and service provider tiers. The responsibilities of this tier include:

- Message routing to/from the service provider
  - Transformations (XSLT) of application data formats to/from service provider dictated formats
  - APNAX is the default in/out payload between application layer and service broker layer when exchanging account and LOB images
    - TDF page/code list is returned by Connect5 in product definition extraction scenarios
    - Proprietary message formats
- Security

The communication protocol between the application and service broker tiers initially adopts the SOAP 1.2 version (currently native to Connect5).

# Application Properties

The application startup framework initializes a static properties file manager object named `com.agencyport.utils.AppProperties`. When running from the servlet container, `AppProperties` is given its properties file from `web.xml` via the "PROPERTY\_FILE\_NAME" startup context parameter, which is read by the `InitializerServletContextListener`:

```
<context-param> <param-name>PROPERTY_FILE_NAME</param-name> <param-value>/WEB-INF/framework</param-value> </context-param>
```

The `InitializerServletContextListener` instantiates the static `AppProperties` object during [application bootstrap](#). Following initialization, classes that want to obtain properties when running in the servlet container should follow the following model:

```
private static AppProperties appProperties = AppProperties.getAppProperties(); ... String myProp =
appProperties.getStringProperty("myPropName");
```

If creating a JUnit driver, or a class with a `main()` to instantiate and then use the static `AppProperties`, the model is the following:

```
private static AppProperties appProperties; ... AppProperties.loadApplicationPropertiesFile(full path to properties file); ... appProperties =
AppProperties.getAppProperties(); ... String myProp = appProperties.getStringProperty("myPropName");
```

## Property Concatenation

Applications can decide to have multiple properties files for a variety of reasons. The property concatenation operator `+=` allows for the distribution of the same property value across multiple files. This eliminates the need to centralize multiple property values in one spot in one property file.

The property concatenation operator allows you to distribute multiple property values to multiple spots in one to n property files.

### Example

Main property file:

```
property.name=property-value1\ property.name+=property-value2
```

## Property Substitution

You can substitute the values of properties for use in other properties by using the  `${property_name}` syntax shown below:

### Example

```
my_portal_app=http://localhost:8105/AgencyPortal my_portal_app_frontservlet=${my_portal_app}/FrontServlet
```

## Customizing Framework Base Classes

The application can easily extend several of the framework classes to modify or extend the framework's own implementation. These are spread out over a variety of functional areas. The technique to convey your derived class to the factory method is done via application properties. The following table lists some examples of classes that you can extend:

Functional Area(s)	Framework Base Class	Factory Method	Java class name application property	Scope	Comments
Reading a work item into a set of HTML data containers	<code>com.agencyport.data.TransactionDataReader</code>	<code>com.agencyport.data.TransactionDataReader.create()</code>	<code>transactionId.transaction_data_reader_class_name</code>	Transaction	Applications can extend the base transaction data reader class to support the data

Functional Area(s)	Framework Base Class	Factory Method	Java class name application property	Scope	Comments
upload, transaction validation and correction					read operations related to non-SDK pages.
DTR - transaction definition provider creation	com.agencyport.trandef.provider.builtin.XMLTransactionDefinitionprovider	com.agencyport.trandef.provider.TransactionDefinitionProvider.createFromAppPropertyName	TDP.<transaction id> where transaction ID relates to the transaction ID in the TDF	Transaction	Applications can extend the product's XML transaction definition provider class to add content to a page that does not originate in the TDF.
Policy change	com.agencyport.policysummary.changesummary.changemanagement.PolicyChangeSummarizer	com.agencyport.policysummary.changemanagement.PolicyChangeSummarizerFactory.createPCS	<LOB from TDF>.pcs_class_name	Transaction LOB	Applications can extend policy change summarizer implementations to modify several different behaviors associated with the base implementation.
User preferences/defaults persistence	com.agencyport.defaults.DefaultsDBManager	com.agencyport.defaults.DefaultsDBManager.create	defaults_db_manager_class_name	Application	Applications can extend to alter the sort/filtering criterion to a user's

<b>Functional Area(s)</b>	<b>Framework Base Class</b>	<b>Factory Method</b>	<b>Java class name application property</b>	<b>Scope</b>	<b>Comments</b>
					default list.
JSP helper	com.agencyport.jsp.JSPHelper	com.agencyport.jsp.JSPHelper.get	jsphelper_class_name	Application	The framework uses this helper class to help reduce JSP Java inline source complexity. Applications can extend this class to add new functionality for custom pages.
Database monitoring	<none> custom class must fully implement the com.agencyport.domXML.web.ICustomDBManagerMonitor	com.agencyport.domXML.web.APDOMDBManager static initializer	custom_dbmanager_monitor_class_name	Application	Application hook for any APDOMDBManager managed data access.

# Property File Reference

The property files included as part of the AgencyPortalSDK supply information that is required at runtime to "find" things and configure other application settings. For example, the location of the log file, schema files, the information required to connect to the database, etc.

You can also configure available services and transactions, such as:

- The various built-in and user-defined services required at bootup
- Available rule providers for field validations or to execute business rules
- The mapping of servlets for each page for either builtin or user-tailored servlets

The following is a list of some the major properties read by the SDK:

## Framework Properties

The framework.properties file is used to configure application properties. The following is a list of properties included in this file and any settings we recommend for production.

Property	Description	Default Value	Recommended Setting (if any)
show_extended_error_detail	Controls whether to display extended error details on the navigation error JSP (APErrorJSP.jsp).	false	
transaction_file_manager_mode	Indicates whether the product artifacts are reloaded by the server when changed. Possible values for this property are dynamic and cache.	cache	Dynamic for testing Cache for production
client_debug	The client_debug property value is passed to the client side JavaScript. A true value enables the AgencyPortal debug console.	false	
print_session_attributes_to_system_out	Defines whether the application should print all the session attributes out to the command window. The value is usually set to true during development and false otherwise.	none	
print_request_parameters_to_system_out	Defines whether the application should print all the HTML form fields out to the command window. The value is usually set to true during development and false otherwise.	none	

## APDataCollection

APDataCollection is the class that supports the Agencyport XML Engine (AXE). It is used to manage any XML document with a predefined XML schema available at runtime. Refer to the [Agencyport XML Engine \(AXE\)](#) topic for more information.

Property	Description	Default Value	Recommended Setting (if any)
default_xml_pretty_format		false	
system_generated_id_attribute_type	This property dictates the use of globally or document unique IDs.	guid	

Property	Description	Default Value	Recommended Setting (if any)
	<ul style="list-style-type: none"> <li>If this property is missing or is equal to the value 'guid,' then a 36 byte globally unique ID generator engages.</li> <li>If the value is 'docuid,' then a document unique sequence number generates.</li> </ul> <p>The value generates in the following format: Nnnnnnnn (where N is the character N and nnnnnnnn is a sequence number unique within the scope of both the current and original documents maintained by APDataCollection).</p>		
autoMaintainIdAttributes	Engages the automatic ID attribute maintenance at application scope.	true	

#### Transaction Definition File

The transaction definition file (TDF) contains a set of metadata that describes a single transaction. Refer to the [Transaction Definition/Page Library](#) topic for more information. The following properties provide additional support for the TDF.

Property	Description	Default Value	Recommended Setting (if any)
print_transaction_definition_file	Indicates whether the transaction definition files have to log.	false	
transaction_definition_file_dump_directory	Provides the location of the directory where to stage the transaction files.	none	
date.maxLength	Indicates the value to be used for the maxLength and size attributes of TDF date fieldElements.	10	N/A

#### Database

Refer to the [Database](#) topic for more information.

Property	Description	Default Value	Recommended Setting (if any)
db_table_prefix	You may need to add table prefixing in some installations when the user ID that is provided in the JNDI is not owner of the tables or stored procedures. The table prefix setting instructs the SDK database framework to apply the specific prefix in front of table names. If you use this facility, it is important to add the dot (.) at the end of the prefix value.	""	
datasource_prefix	Used to prefix the JNDI connection name (datasource) with the required prefix. Some application servers require a prefix to be provided as part of the JNDI name. This property can be left blank or the reference can be removed from the database property.	none	

Property	Description	Default Value	Recommended Setting (if any)
datasource	The JNDI name that refers to the database connection parameters. This can also include a prefix, if necessary. The datasource_prefix parameter is not part of the JNDI name setup in your application server. Additional datasource names can be specified by prepending a customprefix (e.g., codelist_db.datasource=codelistDb).	none	
database_agent_class_name	The SDK database agent class to use for the application.	com.agencyport.database.SQLDatabaseAgent	
force_use_jdbc	The force_use_jdbc property can be used to prevent usage of JNDI to retrieve the JDBC Connection parameters and instead derive the values from Application Properties. This option is typically used for connecting to a database during JUnit tests and, in other scenarios, where JNDI isn't available. If force_use_jdbc is set to true, the jdbc_driver, database_url, database_username and database_password properties become required. Connection parameters for additional databases can be specified by prepending a custom prefix to the property name.	false	
jdbc_driver	The JDBC driver class name to use for connection to the database. Required when force_use_jdbc is set to true. Connection parameters for additional databases can be specified by prepending a custom prefix to the property name.		
database_url	The database URL to use for connection to the database. Required when force_use_jdbc is set to true. Connection parameters for additional databases can be specified by prepending a custom prefix to the property name.		
database_username	The database username to use for connection to the database. Required when force_use_jdbc is set to true. Connection parameters for additional databases can be specified by prepending a custom prefix to the property name.		
database_password	The database password to use for connection to the database. Required when force_use_jdbc is set to true. Connection parameters for additional databases can be specified		

Property	Description	Default Value	Recommended Setting (if any)
	by prepending a custom prefix to the property name.		
audit_connection_count	Boolean flag that governs whether each database connection acquisition and release is tracked and logged.	false	Should only be true to help diagnose a database connection leak or overrun.

#### Logging Properties

The following properties control the logging subsystem and are used to monitor for hot updates. Refer to the [Logging Configuration](#) topic for more information.

#### Logging Destination

Property	Description	Default Value	Recommended Setting (if any)
output_log_dir		none	

#### Logging Configuration

Property	Description	Default Value	Recommended Setting (if any)
defer_logging_initialization	Global flag used to turn off all AgencyPortal logging initialization	false	
application.logging_config_file	A list of the SDK's application logging configuration files. These files are loaded to make up the actual logging configuration for the application. Each entry is delimited with a comma. Refer to the <a href="#">Application Logging Configuration File</a> section for more information.	\${custom_logging_config_file}	
logging_config_file_monitor	Enables/disables live monitoring of the application.logging_config_file to support "hot updates" to the file at runtime.	true	Should be false in a production environment.
application.logger_name_prefix	Unique logger name prefix. Keeps logger hierarchies separated when sharing JVM with other Agencyport applications.	none	

#### Performance Subsystem

The following properties are used to enable the performance subsystem. The subsystem is turned on by supplying a file name. Refer to the [Performance Logging](#) topic for more information.

Property	Description	Default Value	Recommended Setting (if any)
PERFORMANCE_RESULTS_FILENAME	If the performance system is turned on (by presence of a log file name), the framework writes a number of	none	

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
	performance metrics to the output file.		
PERFORMANCE_RESULTS_TRUNCATE_ON_STARTUP	A new performance log file is created on server startup when this property is set to YES. The default value is NO.	none	
PERFORMANCE_RESULTS_NUMBER_OF_GENERATIONS		3	
PERFORMANCE_RESULTS_CAPACITY		200000	

#### Boot Service Providers

The following property is used to implement the IBootServiceIntf subsystem. Refer to the [Application Bootstrap](#) topic for more information.

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
boot_service_providers	This lists several built-in service providers that require some activity to take place during application bootup, such as the HTMLETemplateManager and the TransactionDefinitionManager. User supplied classes or subsystems are also listed here.	none	

Save and Exit Next Page

The following properties define where the application "goes" when the framework standard return button is clicked (on the HTML page).

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
SAVE_AND_EXIT_NEXT_PAGE	This entry defines where the application will "go" when the framework standard Return button is selected (in the HTML page).	/DisplayLogonForm	

#### My Portal URL

For various reasons, such as security, upload integration and Turnstile integration, the application needs to be aware of its own URL at runtime. The URL is registered by setting the following properties:

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
my_portal_app_protocol	Usually 'http' or 'https'	none	
my_portal_app_domain	The domain with any prefixes without any port number.	none	
my_portal_app_port	Whatever port your application container will be listening to. The default for Tomcat is '8080'.	none	
my_portal_app	This is the master property containing the entire URL. The default is set to be the sum of the parts below.	none	
my_portal_app_frontservlet	The FrontServlet URL pattern.	none	

#### Endorsement Policy Image Retrieval

Property	Description	Default Value	Recommended Setting (if any)
POLICY_IMAGE_CLASS	Class that is responsible for retrieval of policy image to initiate the translation.	comagencyport.policyadmin.DemoPolicyImageRetriever	
PAS_URL	Path to the servlets to call to get the policy image.	none	
ACTIVATE_PAS_SIMULATION	Property dictates whether the policy administration simulation needs activated. This is used when endorsement or renewal is created from a new business work item.	false	

#### Other Properties

Property	Description	Default Value	Recommended Setting (if any)
resources_root	The default home directory where the product database and the definition files pertaining to it live	\${ context_path }/WEB-INF	
additional_properties_to_load		none	
file_temp_directory		none	
fileattachment_cache_directory	Specifies the cache directory used for file attachments. The user should create this directory.	DEFAULT_CACHE_DIRECTORY	

#### Autosave

Data entered on a work item transaction page is automatically saved based on the time interval entered in the following property. Refer to the [Autosave](#) topic for more information.

Property	Description	Default Value	Recommended Setting (if any)
auto_save_interval	The amount of time in seconds between checks to autosave data. 0 or less turns auto saving off. The value defaults to 60 if no property is provided.	60	

#### Work Item Assistant

The Work Item Assistant feature can provide comments, file attachments and other active users for a work item. Refer to the [Work Item Assistant](#) topic for more information. This feature is enabled and customized using the following properties:

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
workitem_assistant_supported	Boolean flag that governs whether the work item assistant feature is active.	none	
max_file_attachment_size	The size limit for files uploaded via the work item assistant feature.	10240000	

#### Timeline

The Timeline feature allows a user to view the history of a work item. Refer to the [Timeline](#) topic for more information. This feature is enabled using the following property:

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
timeline_supported	Indicates whether you want to turn on the Timeline feature.	false	

#### Installer

AgencyPortal includes a web-based installer feature. Refer to the [Web-Based Installer](#) topic for more information. This feature is enabled and customized via the following properties:

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
web_installer_enabled	Determines whether the web-based agencyportal installer is used. The installer servlet will be engaged by the EntryPointServlet or accessed directly via the browser when this property is set to true.	false	

#### Automatic Database Installer

The automatic database installer allows you to create the agencyportal database tables during application startup. Refer to the [Database Configuration](#) topic for more information. Use the following properties to enable this feature:

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
automatic_database_installer_enabled	Indicates whether to create the agencyportal tables during application startup.	false	
base_sql_directory	The directory where SQL files known to the installer live.	none	
strict_datasource_lookup		none	
ordered_sql_package_list	A list of SQL "packages" for the installer to use, specified in execution order. A "package" is a sub-directory of the base_sql_directory and must contain a sub-directory for the database type, as well as a sqlOrder.properties file that specifies the files to execute with the orderedSqlScriptList property.	none	

#### Account Management

Account management allows users to create accounts within the application. Refer to the [Account Management](#) topic for more information. The following properties allow you to customize this feature:

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
application.account_management	Boolean flag that governs whether account management is on or off at the application scope.	false	
application.account.management.support.merge	Boolean flag that governs whether account merge is supported.	none	
application.account.management.usergroup.filter	Boolean flag that governs whether accounts can only be accessed through user group permissions.	none	
account.manager_classname	Java class name that implements the com.agencyport.account.model.IAccountManager interface.	com.agencyport.account.impl.AccountManager	
application.workitem_save_exit_to_account	Boolean flag that governs whether a save and exit action on a work item with a linked account will lead the user to the account save_exit page (by default, the summary page).	false	

#### Search and Filter Settings

AgencyPortal uses an embedded Solr engine for the search and filter functionality within the application. Refer to the [Solr Property Reference](#) topic for more information.

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
base_solr_app		none	
base_solr_url		none	
application.worklist.indexsearch.provider.baseurl	Base URL for the work item Solr interface.	none	
application.account.indexsearch.provider	Java class name that implements the com.agencyport.account.search.IAccountIndexSearchProvider interface.	com.agencyport.account.search.SolrIndexSearchProvider	
application.account.indexsearch.provider.baseurl	Base URL for the account Solr interface.	none	
solr_manager	Designates the class name of the implementation class for the ISolrManager interface.	com.agencyport.solr.manager.SolrManager	
solr_url_resolver	Designates the class name that implements the ISolrUrlResolver interface.	com.agencyport.solr.manager.SolrUrlResolver	

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
solr_batch_indexers	Lists the batch processors revealed in the administrative function that synchronizes their respective Solr indexes.	none	
solr_home	Tells the application where the SOLR_HOME directory is located on the filesystem. This property is only respected in non-clustered environments.	none	

#### Solr Security Settings

The following properties control the security settings for Solr. Refer to the [Solr Property Reference](#) topic for more information.

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
solr_keystore_filename	The name of the keystore file that contains a key named "internal_solr_key".	(asResource)keystore/solr.key	
solr_keystore_password_filename	The name of the SecurePasswordFile that contains the password to the keystore file.	(asResource)keystore/solr-info.bin	
solr_token_valid_lifespan_in_seconds	Designates the amount of time that a Solr token is valid for. This value should be as small as possible.	5	
solr_security_manager	Designates the class name of the implementation for the Solr security manager interface.	com.agencyport.solr.security.SolrSecurityManager	
solr_http_request_interceptor	Class used to intercept outbound HTTP requests to Solr before they leave the application (to add security tokens and other request headers).	com.agencyport.solr.security.APSolrHttpRequestInterceptor	

#### Solr Cloud

Solr Cloud provides the ability to set up a cluster of Solr servers. The following properties are used to set up this feature. Refer to the [Solr Property Reference](#) topic for more information.

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
solr_cloud.node.host	Host name	\${application.hostname}	
solr_cloud.node.hostPort	Port		
solr_cloud.node.hostContext	Host context		
solr_cloud.zookeeperHosts	Comma-separated list of addresses for each node in your ZooKeeper ensemble		

#### Turnstile Integration

Enabling the Turnstile feature within your AgencyPortal application is configured via the following properties. Refer to the [Turnstile Integration](#) topic for more information.

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
enable_turnstile_upload		false	
max_file_size	The maximum total filesize (in megabytes) allowed for a single upload to Turnstile via the Turnstile Integration Kit.	10	
APPEND_USER_CREDENTIALS_TO_RETURN_URL		false	
ENCRYPT_USER_CREDENTIALS		false	

#### Work Item Record Locking

The following property is used to determine how long a work item is locked. Refer to the Work Item Record Locking topic for more information.

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (Optional)</b>
work_item_record_lock_time_in_seconds	Number of seconds a work item is locked after a user is no longer making changes to a work item. If a user closes their browser window while in a work item, the work item is locked from the last update, plus number of seconds configured using this property. The default value is 600 (10 minutes) and blank; 0 (zero) or a negative number turns work item locking off, which is not recommended.	600	

#### Default Image Class

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
application.css_image_class	The default image class name if none is found at the LOB property level.	none	

#### Defaults

The following property is used to enable or disable the My Defaults feature. Refer to the [My Defaults](#) topic for more information.

Property	Description	Default Value	Recommended Setting (if any)
supportsDefaults	Used by all transactions that have not set the "supportsDefaults" transaction attribute. This is set temporarily until the defaults behavior is properly scoped out.	true	

#### Progress Bar

The Percent Complete feature includes a progress bar to indicate the percent complete status of a work item. The following property is used to display (or not display) this feature. Refer to the [Percent Complete](#) topic for more information.

Property	Description	Default Value	Recommended Setting (if any)
show_progress_bar	Indicates whether to display the percent complete progress bar on transaction pages.	true	

#### REST API Resource Class Registration

Property	Description	Default Value	Recommended Setting (if any)
application.rest_resource_classes		none	

#### Default Sort and Query Fields

Property	Description	Default Value	Recommended Setting (if any)
WorkItems View.queryField.status.selected		none	
WorkItems View.queryField.status.opCode		none	
WorkItems View.queryField.status.operands		none	
WorkItems View.sortOption.effective_date.selected		none	
WorkItems View.sortOption.effective_dateascending		none	
Accounts View.sortOption.last_update_time.selected		none	
WorkItems View.sortOption.last_update_time.ascending		none	

#### Sensitive Data Fields

The following property is used to list specific client sensitive XPaths that should be encrypted when the data is at rest. Refer to the [Additional Security Considerations](#) topic for more information.

Property	Description	Default Value	Recommended Setting (if any)
application.encryption_fields	List of client sensitive XPaths that should be encrypted when data is at rest.	none	

#### Suppress TVR Error Messages

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
application.map_persisted_TVR_messages_to_ui	Indicates whether to suppress TVR error messages from being mapped to the UI.	false	

#### HTML Template File

The HTML template file provides the location of an `html_element_definitions.txt` file extension. An extension is only necessary if an application wants to alter one or more templates defined in the SDK's copy. The SDK maintains an immutable copy of this configuration file in `apwebapp.jar`. If an application requires an alteration of one or more template names in the SDK's version, the following configuration is loaded on top of the SDK's version, overlaying any templates in the former with the template definitions in the latter:

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
html_template_file	The full path and file name of the file containing any application HTML template alterations/modifications		

## Security Properties

The `acsi.properties` file is used to store all of the application properties specific to ACSI. Each project should have its own `acsi.properties` file. Refer to the [ACSI Step-by-Step Guide](#) topic for more information on ACSI.

The following is an example of what an `acsi.properties` file should look like:

```
#####
Properties specific to the support of ACSI
ACSI properties - engages correct
implementations for various application ACSI extensions
security.profile_classname=com.agencyport.security.profile.builtin.PortalSecurityProfile
security.profile_manager_classname=com.carrier.security.CarrierSecurityProfileManager
#####
Factory list for implementations of
com.agencyport.security.resource.ISecureResourceFactory interface. # Each factory implementation is a factory for a
com.agencyport.security.resource.ISecureResource instance # which is sensitive to the current security profile. # Examples of secure resources
are menus and front servlet (front controller). # These are used by security filter. The menu resource is also used # to support the menu UI.
#####
secure_resource_factories=com.agencyport.secure.menu.provider.SecureMenuFactory;com.agencyport.secure.front.SecureFrontControllerFactory
```

The following is a list of the properties included in this properties file and any settings we recommend for production:

#### ACSI Properties

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
security.profile_classname	Designates the name of the security class to use.	None	
security.profile_manager_classname	Designates the class name of the ACSI security profile manager implementation.	None	
security.credentials_agent_classname	Designates the name of the ACSI security agent credentials class.	com.agencyport.security.credentials.CredentialsAgent	

#### Default Encryption Algorithm, Mode and Padding Cipher Information

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
security.<encryption_type>.cipher_transformation	Designates the default encryption algorithm, mode and padding cipher transformation.		

Implementation of com.agencyport.security.resource.ISecureResourceFactory Interface

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
secure_resource_factories	Designates the list of ISecureResource implementation classes that the security filter will interact with to secure each incoming HTTP request. Multiple entries are delimited with a semi-colon.	None	

CSRF Protection and Configuration

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
security.csrf_uri_include_list	Designates the selected servlets and their method types, which CSRF detection is applied to.	None	
security.csrf_protection	Indicates whether to activate or deactivate CSRF protection True = activated; False = deactivated	True	This field should always be true. You should only change this field to false during performance testing and during no other circumstances.

Work Item and Account Action Verification Configuration

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
security.verify_work_item_actions	Indicates whether to activate or deactivate work item action verification. True = activation; False = deactivation	True	
security.action_to_permission_map	Maps incoming work item action values to their permission name counterparts. Is used to verify that the user has authority to execute the incoming action on the work item. Actions include MoveWorkItems, MergeAccount, Delete, DoAssign, Copy, View, Open, Claim, Approve, Endorse and Link.	None	

URL Verification Configuration

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
security.verify_urls	Indicates whether to activate or deactivate URL verification. True = activate; false = deactivate.	True	
security.valid_url_start_sequences	Used to provide a white list of valid starting sequences for URLs.	None	
security.valid_forward_resource_extension		None	
security.invalid_foward_url_tokens		None	

#### Underlying Details Configuration

Property	Description	Default Value	Recommended Setting (if any)
security.reveal_exception_details	Boolean flag that controls whether underlying details are provided when a security issue or breach has been detected.	True	This property should only be set to false for production implementations. If this flag is set to true, generic secure message text will display to the sender for any AP security exception.

#### HTTP Response Headers

Property	Description	Default Value	Recommended Setting (if any)
security.add_secure_response_headers		True	
security.content_security_policy			
security.secure_response_headers		None	

## Application Logging Properties

Application logging properties are configured in the application.logging.properties file. Refer to the [Logging Configuration](#) topic for more information. The following is a list of properties included in these files and any settings we recommend for production.

Property	Description	Default Value	Recommended Setting (if any)
com.agencyport.portal.handlers	Comma separated list of logging handlers to allocate for the application.	None	
com.agencyport.portal.loggers	Comma separated list of root loggers to allocate for the application.	com.agencyport	
com.agencyport.logging.PortalFileHandler.pattern	Specifies the log file target path name for the framework's default file handler.	None	
com.agencyport.logging.PortalFileHandler.level	Specifies the logging level for this handler.	None	
com.agencyport.logging.PortalFileHandler.limit	Specifies the maximum size in bytes the log file can grow to before rotation begins.	100000	
com.agencyport.logging.PortalFileHandler.count	Specifies the number of generations in the file rotation strategy.	3	
com.agencyport.logging.PortalFileHandler.formatter	Specifies the formatter that the default file handler should use.	None	
com.agencyport.logging.PortalFileHandler.append	Specifies whether to append or truncate the target log file.	True	

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
com.agencyport.logging.DefaultLogFormatter.timestamp.format	Format string that controls the time stamp format portion of the log line.	None	
com.agencyport.logging.syslog.SyslogHandler.formatter	Formatter used by the syslog handler.	None	
com.agencyport.logging.syslog.SyslogHandler.level	Specifies the logging level for this handler.	ALL	
com.agencyport.logging.syslog.SyslogHandler.host	Host name/IP Address of syslog server.	None	
com.agencyport.logging.syslog.SyslogHandler.port	Port number of syslog server.	None	
java.util.logging.ConsoleHandler.level		info	
com.agencyport	Boolean flag determines whether the application root loggers should chain to the handlers at the JVM level.	None	

## Cache Properties

The cache.properties file is used to set any overrides to caching sizes or retention periods. The following is a list of parameters included in this file and any settings we recommend for production:

### Resource Bundle Properties

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
resource_bundle_cache_max_size_system_default		None	
resource_bundle_cache_max_size		20	
resource_bundle_cache_age_in_minutes_system_default		None	
resource_bundle_cache_age_in_minutes		60	

### Cache Properties

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
cache_directory	Designates where the framework serializes compiled product definitions. This defaults to \${application-context}/WEB-INF/cache, which is good for any environments other than Maven based development environments.	cache	
schema_cache_directory	Designates where the framework serializes compiled XSD files. This defaults to \${application-context}/WEB-INF/schema_cache, which is okay for any environments other than local m2e-WTP based development environments.	schema_cache	

## Localization Properties

The localization.properties file is used to configure any application properties relating to localization. Refer to the [Localization](#) topic for more information. The following is a list of parameters included in this file and any settings we recommend for production:

Property	Description	Default Value	Recommended Setting (if any)
language_code	ISO language code value.		
application.base.rb_root			
application.spanish.rb_root			
locale.resource_bundle_paths	Controls where resource bundles are loaded from the file system.		

Locale Emulation

Property	Description	Default Value	Recommended Setting (if any)
locale.locale_emulation			

Locale Awareness Flag

Property	Description	Default Value	Recommended Setting (if any)
application.is_locale_aware	Indicates whether the application will pick up resource bundle translations for product definitions.	false	

Resource Bundle Map

Property	Description	Default Value	Recommended Setting (if any)
aworkItemAssistant.workitemassistant			

## Version Properties

The version.properties file is used to configure JAR version requirements. The following is a list of parameters included in this file and any settings we recommend for production:

Property	Description	Default Value	Recommended Setting (if any)
verify_jar_versions	Boolean flag that engages JAR file version verification at bootstrap time.		
application.jar_list			
SDK_VERSION			Needs to be the same as the actual version of the app you are running.
apbase.classname			

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
apbase.version_location			
apbase.minimum_version			
apsecurity.minimum_version			
apsecurity.version_location			
apsecurity.classname			
apwebapp.classname			
apwebapp.version_location			
apwebapp.minimum_version			
jdom.classname			
jdom.version_location			
jdom.minimum_version			
jaxen.classname			
jaxen.version_location			
jaxen.minimum_version			
xsom.classname			
xsom.version_location			
xsom.minimum_version			
annovention.classname			
annovention.version_location			
annovention.minimum_version			
jackson-core.classname			
jackson-core.version_location			
jackson-core.minimum_version			
jackson-databind.classname			
jackson-databind.version_location			
jackson-databind.minimum_version			

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
jersey-common.classname			
jersey-common.version_location			
jersey-common.minimum_version			
jersey-server.classname			
jersey-server.version_location			
jersey-server.minimum_version			
clientAPI.classname			
clientAPI.version_location			
clientAPI.minimum_version			
solr-core.classname			
solr-core.version_location			
solr-core.minimum_version			
javax.ws.rs-api.classname			
javax.ws.rs-api.version_location			
javax.ws.rs-api.minimum_version			

## Email Notification Properties

The `email_notification.properties` file is used to set email based notification settings based on work item status changes, adding comments or adding file attachments. The following is a list of parameters included in this file and any settings we recommend for production:

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
<code>email.email_provider_class_name</code>	Class that drives email notifications. Used to override create a class that implements <code>com.agencyport.emailEmailProvider</code> .	<code>com.agencyport.email.impl.EmailProvider</code>	
<code>email.send_notifications</code>	Indicates whether to turn on email notifications on or off for the entire application.	False	
<code>email.smtpHost</code>		None	
<code>email.from_address</code>	The from address used for email notifications.	None	

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
email.work_item_owner	Indicates whether to email the current owner.	True	
email.work_item_creator	Indicates whether to email the creator of work item.	True	
email.send_status_change_email	Indicates whether to turn on or off the email notifications for status changes.	False	
email.status_change_message_class_name	The email message class name for status changes.	com.agencyport.email.impl.StatusChangeMessage	
email.send_file_attachment_email	Indicates whether to turn on or off the email notification for adding file attachments.	False	
email.file_attachment_message_class_name	The email message class name for adding file attachments.	com.agencyport.email.impl.FileAttachmentMessage	
email.send_comments_email	Indicates whether to turn on or off the email notifications for adding comments.	False	
email.comments_message_class_name	The email message class name for adding comments.	com.agencyport.email.impl.CommentMessage	
FORGOT_PASSWORD SUBJECT		None	

## LOB Specific Properties

### Transaction Package Name

Normally, every transaction known to the system is listed in the transaction package name section. This tells the system where to find the user-tailored servlets for a transaction.

### Example

CMD has been used to prefix all of the sample servlets. For personalauto, the entry might look like the following:

```
persauto.root_command_class_name=com.yourco.servlets.persauto
personalAuto.root_command_class_name=com.agencyport.servlets.persauto.CMD
endorsePersonalAuto.root_command_class_name=com.agencyport.servlets.persauto.endorse.CMD
workersComp.root_command_class_name=com.agencyport.servlets.workerscomp.CMD
endorseWorkersComp.root_command_class_name=com.agencyport.servlets.workerscomp.endorse.CMD
```

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
transactionName.root_command_class_name	root_command_class_name is constant, transactionName must match a transaction name, packageName is the Java package where any custom		

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
	classes reside and .prefix (optional) is used to form a common prefix for all the custom classes		

# Upload Writer Programming

This section includes information on features available for the upload writer infrastructure and how application developers can best take advantage of these features to maximize functionality within the core product and minimize the need for custom code.

Upload writer functionality includes the following:

- The ability provide a comprehensive merge document to account for data elements:
  - modified, introduced or deleted by custom code before upload data manager runs
  - modified, introduced or deleted by upload data manager's TDF based data transfer mechanism
  - modified, introduced or deleted by data corrections
- The ability to run the ID retention algorithm and data corrections together or independently
- The ability for an application to control which original document version is used for the ID retention algorithm, as well as the final original document system of record that ends up as an XML document stored in the `original_xmlstring` column on the record for the corresponding work item in the `work_item` table.
- The calculation of the target index for roster based updates during a TDF based data transfer process to use the built-in target index calculation provided by the framework whose configuration is conveyed via the TDF roster page's index page element, if present. The `UploadManager.determineTargetIdArray()` method first invokes the `IndexManager.derive()` method, assuming an index page element is present and defers any further calculations if an index is calculated by index manager.
- To allow the `UploadDataManager` class to engage referential integrity operations before and after the TDF based data transfer process.
  - The default referential integrity process [called from `UploadDataManager.executePreDataTransferReferentialIntegrity()`] called to prepare the input XML document before the TDF based data transfer process is carried out removes just the ID attribute reference values, which contain a value that has no corresponding ID value present elsewhere in the document.
  - The default referential integrity process [called from `UploadDataManager.executePostDataTransferReferentialIntegrity()`] called after the TDF based data transfer process executes any referential integrity operations conforming to the `com.agencyport.domXML.refintegrity.IReferentialManager` interface that have been registered.

## Note

Any referential operations registered are also engaged while the output XML document is built-up and later during any data correction operations that execute.

- Two methods to support the ACORD standard to/from ACORD simplified transformations:
  - `ImportBaseProcessor.convertACORDStandardToACORDSimplified()`
  - `ImportBaseProcessor.convertACORDSimplifiedToACORDStandard`Both implementations engage any registered product transformers by default.
- Logging and performance object creation
- Reusability and extensibility

# ImportBaseProcessor and UploadDataManager Classes

The `ImportBaseProcessor` and `UploadDataManager` classes include protected hook points to accommodate application customizations.

## Important

Applications that elect to use the integration kit `CommonUploadWriter` derivation of the product's `ImportBaseProcessor` class cannot directly alter the behavior of the `ImportBaseProcessor` non-final public and protected methods.

The following methods are included to help achieve reusability and extensibility. It is important that an upload writer developer reviews these methods to maximize the reusability of framework logic, reduce the amount of replicated framework logic and minimize the amount of custom code.

## ImportBaseProcessor

`ImportBaseProcessor` includes the following methods:

- **writeResponse** - hook point that provides the application a last chance to alter the response that is delivered back to the client via the step data structure. This is the place where the application can assemble the response step data structure, which may include an XML response as part of `//AgencyConnectConversionStep/OutputStream`. The default implementation invokes the `writeAPDataToDebugOutput()`, which may or may not do anything of note, and then invokes the `writeSuccessMessage()` methods.
- **determineWorkItemOwnership** - hook for sub classes to do something specific about the work item ownership. The default implementation assumes that the owner of the work item is the current subject and that the owning group is the primary user group of the current subject.
- **convertACORDStandardToACORDSimplified** - hook point for converting the ACORD standard document to its ACORD simplified version. The default implementation will carry out any transformation that is a registered product artifact supported by the transformer type.

The following is an example:

```
<transformer name="ACORD standard to AP WC Simplfied ACORD "
type="aggregator" targetFormat="ACORDSimplified"
lob="WORK">
 <step name="WC Transformation target - ACORD Standard
Step 1" type="java"

 className="com.agencyport.workerscomp.transformers.WorkersCompTran
sformer" />
</transformer>
```

- **convertACORDSimplifiedToACORDStandard** - hook point for converting the ACORD simplified document back to its ACORD standard version (reverse transformation). The default implementation will carry out any transformation that is a registered product artifact supported by the transformer type.

The following is an example:

```
<transformer name="AP WC Simplfied ACORD to ACORD standard"
type="aggregator" targetFormat="ACORDStandard"
lob="WORK">
 <step name="WC Transformation target - ACORD Standard
Step 1" type="java"

 className="com.agencyport.workerscomp.transformers.WorkersCompTran
sformer" />
 <step name="Shared Standard Built-in Transformations"
```

```
type="link" linkedTo="builtin-standardizations" />
</transformer>
```

- **writeAPDataToDebugOutput** - handy debug function that writes out all document views currently under management by the data collection to the file system, each as its own XML output file. This method reads an application property `debug_upload_transformations_dump_path`, which dictates the output directory to which the XML files are written to; one for each document view. The pattern for the file name derivation for each document view follows the following pattern:  
 `${application_property:APPLICATION_NAME}_${work_item_id}_${APDataCollection.getViewTypeDescription()}_writeResponse.xml`  
If this property is missing, then this method does nothing. Enabling this method should be restricted to debugging purposes only.

Name	Date modified	Type
pompey_1162_VIEW_CURRENT_ACORD_STANDARD_DOCUMENT_writeResponse.xml	7/5/2011 8:18 AM	XML Document
pompey_1162_VIEW_CURRENT_DOCUMENT_writeResponse.xml	7/5/2011 8:18 AM	XML Document
pompey_1162_VIEW_MERGED_ACORD_STANDARD_DOCUMENT_writeResponse.xml	7/5/2011 8:18 AM	XML Document
pompey_1162_VIEW_MERGED_DOCUMENT_writeResponse.xml	7/5/2011 8:18 AM	XML Document
pompey_1162_VIEW_ORIGINAL_ACORD_STANDARD_DOCUMENT_writeResponse.xml	7/5/2011 8:18 AM	XML Document
pompey_1162_VIEW_ORIGINAL_DOCUMENT_writeResponse.xml	7/5/2011 8:18 AM	XML Document
pompey_1162_VIEW_PREVIOUS_ACORD_STANDARD_DOCUMENT_writeResponse.xml	7/5/2011 8:18 AM	XML Document
pompey_1162_VIEW_PREVIOUS_DOCUMENT_writeResponse.xml	7/5/2011 8:18 AM	XML Document

- **createAndRunUploadDataManager** - creates and runs an upload data manager. The default implementation invokes the `createUploadDataManager()` method and then invokes its `updateInputDataCollectionInPlace()` method.
- **createUploadDataManager** - factory method for the upload data manager instance. Applications that extend the `UploadDataManager` class must override this method for their subclass to get instantiated. The default implementation creates the standard `UploadDataManager` class. Applications using the `CommonUploadWriter` can register an LOB specific data manager under `<LOB>.upload_data_manager_class_name`.
- **executeCustomAlterPostUploadDataManager** - the custom handler for altering the XML after upload data management has finished. Called by the base galaxy note 4.7.0 case galaxy note 4.7.0 case `customAlterImportData()` method after calling the `com.agencyport.upload.UploadDataManager.updateInputDataCollectionInPlace(Transaction, boolean, boolean, int)` call.

## UploadDataManager

`UploadDataManager` includes the following methods:

- **establishPreConditions** - sets up the preconditions for the upload request.
- **createOutputDataCollection** - the factory method for creating and initializing the initial empty output data collection. The default implementation does the following:
  - o creates an empty output data collection sharing the same name as the input data collection.
  - o sets up the original document on the output data collection via the `getOriginalDocumentInPrepForTDFBasedDataTransfer(Transaction, boolean)` method.
  - o copies the last UID value from the input data collection into the output data collection
- **getOriginalDocumentInPrepForTDFBasedDataTransfer** - typically called from `createOutputDataCollection(Transaction, boolean)` to ascertain which document is considered the original document to use during the TDF based data transfer. The main function of this original document is for ID attribute retention purposes. The document returned by this method becomes the original view document on the output data collection during the TDF based data transfer.

### Note

If ID retention is a requirement, then an original document is necessary to install on the output data collection; otherwise, the ID retention algorithm will not be complete or effective.

- **executePreDataTransfer** - responsible for executing the processing immediately before the TDF based data transfer is carried out. The default implementation executes the following:
  - o runs a basic referential operation to delete any aggregates with ID references that are invalid via the `executePreDataTransferReferentialIntegrity()` method.
  - o nulls out the original document on the input document to cut down on the original values processing overhead during the read side of the TDF data transfer process.
  - o sets up the ID retention framework, depending on the `retainIdAttributes` parameter.
- **executePreDataTransferReferentialIntegrity** - runs referential integrity against the input data collection. The default implementation removes any ID references from aggregates that cannot be resolved. Under the covers, the `com.agencyport.domXML.refintegrity.BasicReferentialIntegrityManager` class handles the default referential integrity work.

- **executeDataTransfer** - contains the main data transfer processor taking the input data collection and building the output data collection using the fields in the TDF as the basis for determining the fields to copy from the input data collection to the output.
- **executePostDataTransfer** - processes immediately after the TDF based data transfer. The default implementation:
  - o clears any index management state via `com.agencyport.data.IndexManager.clear()` and runs any registered referential integrity operations via `com.agencyport.data.DataManager.maintainReferentialIntegrity(int)`.
  - o restores any previous field access and ID preserver handlers on the input and output data collections.
  - o fills out any ID attribute values on any remaining elements on the output data collection that were not filled in during the TDF based data transfer.
  - o initializes the original document of record on the output document via the `getOriginalDocumentSystemOfRecord(Transaction, APDataCollection)` method.
- **executePostDataTransferReferentialIntegrity** - runs referential integrity against the output document.
- **getOriginalDocumentSystemOfRecord** - typically called from `executePostDataTranser (APDataCollection, Transaction, boolean)` to ascertain which document should be considered the original document of record that will eventually be persisted to the `xmlstore.original_amlstring` column for this work item. The default implementation will be governed by a transaction/LOB/application level property whose base property name is `ORIGINAL_DOCUMENT_SYSTEM_OF_RECORD_BASE_PROPERTY_NAME`. There are two values that the framework understands for this property:
  - o `ORIGINAL_DOCUMENT_SYSTEM_OF_RECORD_FRONT_DOOR_DOCUMENT_VALUE`
  - o `ORIGINAL_DOCUMENT_SYSTEM_OF_RECORD_CURRENT_DOCUMENT_BEFORE_DATA_CORRECTIONS_VALUE`

This property is only active if the `com.agencyport.trandef.Transaction.supportsOriginalValue()` method for the current upload request returns a value of `true` and the `retainIdAttributes` parameter is set to `true`.
- **verifyDataCollection** - verifies that a data collection contains no elements with the same ID attribute value.

# Upload Message Mapper

The upload message mapping facility supports rich messaging with support to contextual information that relates to validation and corrections events triggered by transaction validation and corrections by upload data manager.

This functionality is engaged automatically and the messaging generated out of the box by the `com.agencyport.upload.UploadMessageMapper` class should suffice most, if not all, requirements.

There are two options for altering out of the box wording:

- extending/altering the message templates stored in the `validation_info` resource bundle that are used by the message mapper class following the approved methodologies for changing resource bundle entries.
- Java extension of the product base class.

The following entries in the `validation_info` resource bundle meet the various business requirements:

Resource Bundle Entry Name	Base Value	Designated Purpose
<code>message.Error.B2B.Required</code>	<code> \${context}, \${field_label} is required</code>	Message template for validation oriented events for missing required fields in cases where a related US state context is not known.
<code>message.Error.B2B.RequiredForUSState</code>	<code> \${context}, \${field_label} is required in the state of \${USState}</code>	Message template for validation oriented events for missing required fields in cases where a related US state context is known.
<code>message.Error.B2B.FieldUnknown</code>	<code> \${context}, \${message}</code>	Message template for validation oriented events when a discreet field element is not available.  Note  Many XARC rules will not have a related field element.
<code>message.Error.B2B.Invalid</code>	<code> \${context}, \${field_value} is not a valid value for \${field_label}</code>	Message template for validation oriented events for an invalid field value in cases where a related US state context is not known.
<code>message.Error.B2B.InvalidForUSState</code>	<code> \${context}, \${field_value} is not a valid value for \${field_label} for the state of \${USState}</code>	Message template for validation oriented events for an invalid field value in cases where a related US state context is known.
<code>message.CORRECTION.B2B.FieldRemoved</code>	<code> \${context}, \${previous_field_value} for the field \${field_label} was removed.  \${message}</code>	Message template for correction oriented events that remove a field value in cases where a related US state context is not known.
<code>message.CORRECTION.B2B.FieldChanged</code>	<code> \${context}, \${previous_field_value} for the field \${field_label} was changed to \${field_value}. \${message}</code>	Message template for correction oriented events that changed a field value in cases where a related US state context is not known.

Resource Bundle Entry Name	Base Value	Designated Purpose
message.Correction.B2B.FieldRemovedForUSState	<code> \${context}, \${previous_field_value} for the field \${field_label} for the state of \${USState} was removed. \${message}</code>	Message template for correction oriented events that changed a field value in cases where a related US state context is not known.
message.Correction.B2B.FieldChangedForUSState	<code> \${context}, \${previous_field_value} for the field \${field_label} for the state of \${USState} was changed to \${field_value}. \${message}</code>	Message template for correction oriented events that changed a field value in cases where a related US state context is known.
message.Context.B2B.Location	Location	When "context" can be supplemented with an additional Location context, this is the localization of the term "Location."
message.Context.B2B.EmptyOrNullField	null	Localization of the term used to convey a field value of empty or null.

All of the entries follow the pattern `message.*.B2B.*`. The particular message template used for a particular validation or correction event is selected based on the following criteria:

- is the event a validation or a correction?
- if it is a validation, does the event have a field element associated with the error?
- if it is a validation, is a required field missing or an invalid data value?
- is the data value on the event related to a US state?
- if it is a correction, does it reflect a data value removal or alteration?

The selection algorithm used to determine which message template to use for the validation contextual message is provided by the following method:

```
String getMessageTemplate(ValidationMapperParameters parameters);
```

While the one used to support correction events is the following:

```
String getMessageTemplate(CorrectionMapperParameters parameters);
```

The out of the box message templates contain several substitutable variables, each served up by a discreet `UploadMessageMapper` protected method. If the out of the box selection criterion needs to be expanded or altered, you must override either or both of the `getMessageTemplate()` methods and provide your own algorithm. The following is the list of the out of the box substitutable variables:

Substitutable Variable	Default Value	Protected <code>UploadMessageMapper</code> method(s) which serves up value
context	Derived only for validations and corrections that occur on entities that are configured on roster pages. The roster title, typically configured in the plural, is "singularized" [via protected method <code>makeSingular(String, Locale)</code> ]. Additional location context may be mapped into the resultant value based on the return value of the protected method <code>Index getRelatedLocationAggregateIndex(MapperParameters)</code> . For scalar atomic fields, this variable is converted to an empty string ("").	<ul style="list-style-type: none"> <li>• <code>String getContext(ValidationMapperParameters parameters);</code></li> <li>• <code>String getContext(CorrectionMapperParameters parameters);</code></li> <li>• <code>String getContext(Page page, MapperParameters parameters);</code> is the core algorithm that both the validation and correction variants delegate to internally.</li> </ul>
message	Sourced from the original message text associated with the originating validation or correction event.	<ul style="list-style-type: none"> <li>• <code>String getValidationMessage&gt;(ValidationMapperParameters parameters);</code></li> </ul>

<b>Substitutable Variable</b>	<b>Default Value</b>	<b>Protected UploadMessageMapper method(s) which serves up value</b>
		<ul style="list-style-type: none"> <li>String getCorrectionmessage&gt;(CorrectionMapperParameters parameters);</li> </ul>
field_label	Sourced from original field label associated with the originating validation or correction event.	<ul style="list-style-type: none"> <li>String getFieldLabel(ValidationMapperParameters parameters);</li> <li>String getFieldLabel(CorrectionMapperParameters parameters);</li> </ul>
field_value	Specifically, the current field value associated with the event. This is sourced from the current field value associated with the validation or correction event.	<ul style="list-style-type: none"> <li>String getFieldValue(ValidationMapperParameters parameters);</li> <li>String getFieldValue(CorrectionMapperParameters parameters);</li> </ul>
previous_field_value	Applies to correction events only when an alteration (not a removal) to a data value occurs. This is sourced from the previous field value associated with the correction event.	String getPreviousFieldValue(CorrectionMapperParameters parameters);
USState	<p>Applies to repeating data (roster based) events for both validation or correction events. By using the roster source, the base algorithm will try to find relevant StateProvCd elements for these candidates:</p> <ul style="list-style-type: none"> <li>WorkCompLineBusiness.WorkCompRateState</li> <li>CommlAutoLineBusiness.CommlRateState</li> <li>WorkCompLineBusiness.WorkCompIndividuals</li> <li>CommlInlandMarineLineBusiness.CommlIMInfo</li> </ul> <p>Additionally, if the roster source has a LocationRef non-null value, the algorithm will attempt to crawl to the related location aggregate and extract StateProvCd accordingly.</p>	String getUSState(MapperParameters parameters); (same method for both validation and correction events)

A placeholder method, void `getCustomProperties(String messageTemplate, Properties properties, MapperParameters parameters)`, is provided if additional substitutable variables are needed to map to the properties map as the resultant message is rendering.

If you determine a need to extend one or more of the `com.agencyport.upload.UploadMessageMapper` methods (option 2), the construction mechanism uses a factory design pattern where the name of the application extension is registered in the application properties. Applications can register a single extension that will handle all LOBs flowing through upload data manager or they can opt to have a discreet extension for each LOB. The `UploadMessageMapper.create(AgencyConnectConversionStepData, Object)` static factory method on the `UploadMessageMapper` class is invoked by the `UploadDataManager` constructor. This factory method requires your extension to implement the constructor variant that accepts `com.agencyport.connect.conversion.distributed`.

The following is an example of `AgencyConnectConversionStepData` followed by a `java.lang.Object` parameter:

## Example

```
public class MyMapper extends UploadMessageMapper { public MyMapper(AgencyConnectConversionStepData agencyConnectStepData,
Object messageOriginator) { super(agencyConnectStepData, messageOriginator); } protected String
normalizeMessage(ValidationMapperParameters parameters){ return super.normalizeMessage(parameters); } };
```

Supporting application property configuration follows:

```
Single extension for entire application
application.upload_message_mapper_class_name=com.carrier.upload.MyMapper

Specific extension for Workers Comp based transactions
WORK.upload_message_mapper_class_name=com.carrier.upload.MyMapper
```

If there are no \*.upload\_message\_mapper\_class\_name entries, then the base UploadMessageMapper class is used.

# Agency Connect Status Message

To derive various ExtendedStatusCd values of DataInvalid, DataMissing, VerifyDataAbsence and VerifyDataValue for the ACORD response, a type data member is included in the AgencyConnectStatusMessage structure. These values are limited to the following enumeration on the AgencyConnectStatusMessage class:

```
/** * The Type enumeration provides a way to distinguish messages by type. * @since 4.5.003 */ public enum Type { /** * The <code>VALIDATION_DATA_MISSING</code> value is used for validation messages for when a required field is missing. */ VALIDATION_DATA_MISSING, /** * The <code>VALIDATION_DATA_INVALID</code> value is used for validation messages when a value for a field is incorrect */ VALIDATION_DATA_INVALID, /** * The <code>CORRECTION_DATA_MISSING</code> value is used for correction messages for fields that were removed because the current value was invalid. */ CORRECTION_DATA_REMOVED, /** * The <code>CORRECTION_DATA_ALTERED</code> value is used for correction messages for fields that were altered because the original value was not valid and a system value was supplied. */ CORRECTION_DATA_ALTERED, /** * The <code>GENERAL</code> value is used for messages that don't fall into one of the above types. */ GENERAL }
```

The type variable in combination with the message severity (error, warning, info) should provide enough state to support a robust ACORD response. The version of AgencyConnect server you are using will dictate whether the AgencyConnectStatusMessage class imbedded in the AgencyConnect server recognizes the type XML element. Applications may need to resort to parsing the upload writer response on the AgencyConnect side if the type element is viewed as critical.

## Note

A final note on MissingElementPath for missing required fields. Due to the realities of framework views, special field helpers, ACORD simplified and since field labeling is a part of the messaging, it was decided not to supply this supplemental information.

# Sample Messaging for Validation Events

The following is a sample message for validation events:

```
<Message><Severity>Warning</Severity> <Category>Business</Category> <Text>Location 001, Vehicle 001, Symbol/Age is required in the state of VT</Text> <StepName>upload writer:com.agencyport.commlAuto.upload.CAUploadDataManager</StepName> <Type>VALIDATION_DATA_MISSING</Type> </Message><Message><Severity>Warning</Severity> <Category>Business</Category> <Text>Location 001, Vehicle 001, Registered State is required in the state of VT</Text> <StepName>upload writer:com.agencyport.commlAuto.upload.CAUploadDataManager</StepName> <Type>VALIDATION_DATA_MISSING</Type> </Message>
```

# Sample Messaging for Correction Events

The following is a sample a message for correction events:

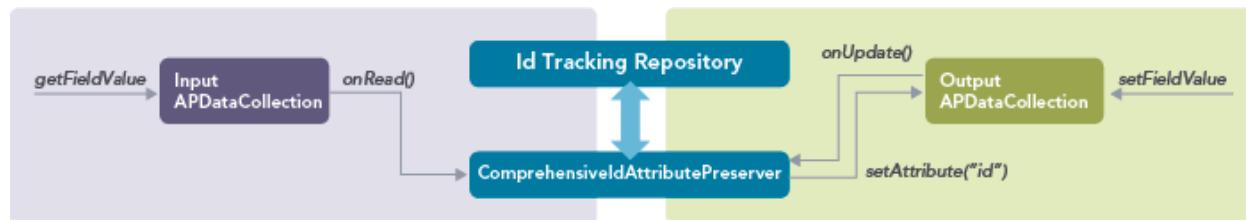
```
<Message><Severity>Informational</Severity> <Category>Business</Category> <Text>Location 001, Vehicle 001, 40/4 for the field
Symbol/Age for the state of VT was removed. Value was removed because no reasonable match could be found</Text> <StepName>upload
writer:com.agencyport.comnlAuto.upload.CAUploadDataManager</StepName> <Type>CORRECTION_DATA_Removed</Type>
</Message>
```

# Technical Solution

ID retention is carried out during the TDF based data transfer process rather than as a followup operation. The TDF based data transfer is primarily a set of read and write operations carried out against two separate APDataCollection instances. These two APDataCollection instances are managed by the upload data manager reading from the input APDataCollection and writing to the output APDataCollection.

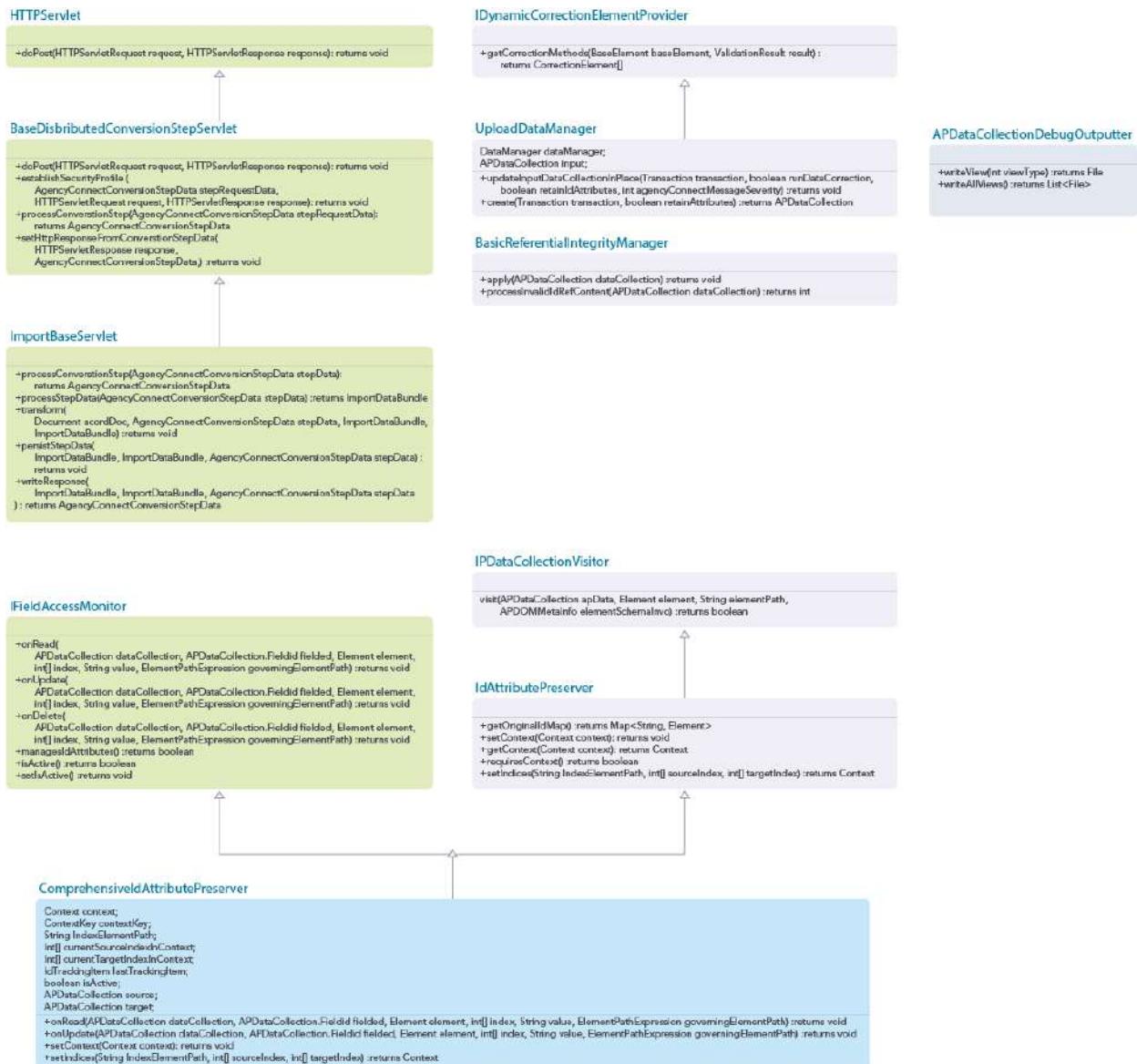
During the read operation, a set of intermediate HTML data containers are created to hold onto the data values that originate from the input data collection. During the write operation, the values contained within the HTML data containers are then written to the output data collection. A natural outgrowth of this design means that the data values extracted from the input data collection are limited to the XML elements that have an explicit definition in the TDF, regardless of whether a view or SFH is in play for that field. In other words, one or more XML elements sitting on the input APDataCollection are silently ignored and not transferred.

The ComprehensiveIdAttributePreserv class monitors both read operations upon the input APDataCollection and write operations applied to the output APDataCollection via the IFieldAccessMonitor interface. This class also supports the IIDattributePerserver interface. During read operations, this class tracks ID attribute values by holding onto XML elements from the input APDataCollection instance and associating these XML elements with their respective source ideas, data value, governing XPath and, most importantly, its source TDF field. As write operations are carried out, namely as new XML hierarchy is built, the ID attribute values corresponding to the current TDF field are applied to the XML hierarchy being build. The following diagram illustrates the collaboration between the input data collection, the comprehensive ID attribute preserver instance and the output data collection at a very high level:



## Class Model

The following diagram illustrates the two salient classes that play various roles during an upload writer/Turnstile request: ImportBaseProcessor and UploadDataManager. The diagram also includes two interfaces and one class that support the ID retention capability: IFieldAccessMonitor, IIDAttributePreserver and ComprehensiveIdAttributePreserver. The referential integrity class, BasicReferentialIntegrityManager, is illustrated since it is used to carry out the default referential integrity operations immediately before the TDF based data transfer.



## com.agencyport.domXML.IFieldAccessMonitor

The **IFieldAccessMonitor** interface provides a mechanism to monitor field access operations against **APDataCollection** instances. All methods are invoked AFTER the operation has taken place against **APDataCollection**. The various methods are listener only in nature and have no impact on the operation just executed.

## com.agencyport.domXML.changemanagement.ids.IIdAttributePreserver

The **IIdAttributePreserver** interface is the interface used by **APDataCollection** to preserver ID attribute values across a working document and a reference read only document.

## com.agencyport.domXML.changemanagement.ids.comprehensive.ComprehensiveIdAttributePreserver

The **ComprehensiveIdAttributePreserver** class provides a comprehensive algorithm for reclaiming ID attribute values, specifically while data values are moved between different XML documents. Extending this class is an advanced topic and should only be done after conferring with product development.

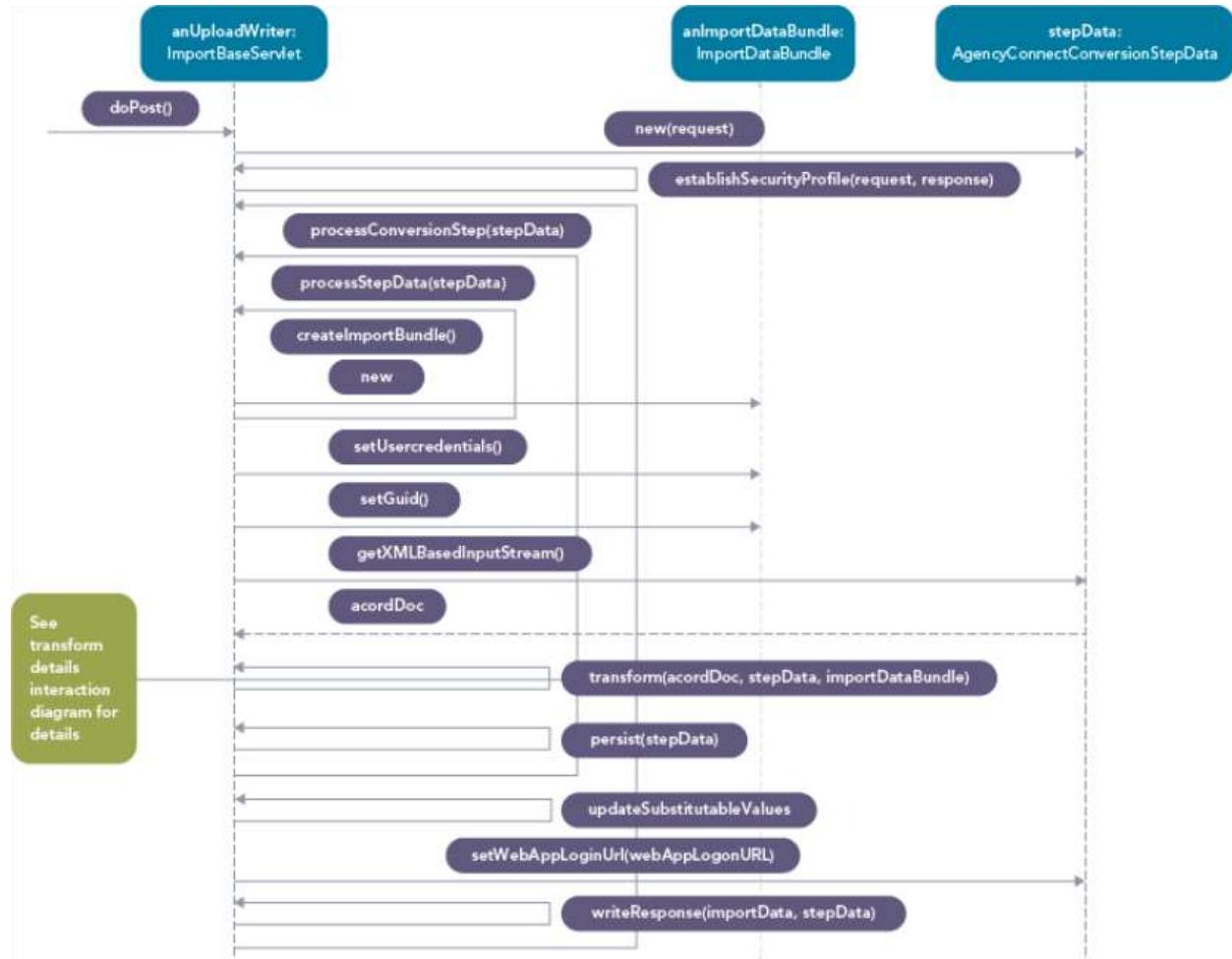
## **com.agencyport.domXML.widgets.APDataCollectionDebugOutputter**

The `APDataCollectionDebugOutputter` class is a handy class that can be used to write the eight various documents managed by `APDataCollection` to a series of files on the file system. `ImportBaseProcessor` engages this class to write out all eight managed documents to the file system as one of the very last things before returning to the upload writer caller.

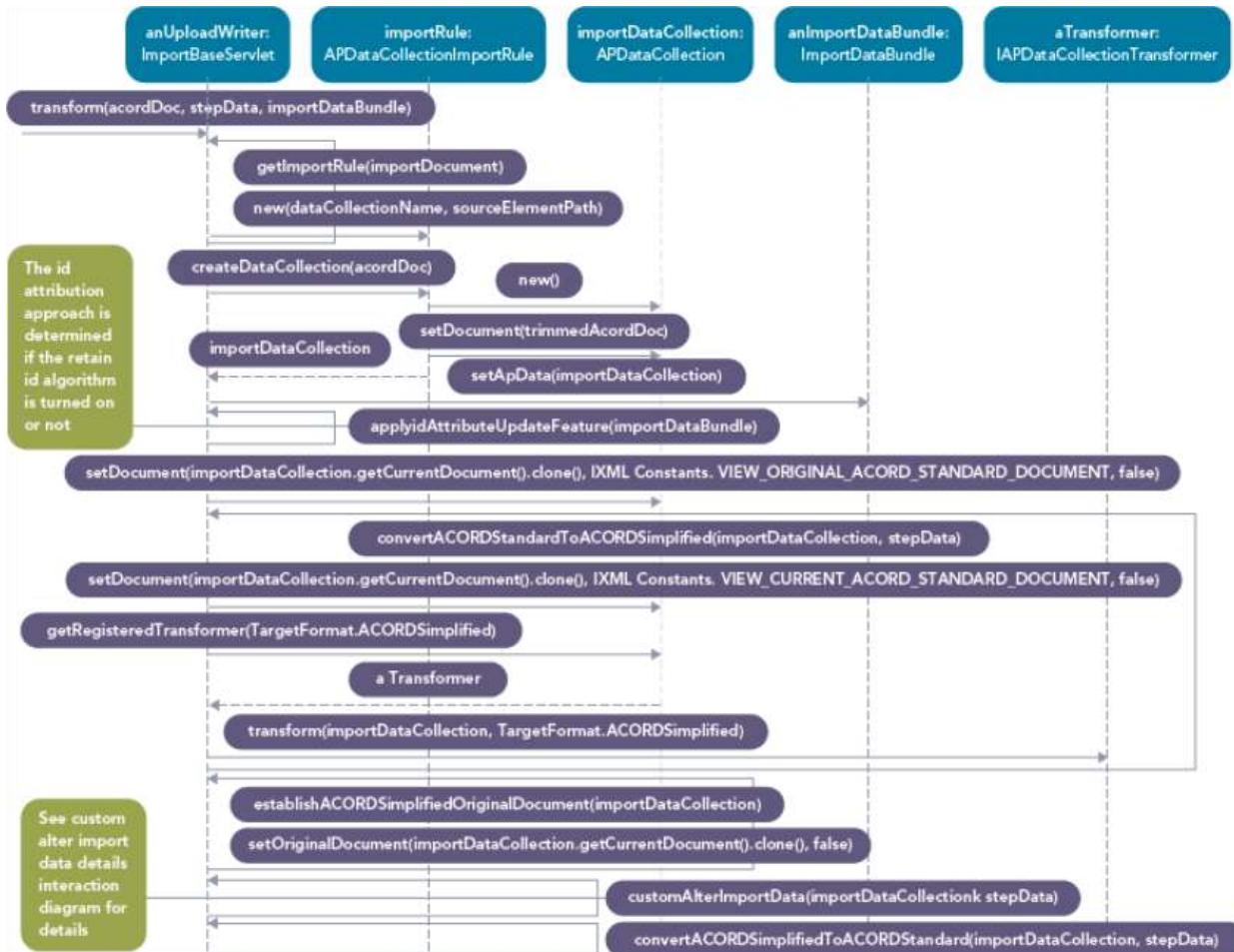
# Sequence Diagrams

The following are sequence diagrams, starting from the highest level to the lowest level:

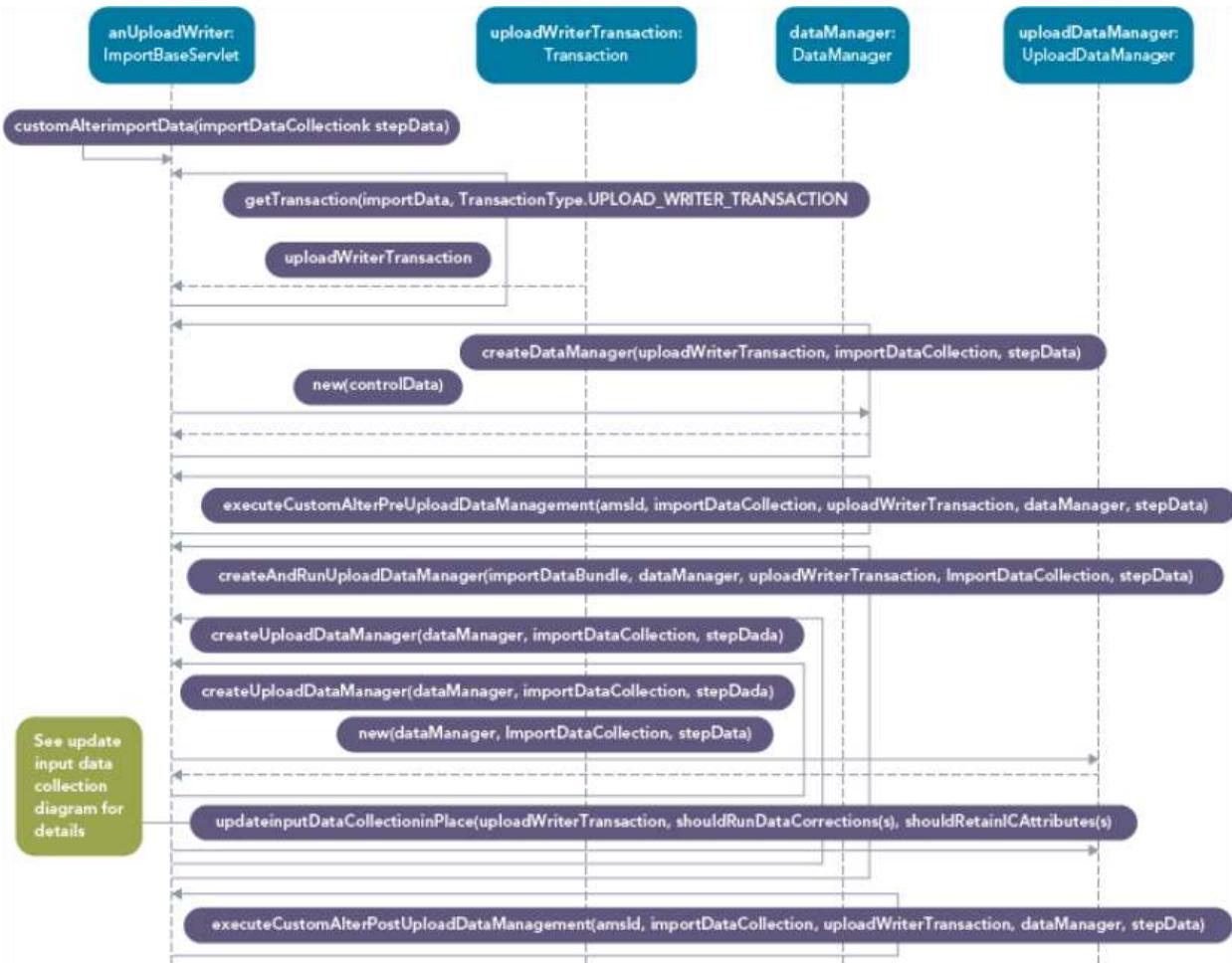
## Upload High-Level



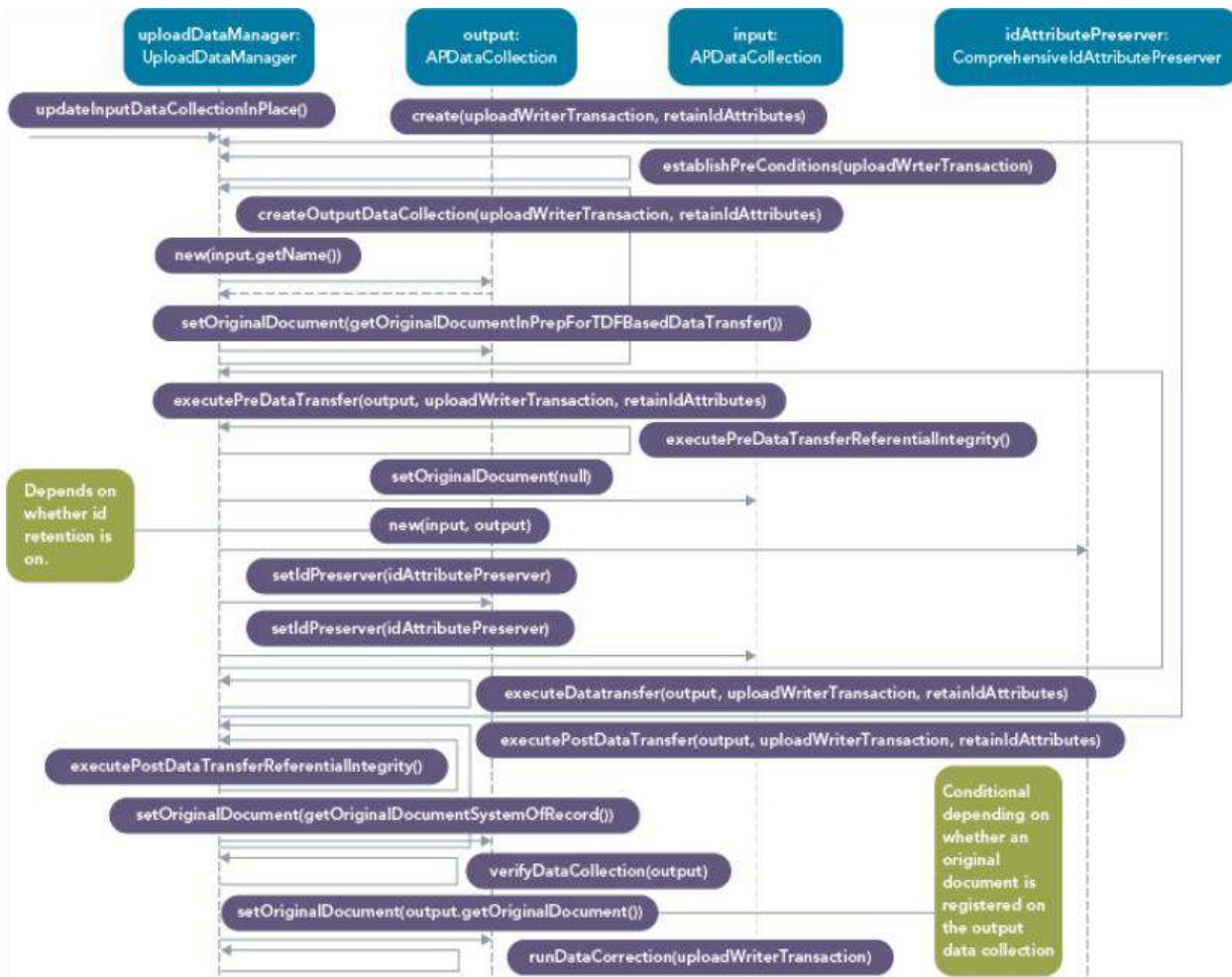
## Transform Details



## Custom Alter Import Data



## Update Input Data Collection



# Configuration Options

This section outlines the mechanics for altering the various aspects of upload writer functionality from its default out of the box functionality. Many of the aspects can be altered entirely through configuration, which should always be a goal of the application development team. Other aspects may require extension to achieve the desired end state. The following table provides an inventory of some of the aspects of upload writer functionality that are alterable, the default behavior and how to modify that behavior.

#	Aspect	Out of the box behavior	Alteration Options	Example	Additional Comments and Constraints
1	Establish the security profile	A security profile is built from the user credentials accompanying the inbound request XML payload using standard ACSI interfaces.	First, exhaust the possibility of establishing the security profile at the security filter level. Second, determine why the out of box implementation is not adequate. Perhaps a slight adjustment to your <code>SecurityProfileManager</code> is all that is needed. Lastly, extend the <code>BaseDistributedConversionStepProcessor.setEstablishSecurityProfile</code> method as a last resort.	N/A	Preferably and optimally, the security filter has already established the security profile using ACSI cookies.
2	Transforming the incoming XML document to maximize the amount of data that is not dropped on the floor	Invokes a registered target ACORD Simplified transformer from <code>ImportBaseProcessor.convertACORDStandardToACORDSimplified</code>	First, consider packaging any logic of this nature into a bonafide registered artifact. Lastly, extend <code>ImportBaseProcessor.convertACORDStandardToACORDSimplified</code> and/or <code>ImportBaseProcessor.executeCustomAlterPreUploadDataManagement</code> methods	N/A	
3	Transforming the outgoing XML document back to its ACORD standard format	Invokes a registered target ACORD Simplified transfer from <code>ImportBaseServlet.convertACORDSimplifiedToACORDStandard</code>	First, consider packaging any logic of this nature into a bonafide registered transformer artifact. Lastly, extend the <code>ImportBaseServlet.convertACORDSimplifiedToACORDStandard</code> and/or <code>ImportBaseServlet.executeCustomAlterPostUploadDataManagement</code> methods	N/A	

#	Aspect	Out of the box behavior	Alteration Options	Example	Additional Comments and Constraints
4	Capturing the various XML documents managed by APDataCollection to disk	Determines the existence of an application property named <code>debug_upload_transformations_dump_path</code> . This property, if present, dictates the directory that all views managed by the outgoing APDataCollection are written to.	<p>Configuration via application property:</p> <pre>debug_upload_transformations_dump_path</pre> <p>If the application property is missing, then this feature is turned off.</p> <p>Programmatically via the protected method:</p> <pre>ImportBaseProcessor.writeAPDataToDebugOutput</pre>	<pre>debug_upload_transformations_dump_path= \${output_dir}upload_transformations</pre>	
5	Controlling which original document is selected as "system of record" original document	<p>This is controlled by a combination of internal state and TDF/LOB/application property named <code>original_document_system_of_record</code>.</p> <p>If the upload writer TDF transaction does not support original values via the method <code>Transaction.supportsOriginalValues()</code>, then the application property is ignored and the original document is nulled out.</p> <p>If the ID retention parameter is true and the value of this property is <code>front_door_document</code>, then the original "front door" document is used; otherwise, a clone of the current document before data corrections serves as the system of record original document.</p>	<p>Configuration via TDF/LOB/application property:</p> <pre>original_document_system_of_record</pre> <p>Valid values:</p> <ul style="list-style-type: none"> <li>• <code>front_door_document</code></li> <li>• <code>current_document_before_data_corrections</code> (default value)</li> </ul> <p>Programmatically via the protected method:</p> <pre>UploadDataManager.getOriginalDocumentSystemOfRecord</pre>	<pre>application.original_document_system_of_record=front_door_document</pre>	<p>If an application needs to vary this setting by LOB or TDF, then the TDF based application property has precedence over the LOB based application property, which has precedence over the application based application property.</p>
6	Turning on/off the ID retention feature	This is controlled by a Boolean TDF/LOB/application property named <code>should_retain_id_attributes</code> .	<p>Configuration via TDF/LOB/application property:</p> <pre>should_retain_id_attributes</pre> <p>Valid values:</p> <ul style="list-style-type: none"> <li>• <code>true</code></li> </ul>		<p>During development/debugging phase, additional debugging information will be sent to the log by the <code>ComprehensiveIdAttributePreserver</code> class if the logging level is increased to the FINE level.</p> <pre>com.agencyport.domXML.changemanagement.ids</pre>

#	Aspect	Out of the box behavior	Alteration Options	Example	Additional Comments and Constraints
			<ul style="list-style-type: none"> <li>• false (default value)</li> </ul> <p>Programmatically via the protected method:  <code>ImportBaseProcessor.ShouldRetainAttributes</code></p>		.comprehensive.level =FINE
7	Determination of work item ownership	Work item ownership given to the current user and the primary user group of the current user identified on the security profile	<p>Programmatically via the protected method:  <code>ImportBaseProcessor.determineWorkItemOwnership</code></p>	N/A	

# Best Practices for an Effective Upload Writer Development Exercise

The following are best practices and guiding principals for an effective upload writer development exercise:

Write the Minimum Code Necessary to Satisfy Business Requirements

Application teams should:

- Review/re-engineer previous upload writer implementations, when appropriate, to take full advantage of the refactoring of the `ImportBaseProcessor`, `CommonUploadWriter` and `UploadDataManager` classes.
- Consider packaging custom transformation logic (XLST or Java) as a registered transformer so that the upload framework can automatically engage it as part of the default process.
- Leverage as much product code as possible.

Beware of Thread Safety and the Stateful `ImportBaseProcessor` Class

The `ImportBaseProcessor` class indirectly inherits from `javax.servlet.http.HttpServlet`, implying that storing any request level data values on instances of this class are a complete "no-no."

If you need to store request level data values spanning function callout points, you should not add them to your `ImportBaseProcessor` extension since this will compromise the accuracy while concurrent upload writer requests are being serviced. Instead, you should extend the `com.agencyport.agencyconnect.ImportDataBundle` class and place any custom data members within the class.

The `ImportBaseProcessor.createImportBundle()` factory method needs to be overridden in the application's `ImportBaseProcessor` extension to facilitate the instantiation of such extended import data bundle class. If you need to reach this instance within a function where the reference to the current import data bundle is not passed as an explicit parameter, you can use the `ImportBaseProcessor.getImportDataBundle()` method to regain addressability.

Conversely, a unique `UploadDataManager` instance is created for each upload writer request serviced so this concern does not apply to extensions of this class.

Turning on the Id Retention Option

ID retention adds a certain amount of overhead to the TDF based data transfer process. It only needs to be on if (and only if) a comprehensive XML change management document with `ModInfo` or `ChangeStatus` segments is required for the upload writer.

Page Composition Should Rely Solely on DTR Rather than Custom JavaScript

Page images are needed to satisfy the read operations carried out by the TDF based data transfer process. The non-excluded fields on the page images govern which fields to extract from the incoming XML document. Service side DTR is engaged to generate those page images; JavaScript is not engaged. Therefore, what previously worked well for a manually entered work item can quickly fall apart as upload is introduced. Unfortunately, upload is many times an "after thought" on projects.

After this guiding principle becomes self evident, the job of refactoring custom JavaScript over to DTR behavior equivalence becomes an unexpected and unfortunate extra line item on the application project plan, adding unnecessary additional cost to the project.

Augmenting the XML Document with Additional Data

You may need to add additional data values to the XML document, especially to support requests that are headless and require an XML response document to be returned to the calling agent (e.g., real time rating requests). Service calls, such as address cleansing, MVR, Clue, Choicepoint and rating engine calls are a few examples to note.

Important

Custom application code that is called upon to apply data values returned from external calls to the output XML document should transfer data values one by one, allowing the `APDATACollection` class to determine whether the update operation is truly an insert or update scenario. Custom code needs to be very careful not to delete existing XML ID attributes of any existing parent XML aggregates that are reflected in the original document.

Custom code that effectively deletes and then re-adds XML aggregate groups with corresponding original document XML aggregates will most likely result in a change management report that does not reflect reality as well as it could. As a goal, avoid the "delete/add" syndrome.

The standard TDF based data transfer process does not execute display or process side connectors as part of the read page/write page operations. Therefore, the invocation of these "data enhancement" service calls will need to be handled with application custom code elsewhere. The placement of this custom code depends on where the respective service calls need to be made in relation to set of operations carried out by the framework. The following describes the most frequent set of possibilities:

- **Augmenting the data collection before the TDF based transfer process executes** - you may determine that one or more of these services may need to be called before the TDF based data transfer process begins. There is an assumption that these additional values would be transferred to the `input` document have been accounted for in the TDF as discreet fields that they are automatically transferred over to the output document during the follow up TDF based data transfer process. Assuming these assumptions are true, then there are a couple of potential places to locate this code:
  - o `ImportBaseProcessor.executeCustomAlterPreUploadDataManagement()`
  - o `UploadDataManager.executePreDataTransfer() [make sure you call super.executePreDataTransfer() before returning]`
- **Augmenting the data collection while the TDF based data transfer process executes** - you may determine that one or more of these services may need to be called while the TDF based data transfer process is executing. To achieve this, you need to override the `UploadDataManager.update(APDataCollection, PageDataContainer)` method.

```
@Override protected void update(APDataCollection output, PageDataContainer pageDataContainer) throws APEException{ //Execute services that need to be run before the data in the page data container is transferred to the output data collection //Transfer the data by calling the framework. super.update(output,pageDataContainer); // Execute services that need to be run after the data in the page data container has been transferred to the output data collection }
```

- **Augmenting the data collection after the TDF based data transfer process concludes, but before the data corrections run** - you may determine that one or more of these services may need to be called after the TDF based data transfer process executes, but before data corrections run. The best way to achieve this is to extend the `UploadDataManager.executePostDataTransfer()` method. Ensure that you delegate to the framework's version of this method before returning from this override.

```
protected void executePostDataTransfer(APDataCollection output, Transaction transaction, boolean retainIdAttributes) throws APEException { // Call services // Delegate to framework super.executePostDataTransfer(output,transaction,retainIdAttributes); }
```

- **Augmenting the data collection after data correction runs, but before the work item is persisted** - you may determine that one or more of these services may need to be called after data correction runs. The best place to locate this code is in `ImportBaseProcessor.executeCustomAlterPostUploadDataManagement()`.

Returning an XML Response via Conversion Step Data Structure

Returning an XML response is required in certain situations when the upload writer is asked to carry out a headless transaction, such as a real time rating call. The best place to locate the code is to extend the `ImportBaseProcessor.writeResponse()` method.

The following is a code example that returns the comprehensive merged document back to the caller:

```
/* (non-Javadoc) * @see com.agencyport.agencyconnect.ImportBaseProcessor#writeResponse(com.agencyport.agencyconnect.ImportDataBundle, com.agencyport.connect.conversion.distributed.AgencyConnectConversionStepData) */ @Override protected AgencyConnectConversionStepData writeResponse(ImportDataBundle importDataBundle, AgencyConnectConversionStepData stepData) throws APEException { //Engage the framework's logic first AgencyConnectConversionStepData responseStepData = super.writeResponse(importDataBundle, stepData); // Send the comprehensive merge document back to the caller APDataCollection output = importDataBundle.getApData(); MergeManager mergeManager = output.createMergeManager(IXMLConstants.VIEW_ORIGINAL_DOCUMENT, IXMLConstants.VIEW_CURRENT_DOCUMENT); int savePoint = output.changeMergeViewType(mergeManager, IXMLConstants.VIEW_MERGED_DOCUMENT); try{ responseStepData.setOutputStream(output.getXMLDocument()); } catch (IOException ioException){ throw APEException.create(ioException); } finally { // Restore view in context output.changeViewType(savePoint); } return responseStepData; }
```

Returning an XML Response with Change Status Segments via Conversion Step Data Structure

The following is a variation that returns `ChangeStatus` segments instead of `ModInfo` segments, which is also an extension to the `ImportBaseProcessor.writeReponse()` method.

```
/* (non-Javadoc) * @see com.agencyport.agencyconnect.ImportBaseProcessor#writeResponse(com.agencyport.agencyconnect.ImportDataBundle, com.agencyport.connect.conversion.distributed.AgencyConnectConversionStepData) */ @Override protected AgencyConnectConversionStepData writeResponse(ImportDataBundle importDataBundle, AgencyConnectConversionStepData stepData) throws APEException { //Engage the framework's logic first AgencyConnectConversionStepData responseStepData = super.writeResponse(importDataBundle, stepData); // Send the comprehensive merge document back to the caller via a true response with change status segments. APDataCollection output = importDataBundle.getApData(); //Set up current and original response documents on the response data collection. APDataCollection response = APDOMFactory.createDataCollection("PersAutoPolicyQuoteInqRs"); response.setDocument(getResponseEquivalentDocument(output.getDocument())); response.setDocument(getResponseEquivalentDocument(output.getDocument(IXMLConstants.VIEW_ORIGINAL_DOCUMENT)),
```

```

IXMLConstants.VIEW_ORIGINAL_DOCUMENT, false); // Allocate a merge manager to generate the change status segments. MergeManager
mergeManager = new MergeManager("ChangeStatus", "MsgStatus", response, IXMLConstants.VIEW_ORIGINAL_DOCUMENT,
IXMLConstants.VIEW_CURRENT_DOCUMENT); int savePoint = response.changeMergeViewType(mergeManager,
IXMLConstants.VIEW_MERGED_DOCUMENT); try{ responseStepData.setOutputStream(response.getXMLDocument()); } catch
(IOException ioException){ } throw APEXception.create(ioException); } finally { // Restore view in context
output.changeViewType(savePoint); } return responseStepData; }/** * Clones a document, changes its root element to
PersAutoPolicyQuoteInqRs and then * returns it. * @param document is the document to clone. * @return the response version of the incoming
document */ private Document getResponseEquivalentDocument(Document document){ Document response = (Document) document.clone();
response.getRootElement().setName("PersAutoPolicyQuoteInqRs"); return response; }

```

Returning XML Response with Change Status Segments via Conversion Step Data Structure Based on a Different Original Document than the Original

Situations may arise when the original document used as the basis for a change management report back to the upload writer client needs to be different than the one that becomes the original document system of record. The following example illustrates this technique:

#### Note

The UploadDataManager class is extended and its `getOriginalDocumentSystemOfRecord()` method is overloaded so that it can save the front door original document, but return a clone of the current document that reflects the state of the XML document immediately before data corrections run.

```

package com.agencyport.persauto.servlets; import java.io.IOException; import org.jdom.Document; import
com.agencyport.agencyconnect.AgencyManagementSystemId; import com.agencyport.agencyconnect.ImportBaseProcessor; import
com.agencyport.agencyconnect.ImportDataBundle; import com.agencyport.connect.conversion.distributed.AgencyConnectConversionStepData;
import com.agencyport.data.DataManager; import com.agencyport.domXML.APDOMFactory; import
com.agencyport.domXML.APDataCollection; import com.agencyport.domXML.Aggregate; import
com.agencyport.domXML.ElementPathExpression; import com.agencyport.domXML.IXMLConstants; import
com.agencyport.domXML.IdIterator; import com.agencyport.domXML.changemanagement.MergeManager; import
com.agencyport.shared.APEXception; import com.agencyport.trandef.Transaction; import com.agencyport.upload.UploadDataManager; public
class PersAutoUploadWriter extends ImportBaseProcessor { /** * The PAUploadDataManager class */ static public class
PAUploadDataManager extends UploadDataManager { /** * The <code>frontDoor</code> is a place to hold onto the original document * that
we use for the purposes of creating the comprehensive change management report which * is returned to the application which invoked this
upload writer. */ private Document frontDoor; /** * @param dataManager * @param input * @throws APEXception */ public
PAUploadDataManager(DataManager dataManager, APDataCollection input) throws APEXception { super(dataManager, input); } /** *
@param dataManager * @param input * @param stepData * @throws APEXception */ public PAUploadDataManager(DataManager
dataManager, APDataCollection input, AgencyConnectConversionStepData stepData) throws APEXception { super(dataManager, input,
stepData); } /** * Override so that we can utilize one original document as the permanent system of record * but hold onto a different document
for the purposes of the comprehensive change management * report returned to the upload writer client. * @see
com.agencyport.upload.UploadDataManager#getOriginalDocumentSystemOfRecord(com.agencyport.trandef.Transaction,
com.agencyport.domXML.APDataCollection, boolean) */ @Override protected Document getOriginalDocumentSystemOfRecord(Transaction
transaction, APDataCollection dataCollectionAfterUploadDataManagerRan, boolean retainIdAttributes) throws APEXception { // Here we
assume that the framework will return the front door document by the way we have configured // this application. frontDoor =
super.getOriginalDocumentSystemOfRecord(transaction, dataCollectionAfterUploadDataManagerRan, retainIdAttributes); // Return a clone of
the current document for the system of record. return (Document) dataCollectionAfterUploadDataManagerRan.getCurrentDocument().clone(); }
/** * @return the front door document */ public Document getFrontDoorDocument() { return frontDoor; } } /** * The
<code>serialVersionUID</code> */ private static final long serialVersionUID = 3986496925077019386L; /* (non-Javadoc) * @see
com.agencyport.agencyconnect.ImportBaseProcessor#getPropertyNamePrefix() */ protected String getPropertyNamePrefix(int workItemId,
String conversionGuid, APDataCollection apData) { return "PERSAUTO"; } /* (non-Javadoc) * @see
com.agencyport.agencyconnect.ImportBaseProcessor#writeResponse(com.agencyport.agencyconnect.ImportDataBundle,
com.agencyport.connect.conversion.distributed.AgencyConnectConversionStepData) */ @Override protected
AgencyConnectConversionStepData writeResponse(ImportDataBundle importDataBundle, AgencyConnectConversionStepData stepData)
throws APEXception { // Engage the framework's logic first AgencyConnectConversionStepData responseStepData =
super.writeResponse(importDataBundle, stepData); // Send the comprehensive merge document back to the caller via a true response with
change status segments. APDataCollection output = importDataBundle.getApData(); // Set up current and original response documents on the
response data collection. APDataCollection response = APDOMFactory.createDataCollection("PersAutoPolicyQuoteInqRs");
response.setDocument(getResponseEquivalentDocument(output.getDocument())); // Get the front door original document
PAUploadDataManager uploadManager = (PAUploadDataManager) importDataBundle.getUploadDataManager();
response.setDocument(getResponseEquivalentDocument(uploadManager.getFrontDoorDocument()));
IXMLConstants.VIEW_ORIGINAL_DOCUMENT, false); // Allocate a merge manager to generate the change status segments. MergeManager
mergeManager = new MergeManager("ChangeStatus", "MsgStatus", response, IXMLConstants.VIEW_ORIGINAL_DOCUMENT,
IXMLConstants.VIEW_CURRENT_DOCUMENT); int savePoint = response.changeMergeViewType(mergeManager,
IXMLConstants.VIEW_MERGED_DOCUMENT); try{ responseStepData.setOutputStream(response.getXMLDocument()); } catch
(IOException ioException){ } throw APEXception.create(ioException); } finally { // Restore view in context output.changeViewType(savePoint);
} return responseStepData; }/** * Clones a document, changes its root element to PersAutoPolicyQuoteInqRs and then * returns it. * @param
document is the document to clone. * @return the response version of the incoming document */ private Document
getResponseEquivalentDocument(Document document){ Document response = (Document) document.clone();
response.getRootElement().setName("PersAutoPolicyQuoteInqRs"); return response; } /* (non-Javadoc) * @see

```

```
com.agencyport.agencyconnect.ImportBaseProcessor#createUploadDataManager(com.agencyport.data.DataManager,
com.agencyport.domXML.APDataCollection, com.agencyport.connect.conversion.distributed.AgencyConnectConversionStepData)*/ protected
UploadDataManager createUploadDataManager(DataManager dataManager, APDataCollection apData, AgencyConnectConversionStepData
stepData) throws APException { return new PAUploadDataManager(dataManager, apData, stepData); } }
```

# Real Time Rating (RTR)

The Agencyport Real Time Rating (RTR) feature provides a business workflow where risk can be rated using data collected through a comparative rater. The risk data usually is in ACORD XML format. The rate is usually compared to the rate obtained from a competing carrier and the business is placed with the carrier that gives the favorable rates.

During RTR processing, the following occurs:

- The risk data is enhanced/augmented with the data obtained from an external source
- Bumping and slotting of limits and deductibles occurs
- Data is validated
- The rating system is engaged to obtain a rate

This RTR feature enhancement within AgencyPortal and the Agency System Integration Kit provides out of the box support for the validation of rate bearing fields/hard stops and executes data augmentation and rating connectors. At the end of the RTR process, a response will be returned with premiums, any change statuses for modified data and a link to the work item that is created.

When implementing an AgencyPortal application to collect data during an agent workflow, integration with external systems is done for validation/data augmentations/rating. AgencyPortal's architecture supports external system integration through its connector infrastructure. Connections are defined and registered for user workflow and are not engaged for upload/RTR workflows. It is common for developers to write custom upload writers to leverage the same connections that are defined for user workflow.

These upload writers are designed to consume text/XML media types only. Developers must devise custom solutions to receive supporting attachments along with the text/XML risk data. During RTR, quote letters are generated. These letters have to be sent to the calling application so that they can display to the user.

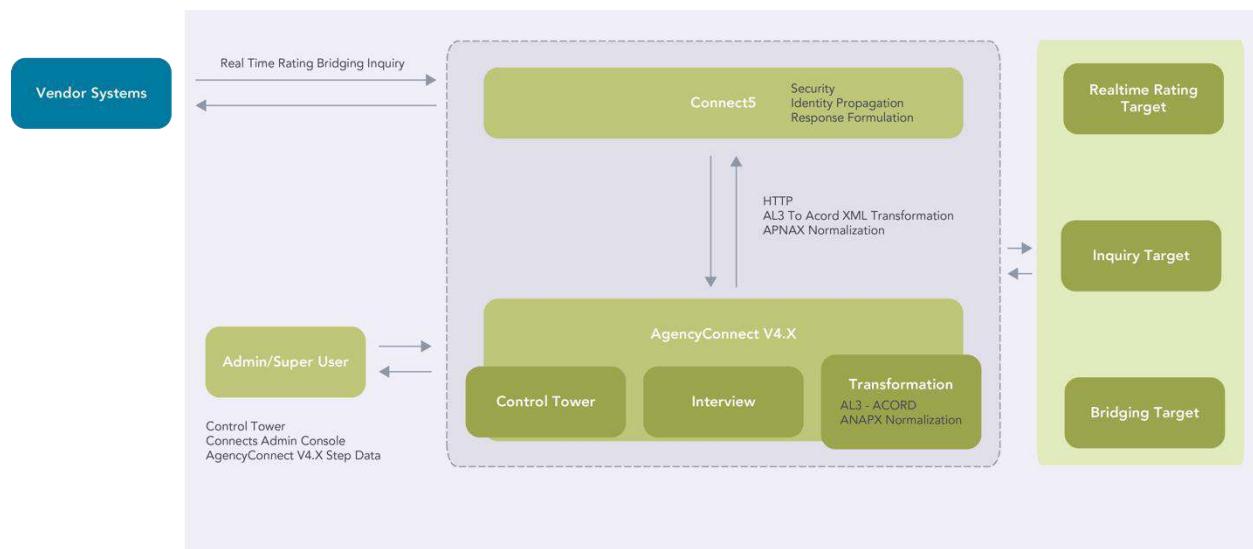
## Note

This feature contains the following assumptions:

A service based on REST was developed to serve the bridging and RTR workflow. This REST service delegates the processing to existing upload writers. Developers can continue to extend customize upload writers as they currently do. Upload writers do not inherit HttpServlet; it's a POJO.

Connect5 will be the broker layer that negotiates data formats between external systems and AgencyPortal.

## Architecture



## Vendor Systems

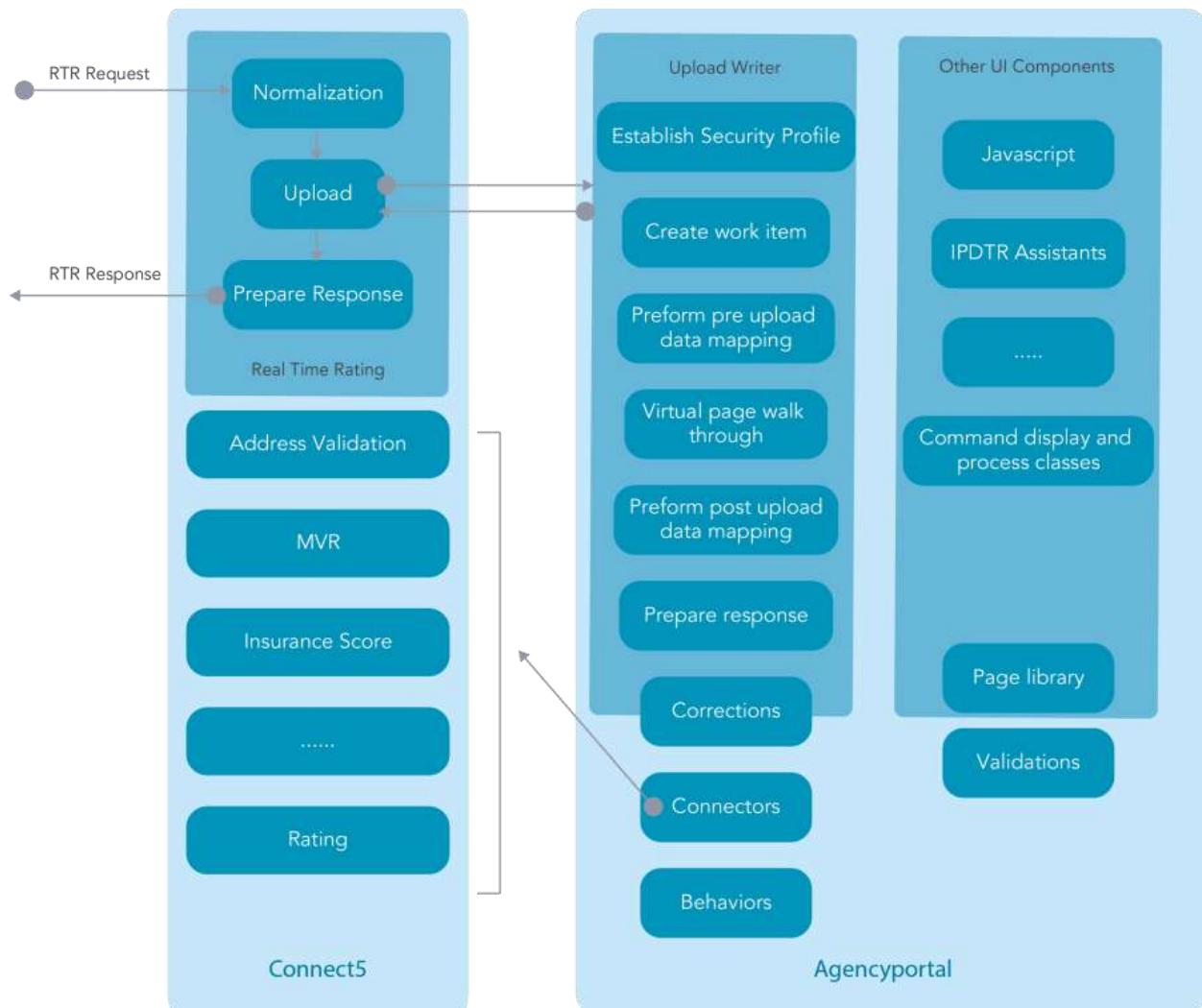
Vendor systems provide the bridging and real-time systems portion of this feature. Transformation Station, Transaction NOW, WebLink and Comparative Rating Vendors allow agents to:

- Obtain quote/rate from a carrier for comparative purposes or otherwise from a client file within their agency management system.
- Bridge a quote to their carrier POS systems where they can review the quote order reports and obtain a quote.
- Inquire to gain access to carrier information, such as billing, policy view, etc, on the client.

The vendor systems will send ACORD standard messages pertaining to the business flow.

## Connect5 to AgencyPortal

The following outlines the communications that occur between Connect5 and AgencyPortal.



## Supporting Data Structures

The following data structures are supported by RTR:

- AgencyConnect Data Step - The AgencyConnectStepData message structure is the message contract between Connect5 and AgencyPortal. This message schema conveys other operations to AgencyPortal.
- AgencyConnect Request/Response - The AgencyConnect Request/Response message structure is the message contract between Connect5 and the AgencySystems integration kit. Refer to the AgencySystems section in the Integration Kit documentation for more information.
- ACORD Message

## Resources and Artifacts

### Java Classes

BaseDistributedConversionStepProcessor	Used as a base class for specific distributed conversion step processors. This class uses the AgencyConnectConversionStepData class as a means to transfer the conversion step data to the derived class. Instances of this class can be shared across threads/requests, so any deviations should not store a request state as data members directly or indirectly.
ImportBaseProcessor	The base class for importing ACORD based data into an Agencyport SDK based application. It is typically the endpoint for an AgencyConnect transaction.  Refer to the <a href="#">ImportBaseProcessor and UploadDataManager Classes</a> topic for more information.

## RTR Configuration

The following section details what's needed to configure and set up RTR transaction processing in AgencyPortal.

### Upload Writer

Bridging/upload is a business workflow where risk from external sources is flooded into an agency portal (POS) system without having to re-key data. This allows users to walk through the application pages to fix any issues and submit a quote. Risk data from an agency management system is then exported as ACORD XML/ACORD AL3/ACORD forms and converted/normalized to APNAX (Agencyport Normalized ACORD XML) before flooding into AgencyPortal. AgencyPortal exposes J2EE servlet interfaces called Upload Writers to receive the data.

In order for this feature to work, an Upload Writer headless transaction is converted to REST API. This allows AgencyPortal to support both bridging and real time rating headless transactions via the AgencyConnectStepData request structure. The following also occurs within AgencyPortal to support this:

- The AgencyConnectStepData request supports both bridging and real time rating headless transactions.
- The connector infrastructure provides the ability to specify connectors that need to be executed for real time and bridging connectors.
- Executing transaction validation and returning pertinent messages back in a response is supported.
- Performing automatic data corrections and the return of pertinent details back in a response is supported.
- Return responses are decorated with premiums from rating calls and XML by other data augmentation connectors if requested.

The upload writer REST API end point is available via <http://host:port/agencyportal/api/uploadwriter>.

The following classes are included to support this feature:

- [BaseDistributedConversionStepProcessor](#)
- [ImportBaseProcessor](#)

### Upload Writer Request

Upload request supports RTR transactions, not just regular bridging transactions, with the use of an aggregate operation.

Name	Description	Valid
Operation type	Indicates the type of upload transaction being performed	Bridge Realtime Inquiry
Submission	Entity being submitted through this transaction	Account Quote Claim
Submission ID	Optional identifier for submission entity	Integer

Name	Description	Valid
Submission ID type	Identifies the ID type	Account Workkit

## Example

The following is a sample operation node in an upload request:

```
<Operation type="Bridge"> <Submission>Quote</Submission> <ID type="Account">0</ID> </Operation> <Operation type="RealtimeRating"> <Submission>Quote</Submission> </Operation>
```

The following is a sample of an upload writer request structure:

```
<AgencyConnectConversionStep name="upload"> <InputStream> <![CDATA[<ACORD> <SignonRq> <SignonPswd> <CustId> <SPName/> <CustPermid/> <CustLoginId/> <CustId> <CustPswd> <EncryptionTypeCd/> <Pswd/> <CustPswd> <SignonPswd> <ClientDt/> <CustLangPref/> <ClientApp> <Org>MULTICO Rating Systems</Org> <Name>MULTICO -online Premium Comparative System</Name> <Version>1.4.151</Version> <ClientApp> <SignonRq> <InsuranceSvcRq> <RqUID/> <PersAutoPolicyQuoteInqRq> ... </PersAutoPolicyQuoteInqRq> <InsuranceSvcRq> </ACORD>]]> <InputStream> <ConversionProcessDescriptor/> <ConversionGUID>bb18bbc5-7e7b-459e-928b-a84291f53c73</ConversionGUID> <Signon> <UserId/> <Password>822gRz5U+mo=</Password> <Signon> <Operation type="RealtimeRating"> <Submission>Quote</Submission> </Operation> </AgencyConnectConversionStep>
```

## Connectors

This feature enables connectors to be defined in the TDF and executed during a real time rating transaction. This is done by using the `RealtimeRating` user action type as a valid value for the `executeWhen userAction` attribute.

### Display Side Connectors

Display side connectors are always run when a user navigates through pages. During upload, they run based on the `executeWhen useraction`. For example, if `executeWhen` has the `RealtimeRating` user action, it is executed for a real time rating transaction not as an upload bridge transaction. If connectors should only be run during an upload and not during page navigation, then these connectors must be excluded using behaviors.

### Process Side Connectors

Process side connectors always run according to what is set for the `executeWhen useraction`.

## Example

The following is a sample connector configuration for real time rating:

```
<connectors type="process"> <connectortype="custom" supportsPrevalidation="false" className="com.agencyport.intkit.core.upload.realtimerating.RealTimeRatingSimulator" scope="aggregate"> <executeWhen userAction="RosterAdd" /> <executeWhen userAction="RosterEdit" /> <executeWhen userAction="RealtimeRating" /> </connector> </connectors>
```

In this example, the connector will be executed for the following:

- a real time rating transaction
- a roster add and roster edit during page navigation

The connector will NOT be executed for continue of a roster page since it is marked as "aggregate".

## Built-In Precondition

The `uploadWriterOperationType` built-in precondition can be used in behaviors to exclude/include connectors.

## Example

The following is an example

```
<behavior> <do action="exclude"/> <where> <preConditionname="uploadWriterOperationType">!Realtime Rating</preCondition> </where>
<for>*/policySummary/process/realtimerating</for> </behavior>
```

Refer to the [Connectors](#) topic for more information on Connectors.

## Rating Connector Utilities

The following methods added to the `ConnectorManager` class allow you to call TVR and perform data corrections:

- `runTransactionValidation`
- `runDataCorrections`

### TVR Filtering

Since upload processing is ran using a full new business transaction and not a quick quote transaction, you must ignore the TVR results that are not related to the fields on the quick quote during a real time rating scenario. You can do this by using the `TransactionBaseValidationResultFilter` class, which allows you to filter out all validation results from a TVR that are not part of a specified transaction.

The following code removes validation results that are not part of a quick quote:

```
if (!IWebsharedConstants.QUOTE_TRANSACTION_TYPE .equals(fullappTransaction.getType())){ Transaction qqTransaction =
TransactionDefinitionManager.getTransaction(fullappTransaction.getLobCode()),
IWebsharedConstants.QUOTE_TRANSACTION_TYPE); if (qqTransaction != null){ tvr.setValidationResultFilter(new
TransactionBasedValidationResultFilter(qqTransaction, fullappTransaction)); } }
```

### Transfer Messages

Messages generated during connector execution can be transferred to upload response using the following:

```
UploadMessageMapper uploadMessageMapper = UploadMessageMapper.create(null, this); for (ValidationResult result :
tvr.getEntireDetailValidationResultSet(false)){ String msg = uploadMessageMapper.getNormalizeMessage(result, tvr);
this.getMessageMap().addMessage(IConnectorConstants.MESSAGE_ERROR_LITERAL, "rating", null, msg); }
```

### Attachments

Attachments returned from the rating call can be transferred to upload response using the following:

```
File cacheFile = new File(filepath); if(cacheFile.exists()) { ByteArrayOutputStream outstream= new ByteArrayOutputStream(); long size =
Streams.copy(new FileInputStream(cacheFile), outstream, true); SecureFileAttachmentInfo attachment = new SecureFileAttachmentInfo(null,
cacheFile.getPath(), fileName, size, "application/pdf"); dataBundle.addAttachment(attachment); }
```

The following is a full rating connector sample:

```
/** Sample RealtimeRating Connector */ package com.agencyport.intkit.core.upload.realtimerating; import java.io.ByteArrayOutputStream;
import java.io.File; import java.io.InputStream; import java.io.IOException; import java.util.List; import java.util.Random; import
org.apache.commons.fileupload.util.Streams; import com.agencyport.connector.ConnectorDataBundle; import
com.agencyport.connector.IConnectorConstants; import com.agencyport.connector.Outcome; import
com.agencyport.connector.impl.ConnectorManager; import com.agencyport.data.DataManager; import
com.agencyport.data.corrections.TransactionCorrectionReport; import com.agencyport.domXML.APDataCollection; import
com.agencyport.domXML.Aggregate; import com.agencyport.domXML.Index; import com.agencyport.domXML.widgets.PolicyType; import
com.agencyport.logging.ExceptionLogger; import com.agencyport.security.http.SecureFileAttachmentInfo; import
com.agencyport.shared.APException; import com.agencyport.trandef.Transaction; import
com.agencyport.trandef.TransactionDefinitionManager; import com.agencyport.trandef.validation.TransactionBasedValidationResultFilter;
import com.agencyport.trandef.validation.TransactionValidationReport; import com.agencyport.trandef.validation.ValidationResult; import
com.agencyport.upload.UploadMessageMapper; import com.agencyport.utils.AppProperties; import
com.agencyport.webshared.IWebsharedConstants;/** The RealTimeRatingSimulator class provides a reasonable example on how to call
prepare * for a rating call and then call rating. */ public class RealTimeRatingSimulator extends ConnectorManager { /** The
<code>CURRENT_TERM_AMT</code> a constant for the XML element where the premium is mapped. */ private static final String
CURRENT_TERM_AMT = ".CurrentTermAmt.Amt"; /** This applies a filter to the TVR so that only the errors associated with the quick
quote * transaction are included on methods like {@link TransactionValidationReport#isClean()} etc. */ @param tvr is the transaction validation
report currently in context. */ protected void establishValidationResultFilter(TransactionValidationReport tvr){ Transaction fullappTransaction =
tvr.getSourceDataContainer().getTransaction(); if
(!IWebsharedConstants.QUOTE_TRANSACTION_TYPE.equals(fullappTransaction.getType())){ Transaction qqTransaction =
TransactionDefinitionManager.getTransaction(fullappTransaction.getLobCode()),
IWebsharedConstants.QUOTE_TRANSACTION_TYPE); if (qqTransaction != null){ tvr.setValidationResultFilter(new
TransactionBasedValidationResultFilter(qqTransaction, fullappTransaction)); } } } /** {@inheritDoc} */
@Override public Outcome
```

```

execute(ConnectorDataBundle dataBundle) throws APEException { UploadMessageMapper uploadMessageMapper =
UploadMessageMapper.create(null, this); TransactionValidationReport tvr = runTransactionValidation(dataBundle);
TransactionCorrectionReport tcr = runDataCorrections(dataBundle, tvr); // Get the very last TVR associated with having run the corrections. tvr
= tcr.getTransactionValidationReport(); // Add a filter onto the TVR that will allow us only to look at the errors related to the quick quote
transaction establishValidationResultFilter(tvr); if (!tvr.isClean()){ for (ValidationResult result : tvr.getEntireDetailValidationResultSet(false)){
String msg = uploadMessageMapper.getNormalizeMessage(result, tvr);
this.getMessageMap().addMessage(IConnectorConstants.MESSAGE_ERROR_LITERAL, "rating", null, msg); } return null; } // Restore the
filter back tvr.setValidationResultFilter(null); DataManager dataManager = dataBundle.getDataManager(); APDataCollection apData =
dataManager.getAPDataCollection(); PolicyType typeFinder = new PolicyType(apData); String path =
typeFinder.getPolicyAggregateElementPath(); apData.setFieldValue(path + CURRENT_TERM_AMT,"9999"); populateRates(apData);
this.getMessageMap().addMessage(IConnectorConstants.MESSAGE_INFO_LITERAL, "rating", null, "Test INFO message");
this.getMessageMap().addMessage(IConnectorConstants.MESSAGE_WARNING_LITERAL, "rating", null, "Test WARNING message");
this.getMessageMap().addMessage(IConnectorConstants.MESSAGE_ERROR_LITERAL, "rating", null, "This is a hard stop message"); try {
File cacheFile = new File(AppProperties.getContextPath() + "/WEB-INF/shared/pdf/forms/",
"AP_WORKITEM_ASSISTANT_COMMENTS.pdf"); if(cacheFile.exists()) { ByteArrayOutputStream outstream = new
ByteArrayOutputStream(); long size = Streams.copy(new FileInputStream(cacheFile), outstream, true); SecureFileAttachmentInfo attachment =
new SecureFileAttachmentInfo(null, cacheFile.getPath(), "AP_WORKITEM_ASSISTANT_COMMENTS.pdf", size, "application/pdf");
dataBundle.addAttachment(attachment); } } catch (IOException ex) { ExceptionLogger.log(ex, this.getClass(), "execute"); } return
Outcome.CONDITIONS_MET; } /** * {@inheritDoc} */
@Override public boolean postProcess(ConnectorDataBundle dataBundle) { return
true; } /** * Place holder for calling the rating engine. * @param apData is the data collection in context. */
protected void
populateRates(APDataCollection apData){ try { List<Aggregate> coverages = apData.selectNodes("//Coverage");
coverages.addAll(apData.selectNodes("//Comm1Coverage")); for (Aggregate cov : coverages) { Random r = new Random(); int rate =
r.nextInt(1000 - 50) + 50; Index index = cov.computeIndex(apData); apData.setFieldValue(index.getElementPath() + CURRENT_TERM_AMT,
index.getIdArray(), rate + ""); } } catch (Exception ex) { ex.printStackTrace(); } } }

```

# Logging Configuration

AgencyPortal's logging configuration feature provides the following functionality:

- Application level logging, which can be configured for multiple applications on the same JVM. This separates application logging from JVM logging, allows application level control over logging and prevents co-mingling logging output of multiple applications on the same JVM.
- Configurable logging output format, which allows for application specific log output formats.
- Ability to vary logging levels by package.
- Ability to specify special logging destinations, such as syslog.
- Supports 'hot updates' to existing logging configuration, which allows for on the fly configuration changes to the application logging configuration file.

This functionality specifically helps large customers who install multiple applications into the same JVM sandbox in WebSphere or WebLogic with their logging infrastructure.

The `com.agencyport.logging` package contains the classes used by logging.framework. The following is a brief outline of the major classes:

- **LoggingManager** - This class is responsible for initializing from application configuration, monitoring hot updates to logging configuration files and providing the facade for logger acquisition for all product and application level code.
- **AbstractFormatter** - The base abstract log formatter class for all concrete Agencyport log formatters. All application custom log formatters should inherit from this base class as a best practice. It provides access to many of the facets that compose your typical log line.
- **DefaultLogFormatter** - Agencyport's concrete default log formatter. This provides a robust log line, leveraging as many facets possible that are available on both the incoming `LogRecord` and `LoggingContext` instances from the current thread.
- **PortalFileHandler** - The recommended canonical file handler for all log file targets. It reads the configuration from application logging configuration or JVM logging configuration, giving precedence to application scoped configuration of JVM logging configuration; thereby, supporting log target separation by application with nice backward compatibility.
- **ConsoleHandler** - The framework's alternative to the `java.util.logging.ConsoleHandler` class. While both the JDK `ConsoleHandler` and SDK `ConsoleHandler` achieve the goal of redirecting `LogRecord` objects to the Console's `OutputStream`, the JDK `ConsoleHandler` redirects everything to `Systemerr` while our `ConsoleHandler` only sends `LogRecord` objects marked `Level.SEVERE` to `Systemerr`; everything else goes to `Systemout`.

The following is a list of general guidelines to follow when setting up this feature:

- The `defer_logging_initialization` application property turns off all of AgencyPortal logging initialization and hot update monitoring when it is set to true. This should only be done if the web container takes over all of the logging responsibilities, including setting of logging levels and establishing logging handlers and formatters.
- Application teams must ensure the uniqueness of the following application properties:
  - `APPLICATION_NAME` - application ID facet of log line.
  - `application.logging_name_prefix` - keeps logger hierarchies separated when sharing JVM with other Agencyport applications.
- Acquisition of all `java.util.logging.Logger` instances from Java code, regardless of whether it is product or application based, MUST adhere to the `com.agencyport.logging.LoggingManager.getLogger()` method.
- If you want to monitor log activity, refer to the default log format (`DefaultLogFormatter`) that ships with the product logs. `DefaultLogFormatter` is Agencyport's concrete default log formatter and provides a robust log line that leverages as many facets as possible, which are available on both the incoming `LogRecord` and `LoggingContext` instances from the current thread.

Logging Facet	Source	Base Derivation
Server host name	<code>AbstractFormatter.getHostName()</code>	<code>InetAddress.getLocalHost().getHostName()</code>
Server port	<code>AbstractFormatter.getServerPort()</code>	<code>HttpServletRequest.getServerPort();</code> access to request via logging context
Application name	<code>AbstractFormatter.getApplicationName()</code>	<code>Application.getStringProperty("APPLICATION_NAME")</code>
Timestamp	format controlled via <code>AbstractFormatter.getTimestampFormat()</code>	<code>LogRecord.currentTimeMillis()</code>

Logging Facet	Source	Base Derivation
Session ID	AbstractFormatter.getSessionId()	HttpSession.getId(); access to session via logging context
Transaction ID	AbstractFormatter.getFormattedTransactionId()	LoggingContext.getTransactionId():LogRecord.getSequenceNumber(); access to transaction id via logging context
Client IP address	AbstractFormatter.getClientIPAddress()	HttpServletRequest.getRemoteAddr(); access to request via logging context
User credentials	AbstractFormatter.getUserCredentials()	LoggingContext.getUserCredentials() – most always just the login ID that is available from logging context. When an audit record is available it is enhanced with the audit link actor short name.
Thread name	Thread.currentThread().getName()	
Java class name		LogRecord.getSourceClassName()
Java method		LogRecord.getSourceMethodName()
Message		LogRecord.getMessage()
Exception		LogRecord.getThrown().getStackTrace()
Extra parameters		LogRecord.getParameters(). Each entry's toString() method is added to the log line.

### Example

The following is a sample log line:

```
dell820-03 8080 AgencyPortal_LOB_Demo INFO 2010-04-05 08:17:56.00283 EDT 411DB63F29C1ED18F1C0DAA18D304E4A
T1395:1013 127.0.0.1 agent http-8080-4 com.agencyport.xarc.evaluate.WhereClause evaluate EVALUATING XPATH=upper-case(
/PersAutoPolicyQuoteInqRq/InsuredOrPrincipal[InsuredOrPrincipalInfo/InsuredOrPrincipalRoleCd='Insured']/GeneralPartyInfo/Addr[
AddrTypeCd='MailingAddress']/City) = upper-case('quincy')
```

Where...

dell820-03 8080 = server host name and port it is listening on

AgencyPortal\_LOB\_Demo = application name

INFO = logging level

2010-04-05 08:17:56.00283 EDT = time stamp

411DB63F29C1ED18F1C0DAA18D304E4A = browser session id

1395 = work item id

1013 = logging sequence number

127.0.0.1 – client ip address

http-8080-4 – thread name

com.agencyport.xarc.evaluate.WhereClause evaluate = source method name from where logging content emitted from

```
EVALUATING XPATH=upper-case()
/PersAutoPolicyQuoteInqRq/InsuredOrPrincipal[InsuredOrPrincipalInfo/InsuredOrPrincipalRoleCd='Insured']/GeneralPartyInfo/Addr[AddrTypeCd='MailingAddress']/City) = upper-case('quincy') = application logging message
```

## Logging Destination & Configuration Properties

The application properties file (framework.properties) is the file that controls the logging subsystem. This file is not monitored for hot updates; changes here require a server restart.

This file includes the following logging configuration properties:

Property	Description	Default Value	Recommended Setting (if any)
defer_logging_initialization	Global flag used to turn off all AgencyPortal logging initialization	false	
application.logging_config_file	A list of the SDK's application logging configuration files. These files are loaded to make up the actual logging configuration for the application. Each entry is delimited with a comma. Refer to the <a href="#">Application Logging Configuration File</a> section for more information.	\${custom_logging_config_file}	
logging_config_file_monitor	Enables/disables live monitoring of the application.logging_config_file to support "hot updates" to the file at runtime.	true	Should be false in a production environment.
application.logger_name_prefix	Unique logger name prefix. Keeps logger hierarchies separated when sharing JVM with other Agencyport applications.	none	

## Application Logging Configuration

The application logging configuration file contains the logging properties specific to a given application. This file is where the real logging configuration takes place. It is based on Java Utility Logging, although processed in a proprietary manner. The application logging configuration file is monitored for hot updates and any changes to the file are consumed by the `LoggingManager` and applied within seconds of it being changed.

### Important

It is important to note that any logging names configured in this file must NOT contain the logger name prefix from application properties.

The following is a correctly configured application logging configuration file:

```
#####
AgencyPort custom logging configuration parameters.
#####
Configuration for logging handler list under the
com.agencyport.portal.handlers property name #####
com.agencyport.portal.handlers=com.agencyport.logging.PortalFileHandler,com.agencyport.logging.syslog.SyslogHandler,com.agencyport.logging.ConsoleHandler #####
Configuration detailing which loggers to install the custom handlers to. All # loggers in this list will have the handlers above installed to. Each entry is delimited with a comma. # If this property is missing then the logger list defaults to 'com.agencyport' # If you have a carrier specific package then you should configure # this to be
com.agencyport.portal.loggers=com.agencyport.com.carrier #####
com.agencyport.portal.loggers=com.agencyport.logging.DefaultLogFormatter #####
DefaultLogFormatter supports a way to alter the time stamp format
com.agencyport.logging.DefaultLogFormatter.timestamp.format=yyyy-MM-dd HH:mm:ss.SSSSS z #####
Configuration for AgencyPortal's own file handler
com.agencyport.logging.PortalFileHandler.pattern=${output_log_dir}templates%g.log
com.agencyport.logging.PortalFileHandler.level=FINEST com.agencyport.logging.PortalFileHandler.limit=20000000
com.agencyport.logging.PortalFileHandler.count=3
com.agencyport.logging.PortalFileHandler.formatter=com.agencyport.logging.DefaultLogFormatter
```

```

com.agencyport.logging.PortalFileHandler.append=true #####
Configuration for portal syslog handler #####
com.agencyport.logging.syslog.SyslogHandler.formatter=com.agencyport.logging.syslog.SyslogFormatter
com.agencyport.logging.syslog.SyslogHandler.level=INFO com.agencyport.logging.syslog.SyslogHandler.host=127.0.0.1
com.agencyport.logging.syslog.SyslogHandler.port=514 #####
Configuration for console handler #####
com.agencyport.logging.ConsoleHandler.level=ALL
com.agencyport.logging.ConsoleHandler.formatter=com.agencyport.logging.DefaultLogFormatter
#####
Logger specific properties. # Provides extra control for each logger.
#####
com.agencyport.useParentHandlers=false
com.agencyport.level=INFO com.agencyport.service.config.level=CONFIG com.agencyport.logging.level=CONFIG # Here is the way to set
logging levels on 3rd party library loggers. Note the use of the starting sequence: <no-prefix>. # This tells LoggingManager not to apply a prefix
to this logger. <no-prefix>.org.apache.solr.level=INFO com.agencyport.cache.serialization.level=INFO
com.agencyport.security.csrf.impl.level=INFO com.agencyport.webshared.level=INFO com.agencyport.security.profile.impl.level=INFO
com.agencyport.secure.front.level=INFO com.agencyport.database.locking.level=INFO com.agencyport.security.filter.level=INFO

```

## Example

The following is a sample log line:

```
dell820-03 8080 AgencyPortal_LOB_Demo INFO 2010-04-05 08:17:56:00283 EDT 411DB63F29C1ED18F1C0DAA18D304E4A T1395:1013
127.0.0.1 agent http-8080-4 com.agencyport.xarc.evaluate.WhereClause evaluate EVALUATING XPATH=upper-case(
/PersAutoPolicyQuoteInqRq/InsuredOrPrincipal[InsuredOrPrincipalInfo/InsuredOrPrincipalRole Cd='Insured']/GeneralPartyInfo/Addr[AddrTyp
eCd='MailingAddress']/City) = upper-case('quincy')</blockquote>
```

The following is a list of the properties included in the logging.properties file:

Property	Description	Default Value	Recommended Setting (if any)
com.agencyport.portal.handlers	Comma separated list of logging handlers to allocate for the application. The built-in handlers provided by the SDK are: <ul style="list-style-type: none"> <li>• <b>com.agencyport.logging.PortalFileHandler</b> - Extension of the java.util.logging.FileHandler, which facilitates an application specific destination logging file distinct from the one associated with the JDK FileHandler class name.</li> <li>• <b>com.agencyport.logging.syslog.SyslogHandler</b> - Handles log redirection via UDP to a syslog server.</li> <li>• <b>com.agencyport.logging.ConsoleHandler</b> - Handles redirection to console application (i.e., System.out and System.err).</li> </ul>	null	Most applications would have used both the PortalFileHandler and ConsoleHandler
com.agencyport.portal.loggers	Comma separated list of root loggers to allocate for the application. All loggers in this list will have the handlers above installed too.	com.agencyport	If you have a carrier specific package, then you should configure this to be com.agencyport.portal.loggers=com.agencyport.com.carrier
com.agencyport.logging.DefaultLogFormatter.timestamp.format	Format string that controls the time stamp format portion of the log	yyyyMMddHH:mm:ss:SSSS	

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
	line shown when the DefaultLogFormatter is in use.		

#### Portal File Handlers

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
com.agencyport.logging.PortalFileHandler.pattern	Specifies the log file target path name for the framework's default file handler.	null	
com.agencyport.logging.PortalFileHandler.limit	Specifies the maximum size in bytes the log file can grow to before rotation begins.	100000	
com.agencyport.logging.PortalFileHandler.count	Specifies the number of generations in the file rotation strategy.	3	
com.agencyport.logging.PortalFileHandler.formatter	Specifies the formatter that the default file handler should use.	null	
com.agencyport.logging.PortalFileHandler.append	Specifies whether to append or truncate the target log file.	true	
com.agencyport.logging.PortalFileHandler.level	Specifies the logging level for this handler.	Level.All	

#### System Loggers

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
com.agencyport.logging.syslog.SyslogHandler.formatter	Formatter used by the syslog handler.	null	
com.agencyport.logging.SyslogHandler.level	Specifies the logging level for this handler.	Level.All	
com.agencyport.logging.SyslogHandler.host	Host name/IP Address of syslog server.	127.0.0.1	
com.agencyport.logging.SyslogHandler.port	Port number of syslog server.	514	

#### Logger Properties

The following properties can be configured for individual Java packages. For example, setting com.agencyport.security.level=FINE would enable fine logging for all classes under the com.agencyport.security package.

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Recommended Setting (if any)</b>
com.agencyport.useParentHandlers	Boolean flag determines whether the application root loggers should chain to the handlers at the JVM level.	true	
<package_name>.level	Package	Level.All	

To configure logging levels for third party code, which lives outside of the application namespace (as defined by the com.agencyport.portal.loggers property described above), then you can use the following special syntax:

```
<no-prefix>.org.apache.solr.level=INFO
```

# Logging Packages and Classes

The `com.agencyport.logging` and `com.agencyport.logging.syslog` packages contain the classes used by the logging configuration feature. The following is a brief outline of the major classes.

## `com.agencyport.logging`

- **LoggingManager** - This class is responsible for initializing from application configuration, monitoring hot updates to logging configuration files and providing the facade for logger acquisition for all product and application level code.
- **AbstractFormatter** - The base abstract log formatter class for all concrete Agencyport log formatters. All application custom log formatters should inherit from this base class as a best practice. It provides access to many of the facets that compose your typical log line.
- **DefaultLogFormatter** - Agencyport's concrete default log formatter. This provides a robust log line, leveraging as many facets possible that are available on both the incoming `LogRecord` and `LoggingContext` instances from the current thread.
- **PortalFileHandler** - The recommended canonical file handler for all log file targets. It reads the configuration from application logging configuration or JVM logging configuration, giving precedence to application scoped configuration of JVM logging configuration; thereby, supporting log target separation by application with nice backward compatibility.
- **ConsoleHandler** - The framework's alternative to the `java.util.logging.ConsoleHandler` class. While both the JDK `ConsoleHandler` and SDK `ConsoleHandler` achieve the goal of redirecting `LogRecord` objects to the Console's `OutputStream`, the JDK `ConsoleHandler` redirects everything to `System.err` while our `ConsoleHandler` only sends `LogRecord` objects marked `Level.SEVERE` to `System.err`; everything else goes to `System.out`.

# Configuration Files

There are two configuration files that are used to control logging. A table of properties for each is provided below.

## Application Properties File

The application properties file is the file that controls the logging subsystem. This file is not monitored for hot updates; changes here require a server restart.

```
APPLICATION_NAME=AgencyPortal_LOB_Demo ##### Logging configuration ##### defer_logging_initialization=false ##### application.logging_config_file contains logging configuration which # is at application scope. ##### application.logging_config_file=${my_context_path}WEB-INF/application.logging.properties ##### application.logger_name_prefix contains the unique logger name # prefix for the application so that logging targets can stay unique # by applications hosted by the same JVM ##### application.logger_name_prefix=${APPLICATION_NAME}
```

The following is the application properties file property list:

Property	Description	Default Value	Comment
defer_logging_initialization	Global flag used to turn off all AgencyPortal logging initialization	false	Useful for pre 4.1 applications, which only supported JVM level logging initialization.
application.logging_config_file	Application logging configuration file list. Each entry is delimited with a comma.	null	Required for 4.1+ applications to support logger segregation by application.
application.logger_name_prefix	Unique logger name prefix.	null	Required for 4.1+ applications to support logger segregation by application. Must be unique by application within JVM. Must be coupled with a coding change in application to acquire logger instances via the <code>LoggingManager.getLogger()</code> .

## Application Logging Configuration File

The application logging configuration file contains the logging properties specific to a given application. This file is where the real logging configuration takes place. It is based on Java Utility Logging, although processed in a proprietary manner. The application logging configuration file is monitored for hot updates and any changes to the file are consumed by the `LoggingManager` and applied within seconds of it being changed.

### Important

It is important to note that any logging names configured in this file must NOT contain the logger name prefix from application properties.

The following is a correctly configured application logging configuration file:

```
AgencyPort custom logging configuration parameters.##### Configuration for logging handler list under the com.agencyport.portal.handlers property name ##### com.agencyport.portal.handlers=com.agencyport.logging.PortalFileHandler,com.agencyport.logging.syslog.SyslogHandler,java.util.logging.ConsoleHandler ##### Configuration detailing which loggers to install the custom handlers to. All # loggers in this list will have the handlers above installed to. Each entry is delimited with a comma. # If this property is missing then the logger list defaults to 'com.agencyport' ##### com.agencyport.portal.loggers=##### Configuration for AgencyPortal's own file handler ##### com.agencyport.logging.PortalFileHandler.pattern=${output_log_dir}templates%g.log
```

```

com.agencyport.logging.PortalFileHandler.level=FINEST com.agencyport.logging.PortalFileHandler.limit=20000000
com.agencyport.logging.PortalFileHandler.count=3
com.agencyport.logging.PortalFileHandler.formatter=com.agencyport.logging.DefaultLogFormatter
com.agencyport.logging.PortalFileHandler.append=true #####
DefaultLogFormatter supports a way to alter the time stamp format #####
com.agencyport.logging.DefaultLogFormatter.timestamp.format=yyyy-MM-dd HH:mm:ss:SSSSS z
Configuration for portal syslog handler
com.agencyport.logging.SyslogHandler.formatter=com.agencyport.logging.syslog.SyslogFormatter
com.agencyport.logging.SyslogHandler.level=INFO com.agencyport.logging.SyslogHandler.host=127.0.0.1
com.agencyport.logging.SyslogHandler.port=514 #####
Configuration for console handler #####
java.util.logging.ConsoleHandler.level=ALL #java.util.logging.ConsoleHandler.formatter=com.agencyport.logging.DefaultLogFormatter
Logger specific properties. # Provides extra control for each logger.
com.agencyport.useParentHandlers=false
com.agencyport.level=INFO com.agencyport.service.config.level=CONFIG com.agencyport.logging.level=CONFIG
org.apache.solr.level=INFO com.agencyport.cache.serialization.level=CONFIG com.agencyport.security.csrf.impl.level=FINEST
#com.agencyport.webshared.level=INFO #com.agencyport.security.profile.impl.level=INFO #com.agencyport.secure.front.level=FINER
#com.agencyport.database.locking.level=FINE #com.agencyport.preconditions.level=FINE

```

## Example

The following is a sample log line:

```
dell820-03 8080 AgencyPortal_LOB_Demo INFO 2010-04-05 08:17:56.00283 EDT 411DB63F29C1ED18F1C0DAA18D304E4A T1395:1013
127.0.0.1 agent http-8080-4 comagencyport.xarc.evaluate.WhereClause evaluate EVALUATING XPATH=upper-case(
/PersAutoPolicyQuoteInqRq/InsuredOrPrincipal[InsuredOrPrincipalInfo/InsuredOrPrincipalRole Cd='Insured']/GeneralPartyInfo/Addr[AddrTyp
eCd='MailingAddress ']/City) = upper-case('quincy')</blockquote>
```

The following is the application logging configuration properties file property list:

Property	Description	Default Value	Comment
com.agencyport.portal.handlers	Comma separated list of logging handlers to allocate for the application.	null	Required
com.agencyport.portal.loggers	Comma separated list of root loggers to allocate for the application.	com.agencyport	Optional
com.agencyport.logging.PortalFileHandler.pattern	Specifies the log file target path name for the framework's default file handler.	null	Required
com.agencyport.logging.PortalFileHandler.limit	Specifies the maximum size in bytes the log file can grow to before rotation begins.	100000	Optional
com.agencyport.logging.PortalFileHandler.count	Specifies the number of generations in the file rotation strategy.	3	Optional
com.agencyport.logging.PortalFileHandler.formatter	Specifies the formatter that the	null	Required

<b>Property</b>	<b>Description</b>	<b>Default Value</b>	<b>Comment</b>
	default file handler should use.		
com.agencyport.logging.PortalFileHandler.append	Specifies whether to append or truncate the target log file.	true	Optional
com.agencyport.logging.PortalFileHandler.level	Specifies the logging level for this handler.	Level.All	Optional
com.agencyport.logging.DefaultLogFormatter.timestamp.format	Format string that controls the time stamp format portion of the log line.	yyyyMMddHH:mm:ss:SSSSS	Optional
com.agencyport.logging.syslog.SyslogHandler.formatter	Formatter used by the syslog handler.	null	Required
com.agencyport.loggin.SyslogHandler.level	Specifies the logging level for this handler.	Level.All	Optional
com.agencyport.loggin.SyslogHandler.host	Host name/IP Address of syslog server.	127.0.0.1	Optional
com.agencyport.logging.SyslogHandler.port	Port number of syslog server.	514	Optional
com.agencyport.useParentHandlers	Boolean flag determines whether the application root loggers should chain to the handlers at the JVM level.	true	Optional
com.agencyport.xxxxxxx.level	Facility logger levels by package.	Level.All	Remember not to prefix with the unique logger name prefix.

# Debug Console

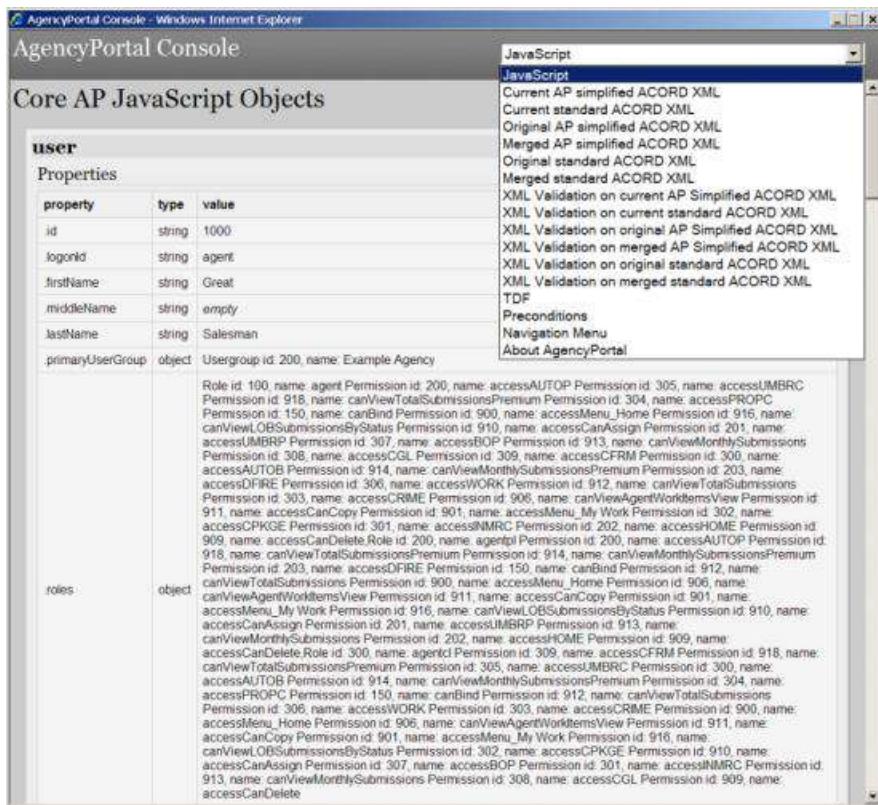
The AgencyPortal debug console is designed as a development aid and provides useful information about the application. The debug console is made up of panels, each of which provides a view into different information. Panels provided by the framework include the following:

- **JavaScript:** The default panel. Provides information about the framework JavaScript variables available on a given page.
- **Current AP simplified ACORD XML:** Provides a view of the underlying XML in its simplified form. This simplified form is the version of the data that is persisted to the disk.
- **Current standard ACORD XML:** Provides a view of the underlying XML in its standard form. This transient document is created on demand by running the simplified document through the simplified to standard transformations. This is the version of the data that should be used to communicate with external systems that aren't aware of the simplifications made to schema.
- **Original AP simplified ACORD XML:** Provides a view of the original uploaded XML document in its simplified form. For new business transactions, this applies to uploaded work items only. During an upload, the document received from the source system is transformed to its simplified form before it is persisted to the disk.
- **Merged AP simplified ACORD XML:** Provides a view of the current simplified document as compared with the original simplified document for change tracking purposes. This transient document is created on demand by MergeManager processing (only applies to transactions that have an original document, such as new business uploads, endorsements and renewals).
- **Original Standard ACORD XML:** Provides a view of the original uploaded XML document in its standard form. For new business transactions, this applies to uploaded work items only. During an upload, the document received from the source system is transformed to its simplified form before it is persisted to the disk.
- **Merged Standard ACORD XML:** Provides a view of the current standard document as compared with the original standard document for change tracking purposes. This transient document is created on demand by MergeManager processing (only applies to transactions that have an original document, such as new business uploads, endorsements and renewals).
- **XML Validation Selections:** Provides schema validation of the various documents and notes any errors.
- **TDF:** Provides a view of the current in-memory TDF after DTR is applied.
- **Preconditions:** A snapshot of application preconditions that drive DTR as they exist at the moment.
- **Navigation Menu:** A view of the underlying XML menu file for the work item.
- **Application Properties:** A view of the current runtime application properties.
- **About AgencyPortal:** A view of version information for framework and other JAR files.

The debug console is disabled by default. You can engage it through the following application property setting:

```
client_debug=true
```

The following is a screen shot sample of the debug console:



## Custom Debug Panels

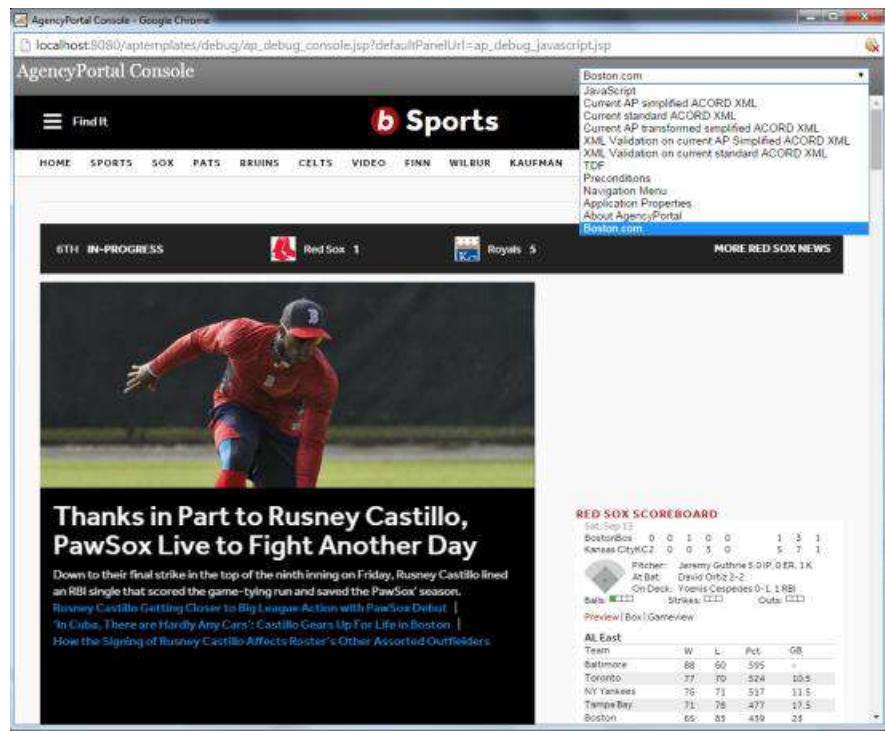
The debug console is customizable. Application development teams can easily:

- define the default panel
- add additional panels, as required

You can accomplish each of these customizations by modifying the `custom_debug.js` file found in the `debug` directory. In the following example, `boston.com` is added as a panel and set as the default:

```
/* add any additional custom debug panels necessary for your application in this file */ debugConsole.addPanel('boston','http://www.boston.com','Go to Boston.com', 'Boston.com'); debugConsole.setDefaultPanel('boston');
```

This is how it might look:



# Application Bootstrap

There are numerous framework resources, such as product definitions and application properties, that need to be bootstrapped during application server or servlet container initialization. For the most part, the JEE ServletContextListener API is used to listen for and handle the application startup and shutdown events (with the main exception being that the Solr engine is initialized during ServletFilter initialization).

The framework includes a `ServletContextListener` implementation class named `InitializerServletContextListener`, which handles the majority of the workload. When the "contextInitialized" event is invoked, the following steps occur in the order specified:

1. Initialization of the ApplicationProperties subsystem
2. Initialization of the Logging subsystem
3. Verification of runtime JAR versions
4. Invocation of the BootServiceDriver, which in turn boots all IBootServerIntf implementations that have been specified by the `boot_service_providers` (token based) property in application properties (refer to the [Bootservice Providers](#) section below for more information).

The `InitializerServletContextListener` class contains a "systemBootSuccess" flag (accessible via a call to `InitializerServletContextListener.getSystemBootSuccessFlag()`), which will be set to false if any of the above steps fail. Some errors may occur before the Logging system has been initialized. As such, the application logs to the `ConsoleAppLogger` (i.e., `std out` and `std err`) before the Logging subsystem is ready to be used.

`ServletContextListeners` are registered to the application server or servlet container in `web.xml` using the `<listener>` element, which is a direct child of the root `<web-app>` element.

```
<listener> <listener-class>com.agencyport.servlets.shared.InitializerServletContextListener</listener-class> </listener>
```

## Note

You can have more than one `<listener>` element in `web.xml`; they will be invoked in the order specified.

The `InitializerServletContextListener` depends upon two "context-param" values that are also specified within `web.xml`:

The "PROPERTY\_FILE\_NAME" context parameter specifies the location and filename (minus the extension) of the main application properties file.

## Example

The following indicates to the framework that it should use this path when loading the main properties file:

```
<context-param> <param-name>PROPERTY_FILE_NAME</param-name> <param-value>/WEB-INF/framework</param-value> </context-param>
```

Path to use: `WEB-INF/framework.properties`.

The "ENV\_ID" context parameter specifies the name of the system property used to identify the current environment by some textual name. The current environment name is used by the `EnvironmentResourceProvider` (refer to the [Environment Specific Resources](#) section for more information).

## Example

Let's say you pass "agencyenv" as shown in the following code snippet:

```
<context-param> <param-name>ENV_ID</param-name> <param-value>agencyport.env</param-value> </context-param>
```

The application will, effectively, do a call to `System.getProperty("agency.env")` to determine the current environment name. So, if you added a JVM argument of "`-Dagencyport.env=perf02`", then the name of the current environment is deemed to be "perf02".

Now, let's say, for example, that the infrastructure operations team didn't want to add the "`-Dagencyport.env=perf02`" JVM argument because they already had a different naming convention for environment names (e.g., "`Dcurrent.server.name=perf02`"). In this case, you can change the parameter value in `web.xml` to match their naming convention like so:

```
<context-param> <param-name>ENV_ID</param-name> <param-value>current.server.name</param-value> </context-param>
```

If you omit this parameter, the default `agencyport.env` systemproperty name is used.

## Note

All of the above information pertains to the framework's standard out of the box `InitializerServletContextListener` implementation. It is important to note, however, that this class is designed to be extended so that application teams can customize various aspects of the application Bootstrap. One example of this is that the (optional) integration kit framework ships with its own extension of this class named `IntegrationInitializerServletContextListener` so that it can load integration configuration files during Bootstrap. The `web.xml` file that ships with the SDK uses the `IntegrationInitializerServletContextListener` by default, but you can change it back to the standard version if integration kits are not going to be used. The following is what the out of the box `web.xml` file looks like:

```
<!-- Context "listeners" and their associated context parameters are where the bootstrap magic happens --> <context-param> <param-name>PROPERTY_Y_FILE_NAME</param-name> <param-value>/WEB-INF/framework</param-value> </context-param> <context-param> <param-name>ENV_ID</param-name> <param-value>agencyport.env</param-value> </context-param> <listener> <description>Boots traps the integration kit framework and the agencyportal framework</description> <listener-class>com.agencyport.intkit.core.servlets.IntegrationInitializerServletContextListener</listener-class> <!-- if you don't have integration kit, use this listener-class instead: <listener-class>com.agencyport.servlets.shared.InitializerServletContextListener</listener-class> --> </listener>
```

## Bootservice Providers

"Bootserice provider" is the name given to a Java class that implements the `IBootServiceIntf` interface. The `BootServiceDriver`, which is invoked during application bootstrap, initializes all bootserice provider implementations that are made known to the application via application properties. The purpose of the bootserice provider is to:

- provide a mechanism to allow "services" or subsystems to be bootstrapped during the application server initialization process.
- provide an interface definition that these services or subsystems should implement so that their initialization is automatically triggered during server initialization.

## Example

Examples of services or subsystems (included in the AgencyPortal runtime system) that are bootstrapped during the application server initialization process are the transaction definition subsystem and the XML processing subsystem.

Examples of services a developer would make bootable via the Bootserice subsystem are

- a service providing static in-memory lookup tables, or
- a service that creates persistent connections to a third party service provider

The subsystem is comprised of two elements:

- `BootServiceDriver` - the driver that boots all the services implementing the bootserice interface.
- `IBootServiceIntf` - an interface definition implemented by services wanting to boot at server startup time.

## Configuration

All services implementing the `IBootServiceIntf` subsystem make themselves known to the `BootServiceDriver` subsystem by appending their class name to the `boot_service_providers` (token based) property in your application properties file.

Property	Description	Default Value	Recommended Setting (if any)
<code>boot_service_providers</code>	This lists several built-in service providers that require some activity to take place during application bootup, such as the <code>HTMLTemplateManager</code> and the <code>TransactionDefinitionManager</code> . User supplied classes or subsystems are also listed here.	none	

The `com.agencyport.bootserice.BootServiceDriver` class is responsible for starting all bootable services or subsystems that have registered themselves in the application properties file.

The main method of this class is:

```
public static void bootSubSystems()
```

As mentioned above, the `InitializerServletContextListener` class calls this method at sever startup. This method reads the list of classes found in the properties file and, using reflection, instantiates each class in turn and calls its `init()` method.

This `init()` method is specified in the `com.agencyport.bootservice.IBootServiceIntf` interface. After each service is instantiated, the service reports back success or failure to the `BootServiceDriver` via the return value of the `init()` method.

The `BootServiceDriver` subsystem keeps track of the overall success or failure of all services in an internal Boolean flag. This internal flag is implemented with a "once false always false" algorithm. Other services or code interested in the boot status of the application can query the `BootServiceDriver` subsystem with the following method:

```
public boolean getSystemBootSuccessFlag()
```

The following is the public API for the `BootServiceDriver` subsystem:

```
public class BootServiceDriver { public static void BootServiceDriver.bootSubSystems() public boolean getSystemBootSuccessFlag() }
```

`IBootServiceIntf` is an interface containing the methods `init()`, `getName()` and `flagAsInitialized()`. Any service or subsystem that needs booted at application server startup needs to implement the following interface:

```
public interface IBootServiceIntf { public boolean init() public String getName() public void flagAsInitialized(boolean flag) }
```

- The `init()` method is called by the `BootServiceDriver` subsystem and is the "hook" to whatever code is necessary to initialize the service. This method should return true if the service booted successfully; otherwise, false. The `BootServiceDriver` subsystem records this result so it can report on the overall status of all the services booted at startup.
- The `getName()` method is called by the `BootServiceDriver` subsystem to log the name of the service being booted.
- The `flagAsInitialized(Boolean flag)` is called by the `BootServiceDriver` subsystem after the call to `init()`. The purpose of this call is to set a flag internally to indicate that the server has been initialized. Implementation teams should capture this flag and use it to ignore additional invocations of the service's `init()` method. As a result, this feature is used to enforce a singleton style service implementation.

All classes implementing this interface should contain a static Boolean variable used to record that the service's `init()` method has been called at least once.

### Example

```
private static boolean initialized = false; public void flagAsInitialized(boolean flag) { initialized = flag; }
```

Using the above sample, the service's `init()` method would look something like the following:

```
public boolean init() { // this is a catch for someone trying to initialize this object // outside of the application initialization process. if (initialized) { throw new IllegalStateException("Object already initialized."); } try { //Do initialization work here. } catch(APException apex) { logger.severe("Exception: " + apex.toString()); return false; } return true; }
```

Notice in the above example that the `init` method does not throw any exceptions (except the runtime exception). This means the service initialization code must catch any and all exceptions that could be thrown. If the service catches exceptions that do not allow the service to boot successfully, the service must return a value of false. Internally, the service can and should log any information.

## Environment Specific Resources

To achieve WAR file compatibility between discreet environments (without the need to make adjustments to end point configuration entries), the application bootstrap framework supports the notion of environment specific resources by utilizing the application's `EnvironmentResourceProvider` mechanism.

### Note

Out of the box, the `EnvironmentResourceProvider` is only used to load environment specific versions of application property files, but the mechanism could be used by custom code to load other types of resource.

The `EnvironmentResourceProvider` is used to load the base resource (i.e., the canonical file that isn't specific to an environment), in addition to any versions of the file that are applicable to the current environment. The resolution of files that are applicable only to the current environment is supported by iterating a set of search paths and then a set of prefixes applied to the base resource file name and providing a consolidated input stream or set of input streams from the resources that are present and readable. The order in which it searches for these resources will follow the rules laid out in the following example.

### Example

For each directory in the search paths passed:

1. Load the base resource. For purposes of illustration, assume the name of this base resource is `framework.properties`. Assuming that the caller is using the default list of prefixes, then the following search algorithm will apply.

2. Search and, if found, load the resource named  `${agencyport.env}.framework.properties` (where `agencyport.env` is a system property that is passed to the JVM as a command line argument `-Dagencyport.env=environment value`).
  1. A systemproperty name other than "agencyport.env" can be used by changing the value passed for the ENV\_ID context parameter in the application's web.xml file.
3. Search and, if found, load the resource named  `${hostname}.framework` (where `hostname` is the server name of the hosting machine running this application).
4. Search and, if found, load the resource named  `${user.name}.framework.properties` (where `user.name` is the username that owns the current process).

## Environment Context

The `EnviornmentContext.setEnvironmentResourceProviderPrecedence(String [])` method can be used if there is a need to alter the default resource prefixes and precedence order used by the `EnvironmentResourceProvider`. This method must be invoked early on during the application bootstrap process before the `EnvironmentResourceProvider` has been statically initialized. In typical implementations, this means that if you need to override `InitializerServletContextListener.initializeEnvironmentContext()`,

1. Call super
2. Call `EnvironmentContext.setEnvironmentResourceProviderPrecedence(String [])`,
3. Pass the desired `customPrecedence` array.

The following rules should be considered when providing the `customPrecedence` array.

- The 0th string entry in the `customPrecedence` array is given the highest level of precedence
- Literal string values are evaluated as literal resource name prefixes
- String's specified under the  `${string}` format will be evaluated as systemproperties

The default product precedence array consists of two entries that are all processed as systemproperties:

1.  `"${agencyport.env}"`
2.  `"${application.hostname}"`
3.  `"${user.name}"`

Given the following example of a `customPrecedence` array:

1. `"always"`
2.  `"${cluster.category}"`
3.  `"${agencyport.env}"`
4.  `"${application.hostname}"`
5.  `"${user.name}"`

Assuming the current OS username is "java", the machine's hostname is "prod-vm20", and the SystemProperties "agencyport.env" and "cluster.category" have been established as "prod" and cat1" using JVM startup arguments, then the prefix array that is used by the default `EnvironmentResourceProvider` instance would be:

1. `"always"`
2. `"cat1"`
3. `"prod"`
4. `"prod-vm20"`
5. `"java"`

# Event Audit

Event logging supports the ability answer the following kinds of queries about application usage patterns:

- What part or individual committed a particular action? This is usually the user making the request, but could also be the "system" in cases where auditing records are created, but the current user is either not available or easily accessible.
- What was the nature of the operation or event? Was it an add operation, delete, update, refer, etc?
- What was the target(s) of the operation that was taken?
- When was the operation committed?

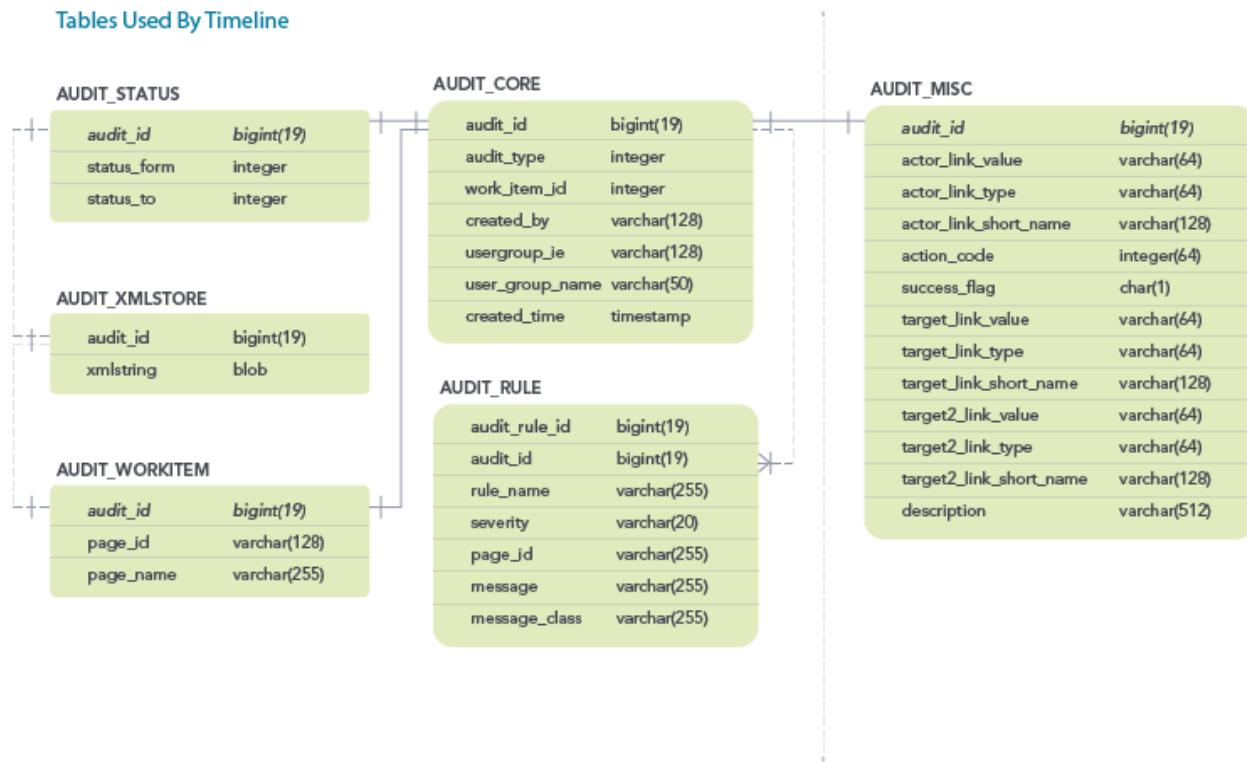
The following are the sample reports that can be generated with specific queries:

- Over the last week, how many work items were bound?
- Over the last week, how many work items were referred?
- Over the last week, how many work items were created?
- How many quotes has a particular user initiated over the last month?
- Who updated this book of business over the last two weeks?
- When was this user added to the system?
- When was this user added to participate in the role of underwriter?
- When was a particular permission granted to a role?
- When was a particular permission revoked from a role?

# Data Model

The following diagram details how to model all events and allow flexible control and easier data retrieval. The auditing data is stored in AUDIT\_CORE and the AUDIT\_RULE, AUDIT\_MISC, AUDIT\_STATUS, AUDIT\_WORKITEM and AUDIT\_XMLSTORE domain specific tables. The AUDITLOG table logs all information from all domains in the system, including user access, toolkit operation and user's administration.

## Tables Used By Timeline



For details of the physical model of the database tables, refer to the Data Model spreadsheet for AgencyPortal 5.2.

The following are the sample queries for specific activities:

Activities	Sample Queries
Status Change	SELECT old_status, ac.status, ac.work_item_id, create_time, created_by, user_group_name FROM audit_status ast, audit_core ac, work_item w WHERE ac.work_item_id = w.work_item_id AND w.status NOT IN (55) AND ac.audit_id = ast.audit_id AND create_time > ? AND create_time < ? AND user_group_id IN (200) ORDER BY create_time DESC
Data Modification	SELECT ac.work_item_id, create_time, created_by, user_group_name, description FROM audit_core ac, audit_workitem aw, work_item w WHERE ac.work_item_id = w.work_item_id AND w.status NOT IN (55) AND ac.audit_id = aw.audit_id AND ac.audit_type = 'DATA_MODIFIED' AND create_time > ? AND create_time < ? AND user_group_id IN (200) ORDER BY create_time DESC
Rule Events	SELECT audit_rule_id, rule_name, severity, page_id, message, message_class, ac.audit_id, ac.work_item_id, create_time, created_by, user_group_name FROM audit_rule ar, audit_core ac, work_item w WHERE ac.work_item_id = w.work_item_id AND w.status NOT IN (55) AND ac.audit_id = ar.audit_id AND create_time > ? AND create_time < ? AND user_group_id IN (200) ORDER BY create_time DESC
Work Item Assign	LECT am.target2_link_value as work_item_id, create_time, am.target_link_value as newowner, created_by, user_group_name FROM audit_misc am, audit_core ac , work_item w WHERE ac.work_item_id = w.work_item_id AND w.status NOT IN (55) AND audit_type='ASSIGN' and ac.audit_id = am.audit_id AND create_time > ? AND create_time < ? AND user_group_id IN (200) ORDER BY create_time DESC

AUDITLOG provides a way to resolve audit records back to various application-specific entities, such as users, work items, book of business transfers, etc.

The underlying audit data model is composed of the following properties:

- actor - identifies who committed the action or operation
- timestamp - when the action or operation was performed
- action code - identifies the operation that was executed
- one or more targets - identifies the target(s) of the action or operation
- success or failure
- primary user group of the current user

This includes a straightforward Java API, a simple data model and the mechanics for the persistence of audit records to the database. Audit records are generated by the application or product making explicit calls to the API exposed by this component. The decision on what activities to track and the degree of granularity is then considered the sole responsibility of the hosting/calling application.

### Example

John Doe added Bill Smith to the role agent. There are two targets: Bill Smith the user and the "agent," which is a role group. The audit record for this example might look like the following:

action_id	100	Target_link_short_name	"Bill Smith"
actor_link_value	"1001"	Target2_link_value	"800"
actor_link_type	"subjectId"	Target2_link_type	"roleGroup"
actor_link_short_name	"John Doe"	Target2_link_short_name	"agent"
action_code	"Add"	create_time	11/01/2008 11:32:00AM
Target_link_value	"1020"	user_group_id	"2001"
Target_link_type	"subjectId"		

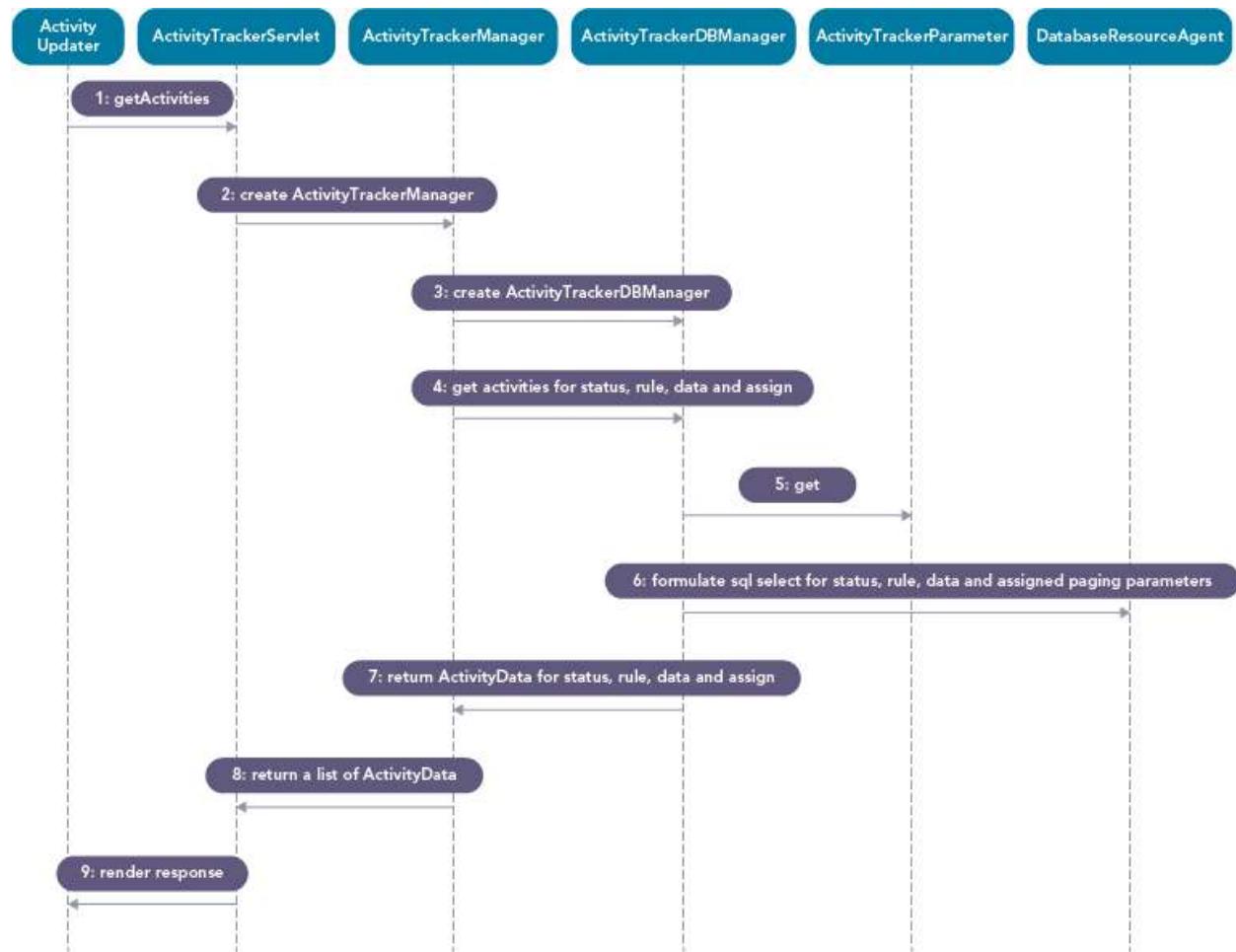
# Event Display

The SDK provides support to send back events for applications to render. ActivityTrackerServlet serves all periodic update request. ActivityTrackerServlet delegates the period request processing to ActivityTrackerManager. Applications can create their own ActivityTrackerManager and notify the system with property `activity_tracker.manager.class`. ActivityTrackerManagers asks ActivityTrackerDBManager to retrieve work item activities for status change, data modification, rule events, assignment and custom activities. Applications can create their own ActivityTrackerDBManager and notify the system with property `activity_tracker.database.manager.class`.

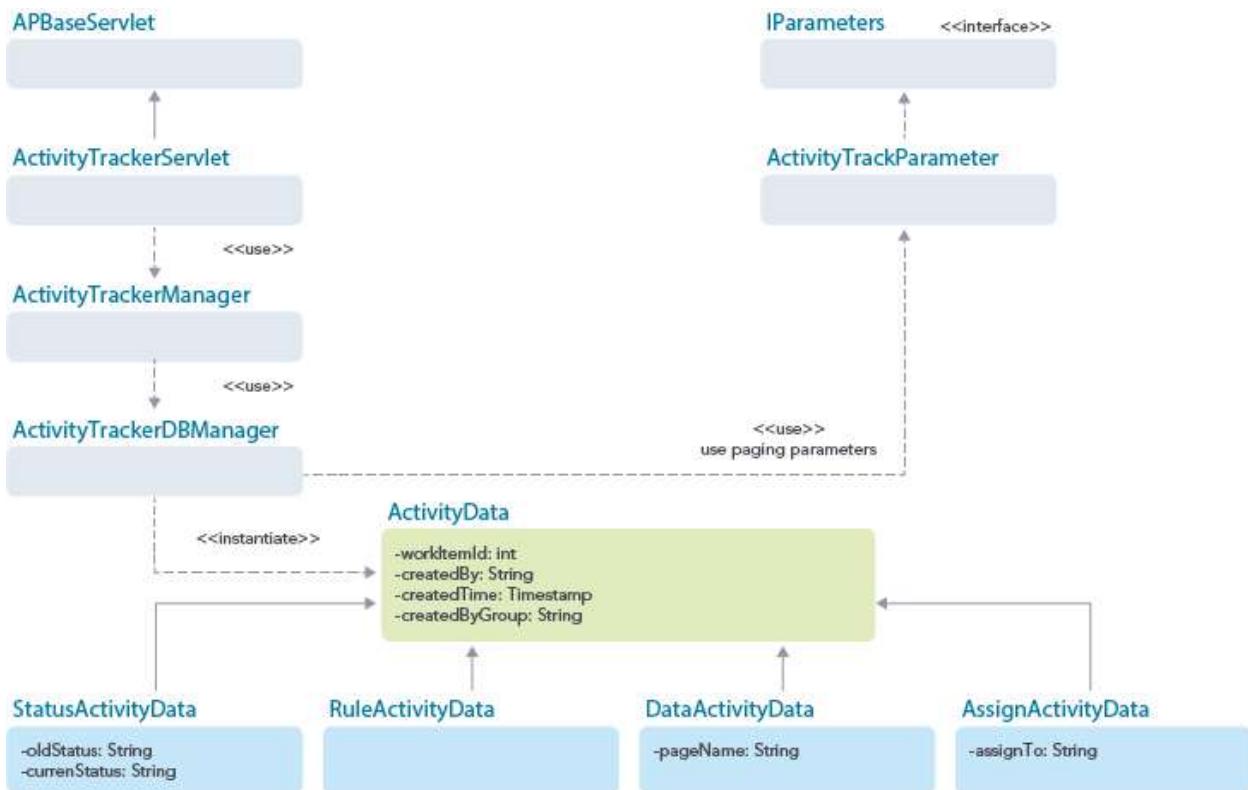
ActivityTrackerDBManager formulates database queries for specific activities and appends paging parameters with help of database specific resource agents.

The activities to track can be limited. If activities are too old, they can be kept out of tracking with property `activity_tracker.start_track_days`. The default value is 20 days. All the activities created 20 days ago will not be tracked.

The following sequence diagram shows the workflow:



The following diagram represents the class model:



Bean classes are used to encapsulate the activity data. Four types of activities are created separately for status change, rule event, data modification and work item assign. All the bean classes are extended from base bean class `ActivityData`. Applications can extend `ActivityData` to create custom activities. Application developers can add their custom activity data.

The following are JavaScript samples to send periodic requests to the server to retrieve specific events:

```

function ActivityTracker(){ this.activityURL = "ActivityTracker"; this.start = function(updateInterval){ if(updateInterval > 0) { this.update(); new PeriodicalExecuter(function() {this.update();}.bind(this), updateInterval); } } this.update = function(){ var ajaxReqParams = {};
ajaxReqParams.LATEST_CREATION_TIME = this.latestCreationTime; new Ajax.Request(this.activityURL, { method:'post', parameters: ajaxReqParams, onSuccess:function(response){this.updateSuccess(response);}.bind(this), onFailure: function(response){this.updateFailure(response);}.bind(this) }); } this.updateSuccess = function(response) { //render events on page }
this.updateFailure = function(response) { //handle errors } }
```

Applications can determine how often the UI should be updated and how the events are rendered. The following are some sample events:



**Activity Tracker**

Work Item #4100 has rule firing events by Great Salesman

Work Item #4100 has rule firing events by Great Salesman

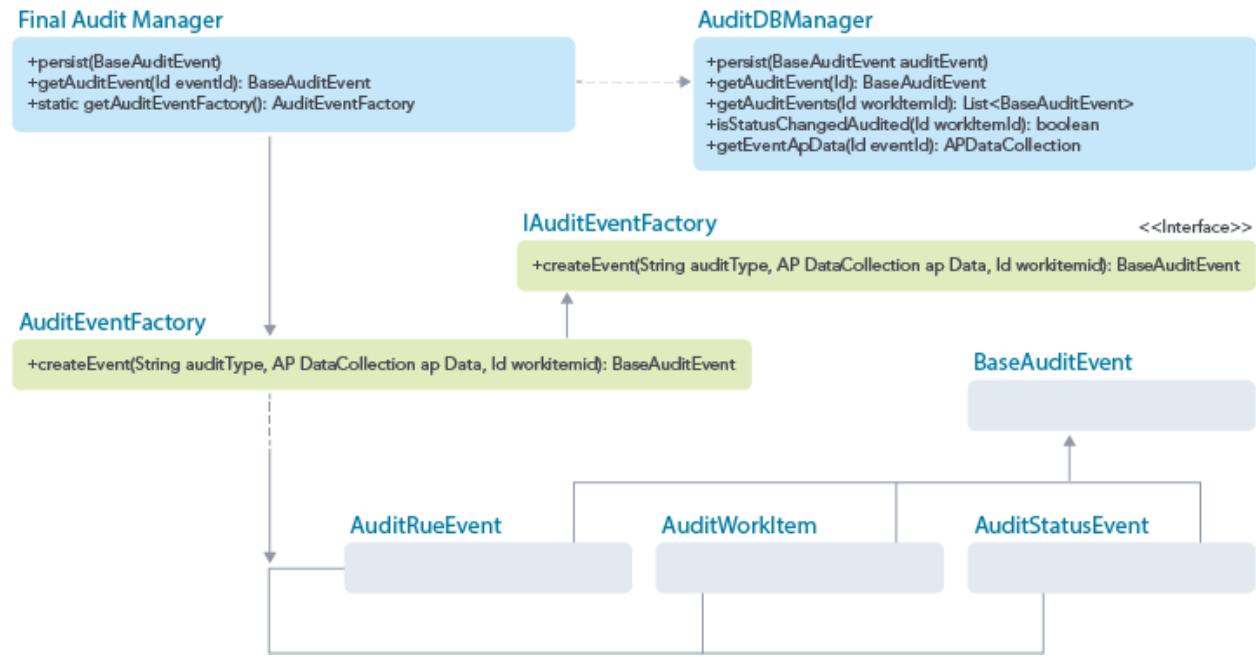
Work Item #4078 has rule firing events by Great Salesman

Work Item #2054 has rule firing events by Great Salesman

Work Item #2004 has rule firing events by Great Salesman

# Event Audit API

These auditing APIs are in the `com.agencyport.audit.core` and `com.agencyport.audit.core.events` packages. The following is the class diagram that describes these APIs:



Event creations are done through `AuditEventFactory`, which supports the following events: STATUS\_CHANGE, RULE, WORKITEM\_ADD, DATA\_MODIFIED, SERVICE\_DATA, SECURITY\_ADMIN, SETTINGS\_ADMIN, LOGIN, FRONT\_DOOR, ASSIGN, TOOLKIT\_SAVE, TOOLKIT\_IMPORT, TOOLKIT\_DELETE, TOOLKIT\_EXPORT, TOOLKIT\_CREATE. Application developers can create custom events and handle custom persistence.

## Code Samples

### Audit Work Item Status Change

```

void audit(int oldStatus, IWorkItem workItem, ISecurityProfile securityProfile, DatabaseResourceAgent dbResourceAgent) throws
APEException{ AuditStatusEvent statusEvent =
(AuditStatusEvent)AuditManager.getAuditEventFactory().createEvent(IAuditConstants.AUDIT_TYPE_STATUS,
workItem.getDataCollection(), workItem.getWorkItemId()); statusEvent.setOldStatus(oldStatus);
statusEvent.setStatus(workItem.getStatus().getStatusCodeValue().intValue());
statusEvent.setCreatedBy(securityProfile.getSubject().getId().toString()); IUserGroups userGrps = securityProfile.getUserGroups(); if(null != userGrps){ IUserGroup grp = userGrps.getPrimaryUserGroup(securityProfile.getSubject().getPrincipal()); if(null != grp){
statusEvent.setGroupId(grp.getId()); statusEvent.setGroupName(grp.getName()); } } AuditManager auditManager= new
AuditManager(dbResourceAgent); auditManager.persist(statusEvent); }

```

### Work Item Assign

```

void addAssignAuditRecord(ISecurityProfile securityProfile, IWorkItem workItem, Id auditOwnerLink, String message,
DatabaseResourceAgent dbResourceAgent) AuditLink actor = securityProfile.getSubjectAuditLink(); AuditLink ownerAuditLink =
AuditLink.create(auditOwnerLink, IAuditLinkTypeConstants.STATUS, "owner"); BaseAuditEvent auditEvent =
AuditManager.getAuditEventFactory().createEvent(IAuditConstants.AUDIT_TYPE_MISC_WORKITEM_ASSIGN, null, null);
AuditMiscEvent miscEvent = (AuditMiscEvent)auditEvent; miscEvent.setActionCode(IAuditActionCodeConstants.UPDATE);
miscEvent.setActorShortName(actor.getShortName()); miscEvent.setActorType(actor.getType());
miscEvent.setActorValue(actor.getKey().toString()); miscEvent.setTargetShortName(ownerAuditLink.getShortName());
miscEvent.setTargetType(ownerAuditLink.getType()); miscEvent.setTargetValue(ownerAuditLink.getKey().toString()); if(null != workItem.getAuditLink()){
 miscEvent.setTargetShortName2(workItem.getAuditLink().getShortName()); }
if(null != message){ miscEvent.setDescription(message); } miscEvent.setWorkItemId(workItem.getWorkItemId());
miscEvent.setCreatedBy(securityProfile.getSubject().getId().toString()); IUserGroups userGrps = securityProfile.getUserGroups(); if(null != userGrps){ IUserGroup grp = userGrps.getPrimaryUserGroup(securityProfile.getSubject().getPrincipal()); if(null != grp){

```

```
miscEvent.setGroupId(grp.getId()); miscEvent.setGroupName(grp.getName()); } } AuditManager auditManager = new AuditManager(dbResourceAgent); auditManager.persist(statusEvent); }
```

# Localization

All text destined for a user's eyes that is associated with the framework is included in special files (resource bundles and Agencyport's locale aware data model) to make it easy for implementation teams to change the application language. This feature also provides you with the ability to control display text via locale-based resource bundle entries instead of hard coded values. Other component features include:

- Access to locale specific data at all layers:
  - client presentation (JavaScript)
  - server presentation (JSP)
  - business (Java)
  - database access layers (SQL)
- All core product JSP, JavaScript and Java that historically had English text embedded has access to resource bundles. This includes data validation, XARC messaging, work list headings and prompts, dashboard titles, legends, etc. However, TDF sourced content, the toolkit, work item specific data and PDF forms are not included.
- A single product Java interface serves as a complete inventory of all resource bundle names used by the core product.
- Applications can leverage the same Agencyport locale support framework for their own purposes. Application do not have to package resource bundles as JAR files or place them in the application class path, which is customary when using the JDK resource bundle class itself.
- Allow UTF-8 or ISO 8859-1 compatibility with a single application property setting.

# Product Definition

Product definition has resource bundle components in which you can define specific languages. This means that after creating a single product definition, application teams can then translate their applications into multiple languages by taking the following additional steps:

1. Translate all code built into the product in to a new resource bundle.
2. Translate and save all editable text for the product definition (either from an existing product template or a newly created line of business).

## Tip

A program is available to extract all editable text from the product definition to make this step simpler.

3. Update the localization property file to point to those files and set the user's browser to the selected language.

Development teams must maintain a single TDF.

# Property Structure

The AgencyPortal framework supports the storage of locale-sensitive data held in flat files whose format conforms to the standard Java property file format.

Resource bundle data items are basically name/value pair where the names or keys are shared across locales and the values are localized. Keys follow a hierarchical pattern whereby each level in the hierarchy is delimited with the dot(.) character. The generalization of that pattern is:

```
<type>.<category 1>.<category n>.<distinct key >=localized value
```

Type can have a value, such as message, title, action or helpText. It describes the type of the property. Zero or more categories can follow the type, which can further refine the property key. The following is an example:

```
message.Security.LoginFailure=Login failure. ||| Type Category Key Value
```

## Example

The following is a snippet of a Spanish version of the core\_prompts\_es.properties file:

```
#Generated by ResourceBundle Editor (http://eclipse-rbe.sourceforge.net) ##### # Spanish Resource Bundle for all of the prompts, messages, text strings # which need to be sent out to the browser #####
action.Actions=Acciones
action.Add=addagregar
action.Back=volver
action.Cancel=cancelar
action.Clear=Perd\u00ed
action.ClickHereToAddTitle=Haga clic aqu\u00ed para a\u00f1adir ${0}
action.ClickHereToAddTitle.AccessKey.Default=[alt-a]
action.ClickHereToAddTitle.AccessKey.Gecko=[shift-alt-a]
action.ClickHereToAddTitle.AccessKey.Opera=[shift-esc a]
action.ClickHereToContinue=Haga clic aqu\u00ed para continuar.
action.ClickToDeleteThisItemTitle=Haz clic aqu\u00ed para eliminar este elemento
action.ClickToEditThisItemTitle=Haga clic para editar este tema
action.ClickToMakeCopyTitle=Haz clic aqu\u00ed para hacer una copia de este tema
```

Eclipse provides a plug-in to facilitate resource bundle property editing, which is useful. Refer to the [Eclipse ResourceBundle Editor](#) site for more information.

## Component Descriptions

The following are the components included:

- Local Support Package (`com.agencyport.locale` Java package in apbase.jar) - container for the public interfaces and implementation classes supporting locale resolution and resource bundle data access.
- Character Encoding Package (`com.agencyport.util.text` Java package in apbase.jar) - container for the public interfaces and implementation classes supporting HTML encoding and character encoding for web pages.
- Resources - container for all product oriented property file oriented resource bundles. This provides a single point of reference for all resource bundles that a localization effort needs to translate. These resources are currently contained in the apwebapp.jar

## Server Side Access to Locale Aware Data Values

Both programmatic access and declarative access to resource bundle data values is supported. Programmatic access is supported by the `com.agencyport.locale` Java package. You can declare resource bundle properties in the message text area of a XARC rule. The declarative form of a resource bundle property name follows the following pattern:

```
rb:<bundle base name>:<property key>
```

(where rb = a constant value denoting that this is a declarative resource bundle property > colon delimiter > the resource bundle base name > colon delimiter > property key)

If the declarative resource bundle property name is part of a larger string instance, it should be delimited within a \${} sequence.

## Example

The following is an example of a declarative resource bundle property in a XARC rule:

```
<rule name="driverTrainingRule" title="Driving Training Credit"> <description>Driving Training Credit</description> <action> <message severity="10"> <text>rb:persauto_messages:message.XARC.gooddriver</text> </message> </action>
```

The following is an example of a declarative resource bundle property in a dashboard report:

```

INSERT INTO report (report_id, is_drill_down, sql_id, default_chart_id, default_list_id, drill_down_report_id, report_name, report_description, permission, rpt_engine_class_name, rpt_servlet, more_chart_settings) VALUES(100, 0, 100, 1, NULL, 104, 'Total Submissions', 'submissions by LOB', 'canViewTotalSubmissions', NULL, NULL, '<settings><decimals_separator> ${rb:dashboard:format.DecimalSeparator}</decimals_separator> <precision>0</precision> <data_labels> <show></show> </data_labels> <balloon> <enabled></enabled> <color></color> <alpha></alpha> <text_color></text_color> <text_size>13</text_size> <show> <![CDATA[${rb:dashboard:balloonText.report100}]]> </show> </balloon> <labels> <label> <x>0</x> <y>20</y> <rotate>false</rotate> <width></width> <align>center</align> <text_color></text_color> <text_size>13</text_size> <text> <![CDATA[${rb:dashboard:title.report100}]]> </text> </label> </labels> </settings>');

```

where the following dashboard resource bundle properties are available:

```

format.DecimalSeparator= ballonText.report100={title}: {value} work items ({percents}%) title.report 100=Work Items by Product between \\${start_date} and \\${end_date}

```

#### Note

Character sequences, such as {title}, {value} and {percents} are keywords to amcharts (not to be confused with \${} sequences).

The escaping of the start and end date \${} variables allows the regular expression parser used by the ResourceBundleStringUtils class within the dashboard report engine to distinguish these variables from some sort of resource bundle property ID. Any variable directly relating to a replaceable dashboard report parameter must be escaped in this manner.

The following is a JSP snippet exemplifying how you should leverage the AgencyPortal resource bundle API:

```

<%@page import="com.agencyport.locale.ILocaleConstants"%> <%@page import="com.agencyport.locale.IResourceBundle"%> <%@page import=%@page import="com.agencyport.locale.ResourceBundleManager"%> =com.agencyport.locale.ResourceBundleStringUtils%> <% ResourceBundle rb = ResourceBundleManager.get().getHTMLEncodedResourceBundle(ILocaleConstants.CORE_PROMPTS_BUNDLE); %> <div> <p><%=ResourceBundleStringUtils.makeSubstitutions(rb.getString("action.Confirm.ExitTitle"), "", "")%></p> <p><%=rb.getString("action.Confirm.ExitQuestion")%></p> <p> <input name="yesButton" type="button" value="<%=rb.getString("action.Yes")%>" accesskey="o" onclick="lb_handleYes()" /><input name="noButton" type="button" value="<%=rb.getString("action.No")%>" onclick="lb_handleNo()" /> </p> </div>

```

where the following core\_prompts resource bundle properties are accessible:

```

action.Confirm.ExitTitle=There are ${0}unsaved changes${1}.
 action.Confirm.ExitQuestion=Leave Anyway?
 action.Yes=Yes
 action.No=No

```

#### Client Side Access to Locale Aware Data Values

You may need locale sensitive values for the client side for various reasons. Client side access uses the same resource bundling naming conventions as the server side.

There are two parts in play: for a given resource bundle base name (i.e., core\_prompts), all of the property keys and locale specific data values can be easily made part of the page image by simply calling the AgencyPortalJSP tag ap\_rb\_loader.tag.

All SDK rendered pages, including dashboard reports, push the core\_prompts resource bundle in its entirety across to JavaScript. If you need to push your own set of custom values over to the client, then this tag accepts a parameter, which is the base resource bundle name. This JSP accepts the base resource bundle name, default or explicit, and renders a JavaScript "map" wherein the resource bundle name is the name of the JavaScript container variable and the property key is the lookup value within the container. It puts the container variable under the ap.namespace.

The tag directive:

```
<ap:ap_rb_loader rbname="account_management"/>
```

emits the following JavaScript:

```

<script type="text/javascript">var account_management = { "menu.wi.title.ReferenceNum.Short":"Reference # ${0}", "action.Revert.UnavailableOptionMessage.2":"action.Save":"Save", "action.Edit":"Edit", "action.Ok":"Ok", "action.Add":"Add" ... } ap.core_prompts = core_prompts; </script>

```

#### Note

The AgencyPortal JavaScript applyPositionalVariableSubstitutions() method, which takes a message template followed by one or more dynamic arguments, applies substitution wherever a \${n} sequence is encountered in the template with the nth dynamic argument.

## Example

The following is an example using the resource bundle properties:

```
action.MaxNumberOfItemsReached=The maximum number of items has been reached. ${0} action.MaxNumberOfItemsReached.1=You must delete at least one entry before this item can be restored.var actionPrompt = applyPositionalVariableSubstitutions(ap.core_prompts["action.MaxNumberOfItemsReached"], ap.core_prompts["action.MaxNumberOfItemsReached.1"]);
```

This statement replaces the \${0} in the `action.MaxNumberOfItemsReached` property with the value of the `action.MaxNumberOfItemsReached.1` property

## Database Access to Locale Aware Data Values

In some instances, a database oriented resource bundle is necessary to have the localized values accessible from SQL. Database locale identity is resolved through the `locale_entries` product table. For every unique combination of language code, country code and variant an application expects to support, there can be a unique record. Records sharing the same language code can refer to the same `locale_id`, if desired.

### Important

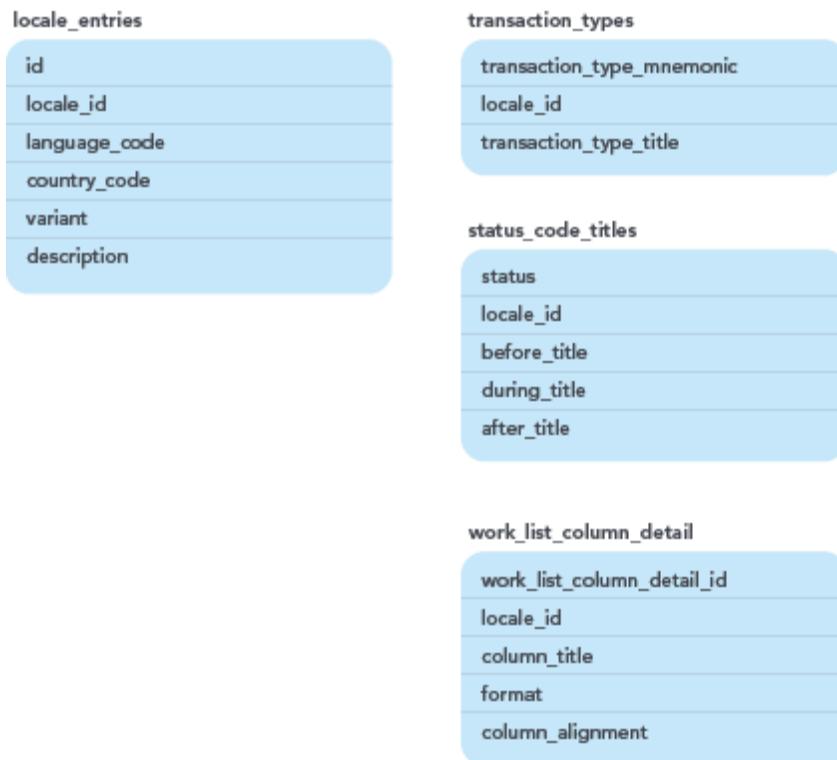
It is important to note that there is a difference between the JDK resource bundle access mechanism and the one supporting Agencyport's database resource bundle access. The former one traverses an access route of language, country and variant on a per key basis, serving up values that could potentially be sourced from different property files within a unique resource bundle base name. The access path for Agencyport's database oriented resource bundle data values assumes a much simpler access path wherein, for a given locale, there is only one `locale_id` evaluated for a given locale. The implication of this is that for each distinct `locale_id` value registered within an application, a complete set of records must exist on all of the locale aware data tables for that locale.

The schema of the `locale_entries` table is as follows:

Column Name	Business Description	Data Type	Nullable	Default	Primary Key	Foreign Key
id	Primary key identity of this physical record. This is not the locale ID.	Integer	false	N/A	true	false
locale_id	Numeric locale identity value used to express the specific locale	Integer	false	1 - denotes the default locale (English)	false	false
language_code	ISO language code value	char(2)	false	N/A	false	false
country_code	ISO country code	char(2)	false	* - an asterisk will act as a wild card character	false	false
variant	3rd qualifier in locale	carchar(16)	false	* - an asterisk will act as a wild card character	false	false
description	user friendly description of this locale	varchar(64)	true	N/A	false	false

A `com.agencyport.locale.Locales` product class accepts a Java locale instance and is responsible for selecting the record in the `local_entries` table that best matches the language code, country code and variant properties of a given locale. The `locale_id` from the best matching record is then used to filter records from other product and custom application tables that serve up locale specific data values. The following diagram illustrates the `locale_entries` table alongside three product tables that contain records that need filtered by locale. Applications can leverage this same technique.

The following diagram depicts the `locale_entries` table and a local aware data table:



Navigation into locale aware tables works in the following way:

- The best `locale_entries` record is selected based on the language code, country code and variant of the locale currently in context. An exact match is attempted first, followed by various attempts that end up finally just comparing the language. If none of these match, a default `locale_id` of 1 is returned.
- The selected `locale_id` is used (in part) to select records from various locale aware data tables.

The following Java code illustrates how to access data from a locale aware data table. In particular, the `Locales.getLocaleId()` line of code, whose return value is rendered on the prepared statement, is salient.

```

/** * The TransactionTypeProvider class is the resource data bundle provider implementation for transaction * type localization. * @since 4.1 */
public class TransactionTypeProvider extends BaseDBProvider implements IResourceBundleDataProvider { /** * The
<code>selectTransactionTypesByLocale</code> is the select statement for retrieving records from the transaction * type table by locale. */
private static final String selectTransactionTypesByLocale = databaseAgent.normalizeSQLStatement("select transaction_type_mnemonic,
transaction_type_title from ${db_table_prefix}transaction_types where locale_id = ?"); /* (non-Javadoc) * @see
com.agencyport.locale.IResourceBundleDataProvider#load(com.agencyport.locale.ResourceBundleName, java.util.Locale) */
public Map<String, String> load(ResourceBundleName resourceBundleName, Locale locale) { try { Map<String, String> bundle = new
HashMap<String, String>; dbResourceAgent.getConnection(true); PreparedStatement pstmt =
dbResourceAgent.getPreparedStatement(selectTransactionTypesByLocale); pstmt.setInt(1, Locales.get().getLocaleId().intValue()); ResultSet rs
= dbResourceAgent.executeQuery(pstmt); while (rs.next()) { String mnemonic = rs.getString("transaction_type_mnemonic"); String title =
rs.getString("transaction_type_title"); bundle.put(mnemonic, title); } return bundle; } catch (Exception exception){
ExceptionLogger.log(exception, this.getClass(), "load"); return null; } finally { dbResourceAgent.closeDatabaseResources(this); } } }

```

## Date Format and Masking

Date and timestamp display formatting and masking is centrally controlled by a format information resource bundle. The formatting information drives the display of dates and timestamps in dashboard reports, work list columns, date fields on SDK pages, as well as date fields on policy summary and policy change summarized pages. This allows an application to alter the default date format by changing one value in one resource bundle without the need to change every date field in the TDF. The SDK date field element automatically inherits this feature.

## Configuration

This section details the few configuration parameters relating to this initiative:

- locale.resource\_bundle\_paths** - this application property controls where resource bundles are loaded from the file system.

## Note

All product base resource bundles are automatically loaded from apwebapp.jar. This configuration parameter does not, and should not, account for product base resource bundles.

## Example

```
locale.resource_bundle_paths=${my_context_path}WEB-INF/resource_bundles/
```

- **application.character\_encoding** - allows an application to change the default character encoding from UTF-8 back to ISO 8859-1.  
There are two supported values: UTF-8 and ISO-8859-1.

## Extending the Base Resource Bundles for Custom Applications

All custom resource bundles should be extended from the same interface to allow for centralization of resource bundle references. Use the following example as a guideline:

## Example

```
package com.agencyport.shared.custom; import com.agencyport.locale.ILocaleConstants; import com.agencyport.locale.ResourceBundleName;
/** * The IApplicationLocaleConstants interface serves as an example on how to extend * AgencyPortal's ILocaleConstants interface in the
proper manner. */ public interface IApplicationLocaleConstants extends ILocaleConstants { /** * The
<code>CUSTOM_RESOURCE_BUNDLE</code> */ ResourceBundleName CUSTOM_RESOURCE_BUNDLE = new
ResourceBundleName("custom"); }
```

## Extending Default Resource Bundle Properties Using English Properties

You can extend the default resource bundle properties by overriding selected properties in a \_en.properties bundle. This technique is useful when only a few properties need overridden.

## Example

If you want to override the following properties in the core\_prompts.properties file:

```
action.Help=Help action.Logout=Logout
```

All you need to do is create an English language version of the core\_prompts.properties file named core\_prompts\_en.properties in the directory specified by the locale.resource\_bundle\_paths and override only the properties in question, as indicated below:

```
action.Help=Need Help action.Logout=Leave Application
```

## Best Practice

In JSPs, always use the HTML encoded wrapper for a resource bundle data value. You won't need to call `JSPHelper.prepareForHtml()` from your JSP.

For English applications, override only the default resource bundle properties that need overridden. Instead of replicating an entire resource bundle store at the application level, the application can provide a \_en.properties version where only the properties that need overridden are present in the \_en version. Refer to the example in the [Extending Default Resource Bundle Properties Using English Properties](#) section.

For multiple resource bundle data values, get the `IResourceBundle` interface and use it rather than calling `ResourceBundleManager.getString()` over and over again. This is for runtime efficiency.

Don't use Java's `ResourceBundle` class directly. You want a consistent approach across all product and client delivery fronts. Use `com.agencyport.locale.IResourceBundle`.

Don't assume the format of dates. Use the  `ResourceBundleManager` to get the right format appropriate for the locale in context. Refer to the  `ResourceBundleManager.getDefaultDateDisplayFormat` below in this IPDTR code sample, which defaults the expiration date from a user's effective date:

```
if (baseElement.getUniqueId().equals("policyExpirationDate")){ if (origin.getUniqueId().equals("PolicyEffectiveDt")){ String effectiveDate =
getFieldValue(intraPageDTRBehaviorManager, origin.getUniqueId()); if (!BaseValidator.checkIsEmpty(effectiveDate)){ try { APDate
```

```
expirationDt = new APDate(effectiveDate); Calendar calendar = expirationDt.get(); calendar.add(Calendar.YEAR, 1);
baseElement.setValue(expirationDt. getStringDate(ResourceBundleManager.get().getDefaultDateDisplayFormat(
APDate.DEFAULT_DATE_DISPLAY_FORMAT_PATTERN))); //signal to framework that we want the value to be carried to the client return
true; } catch (ParseException pe){} {} }
```

# Localization of AgencyPortal Product Definitions

This section reviews the steps an application team must take to prepare an application for a locale other than US English. Before embarking on such an endeavor, the application team needs to take into consideration the following best practices and guidelines:

1. Take the time to fully read and understand the Localization documentation here on the Developer Zone.
2. Use the approved Eclipse plugin specifically designed for editing resource bundle content when editing property based resource bundle entries. All of the core base resource bundles of the framework, as well as those for the Spanish translation of the personal auto LOB template were authored using this valuable plugin.
3. Place any application properties relating to localization into its own private properties file. The LOB templates use a properties file called `localization.properties`. The following application properties should be included:
  - `local.resource_bundle_paths`
  - `local.local_emulation`
  - `application.is_locale_aware`
  - the resource bundle map, which links the product artifacts to their respective base resource bundles

## Example

The following is an example of such a properties file:

```
#####
localization.properties
Properties specific to localization.

#####
Resource Bundle Path locale.resource_bundle_paths
A semi-colon delimited paths which will be searched when a
resource bundle
is being requested.

#####
language_code=es
application.base.rb_root=${my_context_path}WEB-
INF/resource_bundles

application.spanish.rb_root=${application.base.rb_root}/${lan-
guage_code}

locale.resource_bundle_paths=${application.base.rb_root};${ap-
plication.spanish.rb_root}

#####
Locale emulation
format is <language code>,<country code>,<variant string>
```

```

#####
locale.locale_emulation=
#####

#####
Locale awareness flag defaults to false. Once turned on an
application will
begin to pick up resource bundle translations for product
definitions

#####
application.is_locale_aware=true

#####
Resource bundle map

#####
#rb-map.properties auto generated by
com.agencyport.locale.ResourceBundleGenerator
#Wed Aug 25 10:44:54 EDT 2010
pageLibrary.AUTO=pageLibrary_AUTO

behavior.persautoBehaviors.xml=behavior_persautoBehaviors_xml
tdf.endorsePersauto=tdf_endorsePersauto

xarcRule.agencyApplicantInformation=xarcRule_agencyApplicantI
nformation

optionList.persautoCodeListRef.xml=optionList_persautoCodeListRef_xml
optionList.codeListRef.xml=optionList_codeListRef_xml

optionList.persLinesCodeListRef.xml=optionList_persLinesCodeListRef_xml
tdf.persautoQuickQuote=tdf_persautoQuickQuote
tdf.renewPersauto=tdf_renewPersauto
tdf.persauto=tdf_persauto

workItemAssistant.workitemassistant=workItemAssistant_workite
massistant

```

4. The translation effort (constructing the other language translation of the US English entities) should be deferred as long as possible until an application is stable with regards to the following:

- The configuration and content of product artifacts (TDF, page libraries, XARC rules, option lists) is well-established and not subject to large scale change.
  - The identity attribute values across the various entities are well-established and not subject to wholesale future changes. In general, the more volatile a set of product definition entities are (concerning their identity and their placement in the transaction workflow), the larger the effort to synchronize existing localized content.
5. Before running the resource bundle generator utility, ensure each of the following has first been carried out:
- For any field element not having a unique ID, assign a short and yet meaningful unique ID to those field elements. Unique ids for fields need to be unique within an artifact. Internally, these unique ids partially compose a resource bundle entry key, which is used to access the resource bundle value. Since resource bundle keys are really nothing more than an application property key, the key cannot contain any '=' characters. **Any transaction with one or more unique ids containing XPath predicates must be refactored so that the unique ids do not contain such predicates. The refactoring could have a cascading effect on any existing DTR behaviors or other custom code written against such fields.**
  - Assign a meaningful ID to every field set that has a legend.
  - Any validations and corrections with custom messaging must be assigned a meaningful ID value.
  - Every XARC message must be assigned a meaningful ID value.
6. After the resource bundle generator utility has run, examine the output of the resource bundle generator utility to ensure that neither systemgenerated GUIDs or null values have made their way into the resource bundle entry keys. The presence of either indicates the operations described in the previous step were not thorough. The following are examples of the types of output that must be addressed before a translation effort can be embarked upon:
- An XARC rule example where there is no ID on the message element:

MultiStateRule\_field.null=Using Field - This is a multistate policy

- A TDF field of type message where a GUID ID is rendering because the field element does not have an explicit one:

AACF455EECFA6EA8828A4D20997F08540A.message=Please enter the previous address if the time at current residence is less than 3 years.

- After you've started the translation effort, additional product definition entries may need to be introduced into an artifact. Rerunning the resource bundle generator tool allows you to pick up the translations of existing content. The resultant resource bundles should contain the localized content for the old entities and un-localized (US English) content for the newly introduced entities.
- If the identities of any entities have changed, the related resource bundle entries must be adjusted manually. The task to move content registered under one key in an old resource bundle to its new key in a new resource bundle is a manual one.
- If product entity deletions are committed, then rerun the utility to remove the associated entries from the resource bundles.
- Localization of 3.x Arc rule messaging is not supported.

Other important items to note:

- The `application.is_locale_aware` Boolean application property controls whether the page subsystem attempts to render localized content. The property defaults to `false`, so it must be set to `true` to trigger the access of localized content. Even if an application's product definitions are configured with all of the correct localized resource bundles, the framework will ignore any attempts to use such resource bundles when this flag is `false`.
- Each product artifact resource, TDF, page library, option list, XARC rule and DTR behavior have an associated base resource bundle name. The construct used to draw the relationship between a product artifact resource and its base resource bundle name is the resource bundle map. The resource bundle generating utility creates the resource bundle map. However, the responsibility of adding that map into the application properties (`localization.properties`) is that of the developer.
- The framework uses a technique called key inferencing for relating the various page, option list, rule, and behavior entities to their respective resource bundle entries. This design allows for the complete separation of product artifact content from the localization of that content. As a result, the product artifacts are free from any additional decorations that are locale oriented. The key inferencing algorithm uses the entity ID values partly to build the keys for accessing localized content entry from the resource bundle.

# Localizing Your Content

Except for the localization of work list column details, transaction types, status code descriptions and LOB descriptions, the vast majority of the localized content is contained in property file based resource bundles. The remainder of the localizable content is contained in locale aware database tables. The decision to store the content one way or the other is driven by the need to access localized content directly from SQL select statements.

## Localizing Content Stored in Property File Oriented Resource Bundles

The resource bundle generator utility is located in the apwebapp.jar class. The purpose of this utility is to automate the creation or update of property based resource bundles. These property based resource bundles contain localized content for a set of application product definitions, as well as the core content used by the AgencyPortal framework itself. The resource bundle generator utility is controlled by a combination of command line arguments, as well as application properties. In general, you can point the utility to the framework properties and the WEB-INF directory, where an application's product database resides. This action creates the target locale web content.

To get help on the command line arguments that the utility accepts, pass the -h command line argument. The following helpful text should be emitted when the -h parameter is passed:

```
java -cp apwebapp-5.0.0-SNAPSHOT.jar com.agencyport.locale.ResourceBundleGenerator -h Sep 05, 2014 12:28:59 PM
com.agencyport.locale.ResourceBundleGenerator main INFO: Start resource bundle generator Sep 05, 2014 12:28:59 PM
com.agencyport.locale.ResourceBundleGenerator main INFO: Parsing command line Usage: java
com.agencyport.locale.ResourceBundleGenerator -id <input product root directory>, no default, required -od <output directory to write the
generated resource bundles to>, defaults to ./generated_rbs if not specified -out_rb_map <output resource bundle properties map>, defaults to
./rb-map.properties if not specified -include_products <input include product filter list>, defaults to all products -app_prop <input application
properties to load>, no default, required -locale <input target locale>, defaulted to en_US -include_core <input flag governing whether or not to
render resource bundles for core resource bundles>, defaulted to false -h prints this screen Sep 05, 2014 12:28:59 PM
com.agencyport.locale.ResourceBundleGenerator main
```

INFO: End resource bundle generator

Before you can run the utility, you may have to alter a small number of the framework properties in your application so that it will be able to run successfully from a console application utility. You must change the property that points to the current context path and point to the physical directory on the file since it will not be running in a web application server container.

## Example

```
my_context_path=${application-context}/web/WEB-INF/web/
```

Also, you should probably eliminate all additional property files except for the localization.properties file from the additional\_properties\_to\_load property.

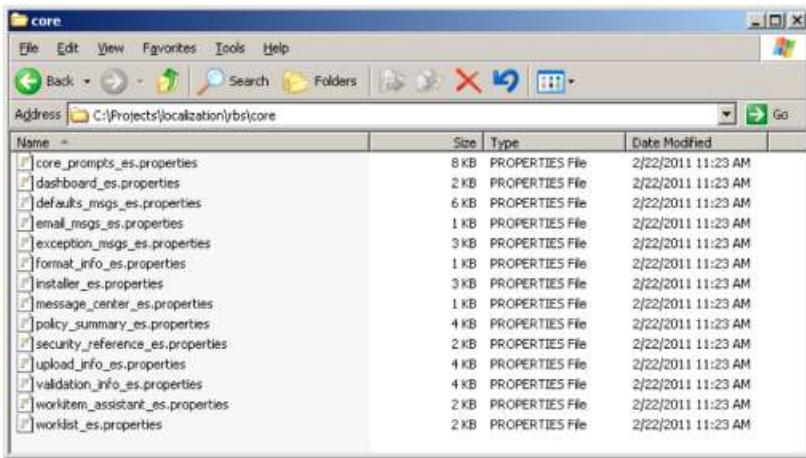
## Example

```
my_context_path=${application-context}/WEB-INF/web/ additional_properties_to_load=${resources_root}localization.properties
```

If you need to pick up existing translated content, then the local\_resource\_bundle\_paths property in the localization.properties file needs to specify the location of those pre-existent resource bundles.

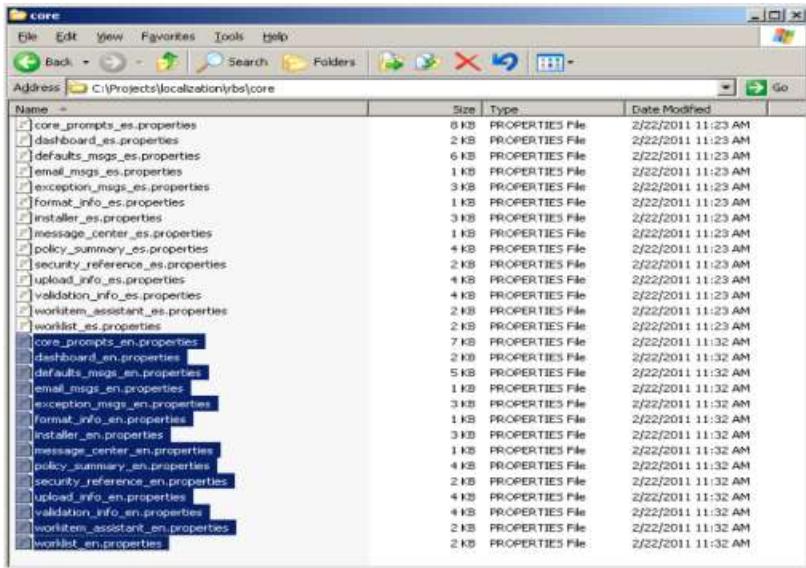
To create the localized content of all the product's core resource bundles for a target locale, run the utility by positioning a command window in WEB-INF/lib directory itself. The command below creates copies of the core product's resource bundles targeting, in this case, Spanish.

```
java -cp apwebapp-5.0.0-SNAPSHOT.jar com.agencyport.locale.ResourceBundleGenerator -id
${application-context}/WEB-INF/ - include_products NONE -locale es -include_core true -od
c:/projects/localization/rbs/core -app_prop
${application-context}/WEB-INF/framework.properties
```

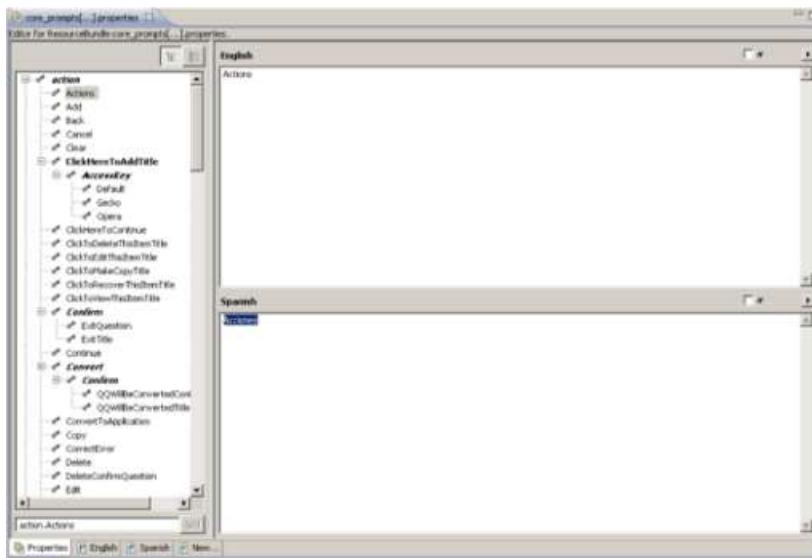


To create the base US English content for all the product's core resource bundles, change the `-locale` argument to `en` and rerun the utility. This places the target content alongside the base US English content, making the resource bundle editor session much easier. The command below creates copies of the core product's resource bundles in the base US English locale:

```
java -cp apwebapp-5.0.0-SNAPSHOT.jar com.agencyport.locale.ResourceBundleGenerator -id ${application-context}/WEB-INF/
include_products NONE -locale en -include_core true -od c:/projects/localization/rbs/core -app_prop
${application-context}/WEB-INF/framework.properties
```



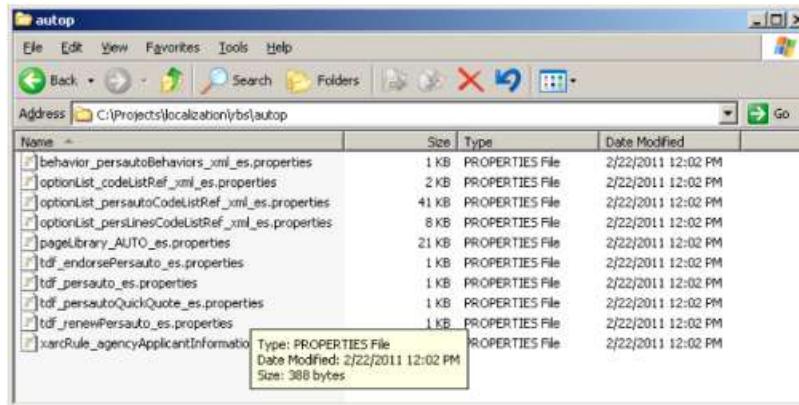
This allows a very pleasant and efficient side-by-side translation experience in the resource bundle editor plug-in for your translation:



To create the localized content for a specific product, run the utility again, passing the product type code (usually the LOB) as the `include_products` argument.

```
java -cp apwebapp-5.0.0-SNAPSHOT.jar com.agencyport.locale.ResourceBundleGenerator -id ${application-context}/WEB-INF/ -include_products AUTOP -locale es -include_core false -od c:/projects/localization/rbs/autop -app_prop ${application-context}/WEB-INF/framework.properties -out_rb_map c:/projects/localization/rbs/resource_bundle_map.properties
```

In this particular case, the above command creates the appropriate resource bundles targeting Spanish for various AUTOP product artifacts.



The resource bundle map property entries file shown below also generates. You need to lift out the entries from this map and into the application's localization.properties file.

```

Notepad++ - C:\Projects\localization\rbs\resource_bundle_map.properties
File Edit Search View Format Language Settings Macro Run Plugins
resource_bundle_map.properties
1 #resource_bundle_map.properties auto generated by com.agencyport.locale.ResourceBundleGenerator
2 #Tue Feb 22 12:02:11 EST 2011
3 pageLibrary.AUTO=pageLibrary_AUTO
4 behavior.persautoBehaviors.xml=behavior_persautoBehaviors_xml
5 tdf.endorsePersauto=tdf_endorsePersauto
6 xarcRule.agencyApplicantInformation=xarcRule_agencyApplicantInformation
7 optionList.persautoCodeListRef.xml=optionList_persautoCodeListRef_xml
8 optionList.codeListRef.xml=optionList_codeListRef_xml
9 optionList.persLinesCodeListRef.xml=optionList_persLinesCodeListRef_xml
10 tdf.persautoQuickQuote=tdf_persautoQuickQuote
11 tdf.renewPersauto=tdf_renewPersauto
12 tdf.persauto=tdf_persauto
13

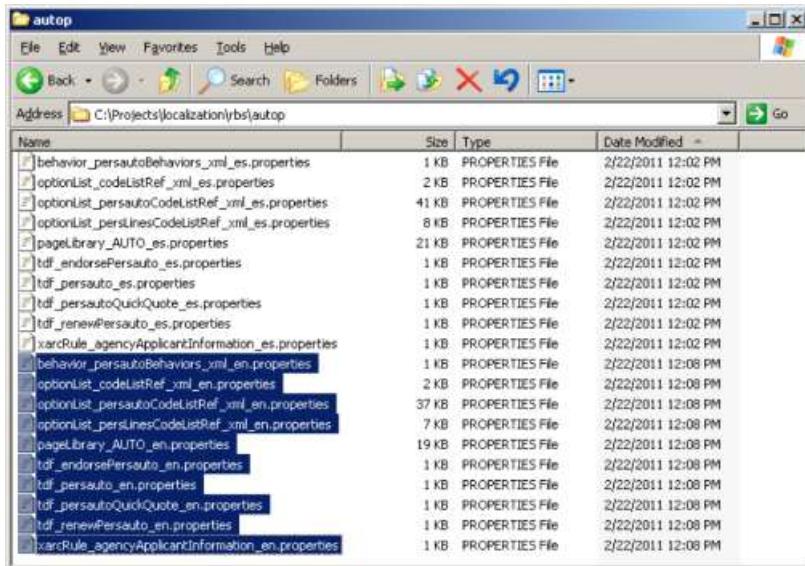
```

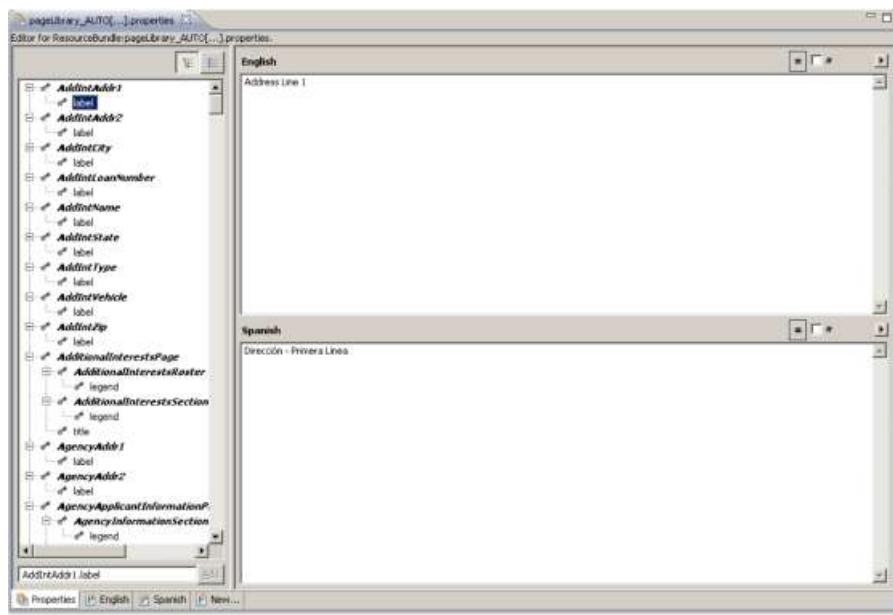
To create an efficient side-by-side editing experience in the resource bundle editor, create the base US English content with the following command:

```

java -cp apwebapp-5.0.0-SNAPSHOT.jar com.agencyport.locale.ResourceBundleGenerator -id
${application-context}/WEB-INF/ -include_products AUTOP -locale en -include_core false -
od c:/projects/localization/rbs/autop -app_prop ${application-context}/WEB-
INF/framework.properties

```





# Performance Logging

The AgencyPortal SDK provides facilities to perform performance testing on our application. There are two main components to this feature:

- **PerfObject** - this class is woven into the SDK and is used to provide timings for a given unit of work. **PerfObject** timings can be engaged by entries in the framework.properties file. Individual teams can add additional units of work to the application as needed.
- **SDK performance package** - this tool is used to take **PerfObject** timing results to create a CSV file for further analysis. The generated CSV file can then be imported into Excel for easy analysis.

## PerfObject

This class is used to measure the elapsed time of a given unit of work within the application. Many units of work are measured after the performance subsystem is enabled, including units of work, such as:

- executeDataStaging
- CMDDisplayPolicySummary.process
- forwardNextPage
- executeUpdateDataAccess
- Overall Total Time
- getResults
- updatePreConditions
- updateMenuController
- initControlData
- com.agencyport.servlets.base.IntrapageDTRServlet.doPost
- Non connector time
- initPage
- XArcConnectorManager
- executeReadDataAccess
- com.agencyport.connector.impl.ConnectorProcessor.run
- BuiltInFieldValConnectorManager
- initMenu
- getTransaction
- and many more

Application teams using the **PerfObject** to enclose the unit in question can add additional units of work.

## Example

Measuring the elapsed time needed to obtain a transaction:

```
PerfObject perfObject = PerfObjectCollector.startNew(TransactionDefinitionManager.class.getName() + ".getTransaction(\"" + transactionName + "\")"); try { return internalGetTransaction(transactionName, true); } finally { perfObject.stop(); }
```

## Enabling the Performance Subsystem

If the performance system is turned on (a log file name displays), the framework writes a number of performance metrics to the output file. The file log characteristics are controllable in the same manner as the base log file (described above).

You can enable the performance subsystem via the framework.properties file using the following properties:

Property	Description	Default Value	Recommended Setting (if any)
PERFORMANCE_RESULTS_FILENAME	If the performance system is turned on (by presence of a log file name), the framework writes a number of performance metrics to the output file.	none	
PERFORMANCE_RESULTS_TRUNCATE_ON_STARTUP	A new performance log file is created on server startup when this	none	

Property	Description	Default Value	Recommended Setting (if any)
	property is set to YES. The default value is NO.		
PERFORMANCE_RESULTS_NUMBER_OF_GENERATIONS		3	
PERFORMANCE_RESULTS_CAPACITY		200000	

### Example

```
#####
PERFORMANCE SUBSYSTEM # This subsystem is turned on by
supplying a file name. # If this file name is not there then the performance collector # is dormant. # The default file open is append (NO). If you
want a fresh copy of the performance # record file on every new Jrun instance then use YES.
PERFORMANCE_RESULTS_FILENAME=${output_dir}perflogs/performance.log
PERFORMANCE_RESULTS_TRUNCATE_ON_STARTUP=NO PERFORMANCE_RESULTS_NUMBER_OF_GENERATIONS=3
PERFORMANCE_RESULTS_CAPACITY=200000
```

This subsystem is turned on by supplying a file name. If this file name is not there, the performance collector is dormant.

## SDK Performance Package

The SDK performance package is used to generate a CSV file from the performance logs, which you can then import into an application, such as Excel, for easy analysis. This package includes a copy of the performance.properties, a performance log parser/analyzer and associated libraries. Click [here](#) to download a ZIP file containing the contents of this package.

Each Agencyport product has an associated property file located in the properties directory. You can use these properties to customize the output generated by the performance package. The following are some of the most useful properties located in the performance.properties file that relate to SDK performance logging:

- `datetime_input_format` - Allows the user to specify output format of date/time.
- `com.agencyport.performance.filters.UnitOfWorkFilter_initargs` - List of events that will be analyzed.
- `track_by_category` - If true, the names in the above property are used and organized in cumulative totals. If false, the actual event names are used.

## Using the Agencyport Performance Log Parser/Analyzer Tool

### Note

This tool only runs on Windows machines.

Before you run the performance log parser/analyzer tool, you must make sure:

1. The run.bat file's JAVA\_HOME parameter points to their local Java instance.
2. The apbase.jar, found in the lib directory of the SDK release, and jdom.jar and xercesImpl.jar, found in the lib-3rdParty directory of the SDK release, are properly referenced in run.bat.

Perform the following steps to set up and use the performance log parser/analyzer tool:

- a. Expand the ZIP file to a destination folder (e.g., c:\temp\performance folder). We will refer to this folder as the \$install\_folder.
- b. Change your working directory to the \$install\_folder. It should look something like the following:

```
C:\temp\performance>dir Volume in drive C has no label. Volume Serial Number is B4FE-F7D7 Directory of C:\temp\performance
03/25/2016 12:05 PM <DIR> . 03/25/2016 12:05 PM <DIR> .. 03/25/2016 12:04 PM <DIR> lib 03/25/2016 12:06 PM <DIR>
properties 03/25/2016 08:27 AM 1,310 run.bat 03/25/2016 12:03 PM 491 runperf.bat 2 File(s) 1,801 bytes 4 Dir(s) 332,538,105,856
bytes free C:\temp\performance>runperf You must pass the location of the performance logs Press any key to continue...
```

- c. Determine the location where your AgencyPortal performance logs are located.

### Example

```
C:\temp\performance>dir c:\runtime\AgencyPortal\perflogs Volume in drive C has no label. Volume Serial Number is B4FE-F7D7
Directory of c:\runtime\AgencyPortal\perflogs 03/25/2016 12:06 PM <DIR> . 03/25/2016 12:06 PM <DIR> .. 03/25/2016 12:06 PM
7,268,987 perf.csv 03/17/2016 09:12 AM 18,754,565 performance.log 10/30/2015 11:20 AM 20,001,916 performance.log.1 10/12/2015
```

```
11:20 AM 20,001,249 performance.log.2 08/25/2015 07:34 AM 20,012,849 performance.log.3 07/13/2015 04:24 PM 20,000,276
performance.log.4 05/01/2015 10:21 AM 20,000,974 performance.log.5
```

- d. Execute the runperf.bat batch file, passing the \$install\_folder as an argument.

#### Example

```
C:\temp\performance>runperf c:\runtime\AgencyPortal\perflogs call run properties/sdk.performance.properties
c:\runtime\AgencyPortal\perflogs\perf.csv c:\runtime\AgencyPortal\perflogs\perf*.log.* JAVA_HOME=c:\Program
Files\Java\jdk1.7.0_45 Running "c:\Program Files\Java\jdk1.7.0_45\bin\java" -Xms128m -Xmx2G -classpath
"lib/jdom2.jar;lib/apbase.jar;lib/jdom2.jar;lib/apbase.jar;lib/jdom2.jar;lib/apbase.jar;lib/jdom2.jar;lib/apbase.jar;"
com.agencyport.performance.PerformanceObjectsAnalyzer properties/sdk.performance.properties c:\runtim
e\AgencyPortal\perflogs\perf.csv c:\runtime\AgencyPortal\perflogs\perf*.log.* Start analysis EnvironmentResourceProvider
[envFilePrefixes=, ${agencyport.env}, ${application.hostname}, nbaker]] Loading contents of resource file :
C:\temp\performance\properties/sdk.performance.properties Loaded: 'properties/sdk.performance.properties' and all related
environment property file content into application properties. Reader class name:
'com.agencyport.performance.reader.PerformanceObjectTextFileReader' Filter class names:
'com.agencyport.performance.filters.UnitOfWorkFilter' Transaction filter class names:
'com.agencyport.performance.filters.TransactionFilter' Reading performance objects from
c:\runtime\AgencyPortal\perflogs\performance.log.1 Number of transactions read: 34979, dropped: 8957, written to db: 26022 Reading
performance objects from c:\runtime\AgencyPortal\perflogs\performance.log.2 Number of transactions read: 17258, dropped: 1199,
written to db: 16059 Reading performance objects from c:\runtime\AgencyPortal\perflogs\performance.log.3 Number of transactions
read: 17741, dropped: 1182, written to db: 16559 Reading performance objects from
c:\runtime\AgencyPortal\perflogs\performance.log.4 Number of transactions read: 18522, dropped: 1676, written to db: 16846 Reading
performance objects from c:\runtime\AgencyPortal\perflogs\performance.log.5 Number of transactions read: 16480, dropped: 1917,
written to db: 14563 Sort class name: 'com.agencyport.performance.sort.ElapseTimeDescending' Analyzing 90049 transactions Writing
output to CSV formatted file: 'c:\runtime\AgencyPortal\perflogs\perf.csv' End analysis Press any key to continue . . .
```

- e. Results are written to the same directory as where the performance logs are located.

#### Example

```
C:\temp\performance>dir c:\runtime\AgencyPortal\perflogs\perf.csv Volume in drive C has no label. Volume Serial Number is B4FE-
F7D7 Directory of c:\runtime\AgencyPortal\perflogs 03/25/2016 12:08 PM 7,268,987 perf.csv 1 File(s) 7,268,987 bytes 0 Dir(s)
332,540,088,320 bytes free
```

- f. Open the CSV file and scroll all the way down to the bottom. The rolled up totals are at the bottom of the worksheet.

## Performance "Math"

Although it may sound counterintuitive, the timing provided by a performance object does not usually equal the sum of the parts that are the performance objects nested inside of that measure. This is because there are many inner units of work that go unmeasured.

#### Example

Seeing something like the following is completely normal and expected:

```
Begin TRAN #46 @2012-12-12 10:58:11:00371 { com.agencyport.security.filter.builtin.PortalSecurityFilter.applyFilter: 2ms
com.agencyport.database.TranManager.getConnectionByJNDI: 0ms DataSource.getConnection: 0ms
com.agencyport.database.TranManager.getConnectionByJNDI: 0ms DataSource.getConnection: 0ms
com.agencyport.database.TranManager.getConnectionByJNDI: 0ms DataSource.getConnection: 0ms
com.agencyport.trandef.TransactionDefinitionManager.getTransaction('commAuto'): 0ms
com.agencyport.trandef.TransactionDefinitionManager.getTransaction('commAuto'): 1ms
com.agencyport.trandef.TransactionDefinitionManager.getTransaction('commAuto'): 1ms TOTAL: 18ms } End TRAN #46 @2012-12-12
10:58:11:00389
```

If you add together the line items, you would expect the total duration of TRAN #46 to take 5ms, yet the TRAN's echo (at the bottom) states that the total was 18ms.

The big thing to remember is that just because "something" in the larger block is measured does not mean that "all things" are measured. Conceivable, there are other blocks of code within the parent TRAN that are not measured, thus the total is not a reflection of the sum of the parts.

The SDK performance logs work like application logging does in that performance objects are placed at "strategic points." They are not instrumentation instructions like how profilers, such as Yourkit work, so, in most cases, you do not have a "complete" breakdown at each level.

In the above example, the TRAN total time is started at the security filter at the earliest point in the workflow that the SDK can get. A nested performance object situation is created when another performance object is started, before the one previous to it is stopped. There can be gaps of

execution between the child and parent, and also between siblings where time is not measured at the level; this is what creates the holds and gaps in the performance logs.

## Note

When running performance tests, it is helpful to disable all non-essential features, such as the debug console, autosave, transaction manager mode, etc. You can disable autosave by setting the following property:

```
auto_save_interval=0
```

Transaction manager mode should be set to cache instead of dynamic via the following property:

```
transaction_file_manager_mode=cache
```

Also, set logging to SEVERE so that writing to logs does not distort the performance results. You can set logging levels in the custom.logging.properties file. The following are examples:

```
com.agencyport.xarc.level=SEVERE com.agencyport.servlets.level=SEVERE com.agencyport.workerscomp.level=SEVERE
com.agencyport.persauto.level=SEVERE com.agencyport.homeowners.level=SEVERE
```

If, during the course of performance testing, stale database connections are common, you can increase the staleness threshold for inactive connections with the following property:

```
#Increase staleness threshold for open connections from the default of 4 seconds #to 30 seconds db_connection_staleness_thres hold=30000
```

If load testing is required, we recommend using JMeter (this is the tool used for SDK performance testing).

In any load testing scenario, you should evaluate the number of database connections available in the context of the number of users/threads that will be involved in the load test. This also applies to the JVM heap size of the web server. In some cases, you will need to increase the number of available connections, as well as the heap size, to accommodate the load test.

# JMeter Instruction Guide

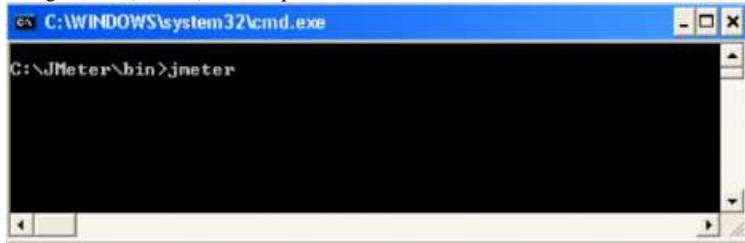
This section details how to install, set up and use JMeter for performance testing:

- [Installing JMeter](#)
- [Recording a Script](#)

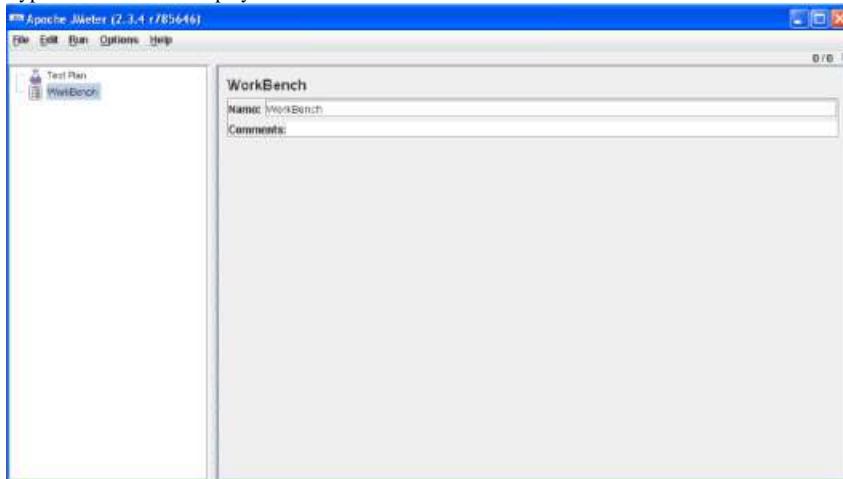
# Installing JMeter

Perform the following to install JMeter for performance testing:

1. Navigate to the [JMeter](#) website.
2. Download the jakarta-jmeter-2.3.4.zip file.
3. Unzip the file to a folder, such as C:\JMeter.
4. Navigate to C:\JMeter\bin and open a cmd window.



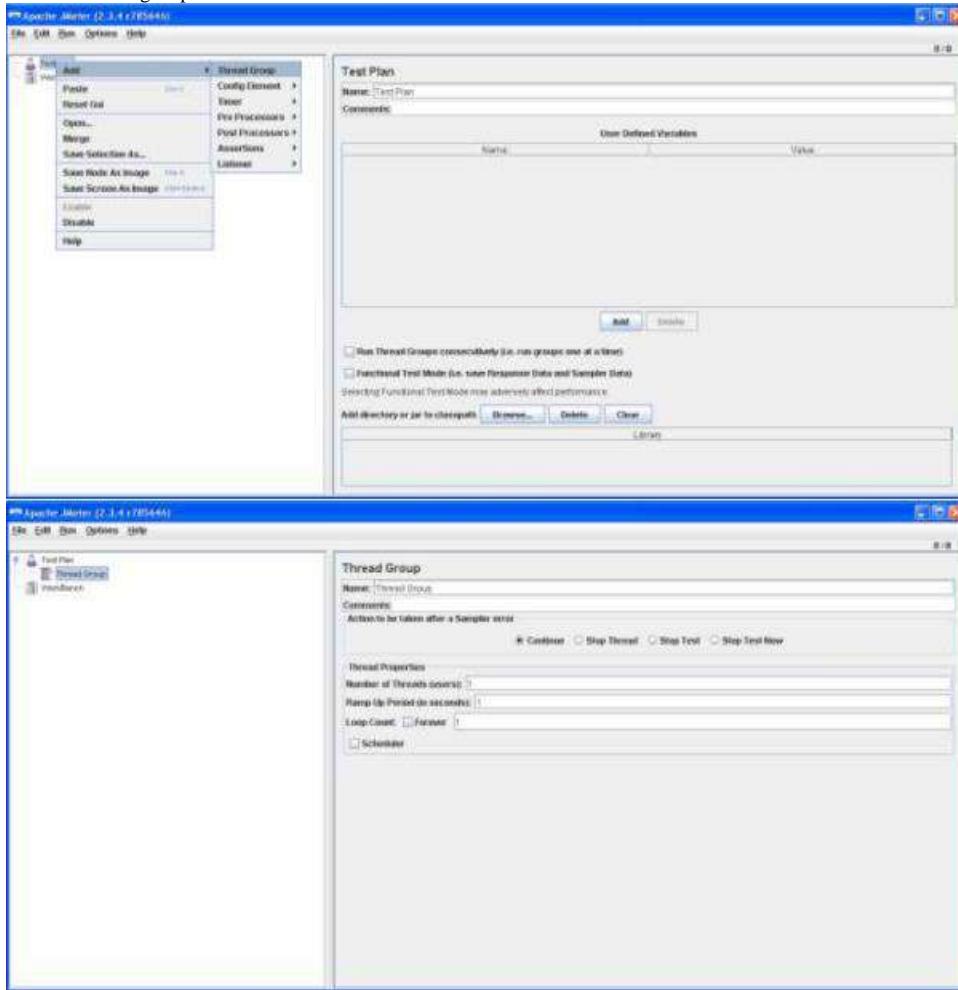
5. Type JMeter. JMeter displays.



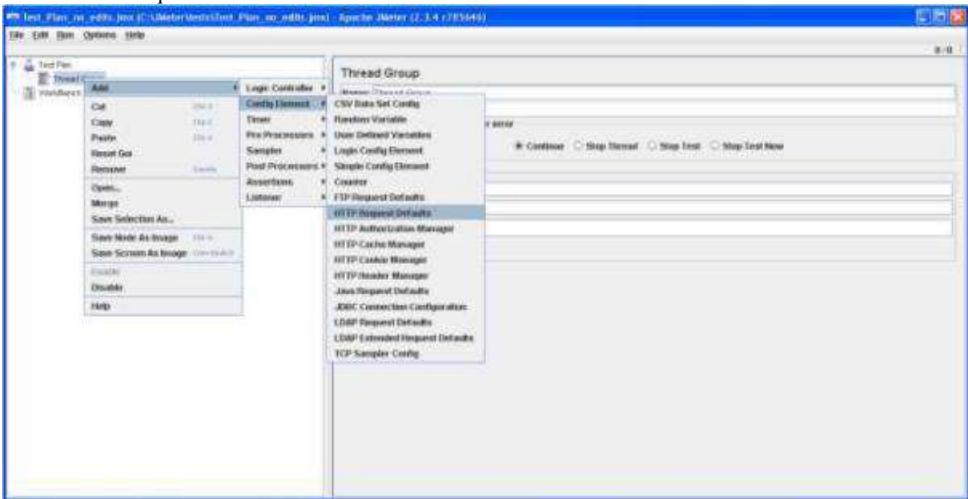
# Recording a Script

Perform the following to record a script via JMeter:

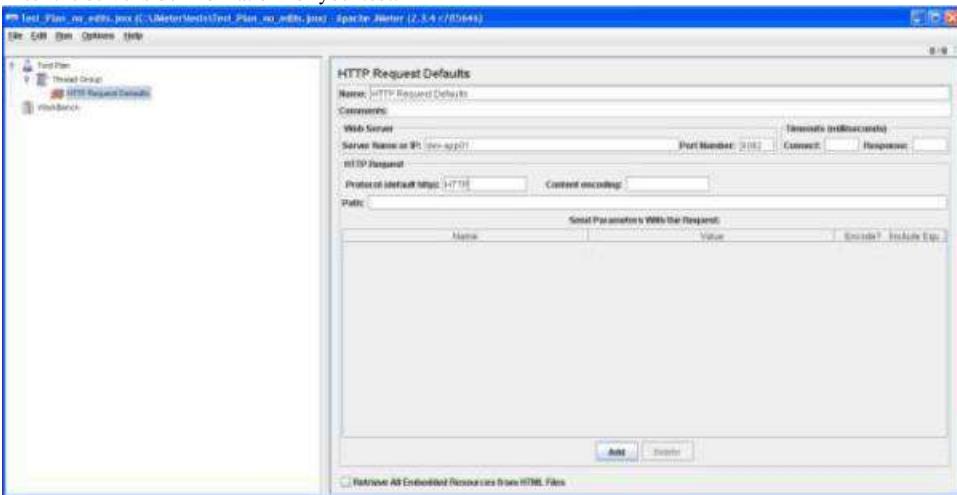
1. Create a thread group.



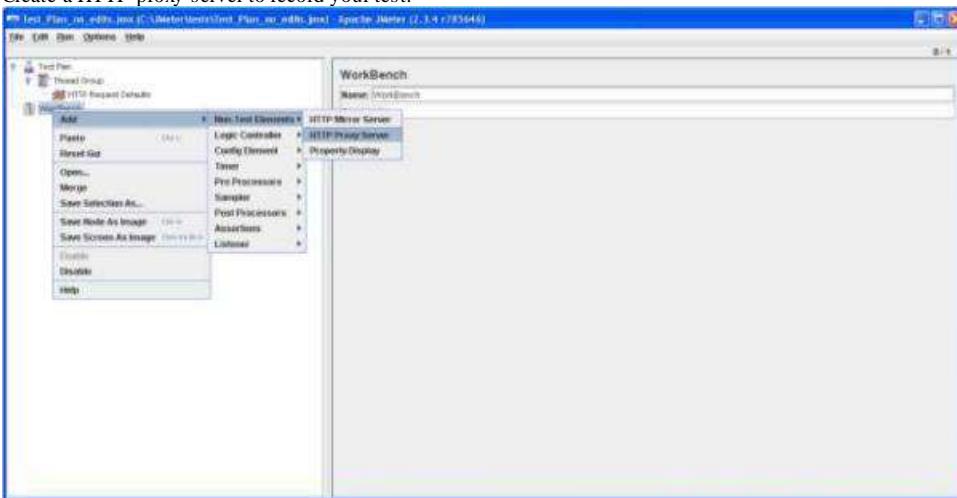
2. Create a HTTP request defaults.



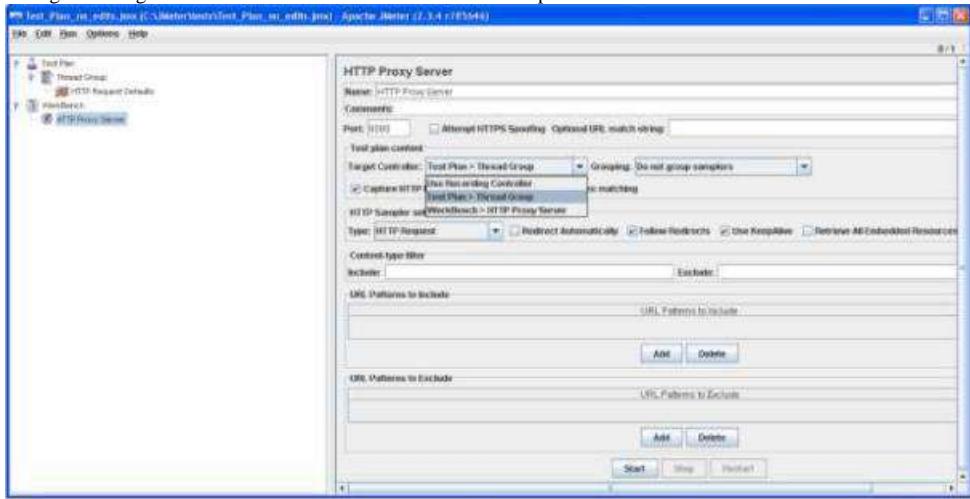
3. Enter the server side information for your test.



4. Create a HTTP proxy server to record your test.



5. Change the Target Controller field to Test Plan > Thread Group.



6. Add the following common extensions that you don't want to record in regular expression format in the URL Patterns to Exclude section (there may be a bug in JMeter that does always filter all of these, but it should save you some time):

- \*.js
- \*.gif
- \*.png
- \*.css
- \*.jsp
- \*.ico
- \*.GIF

#### Tip

Make sure the debug console is turned off in framework.properties so that JMeter does not try to record the debug console:

```
client_debug=false
```

7. Keep JMeter open.  
8. Open Internet Explorer.

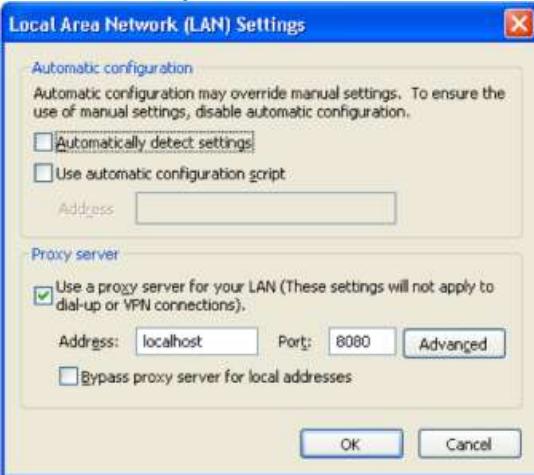
9. Open the Internet Options window from the Tools/Internet Options menu.



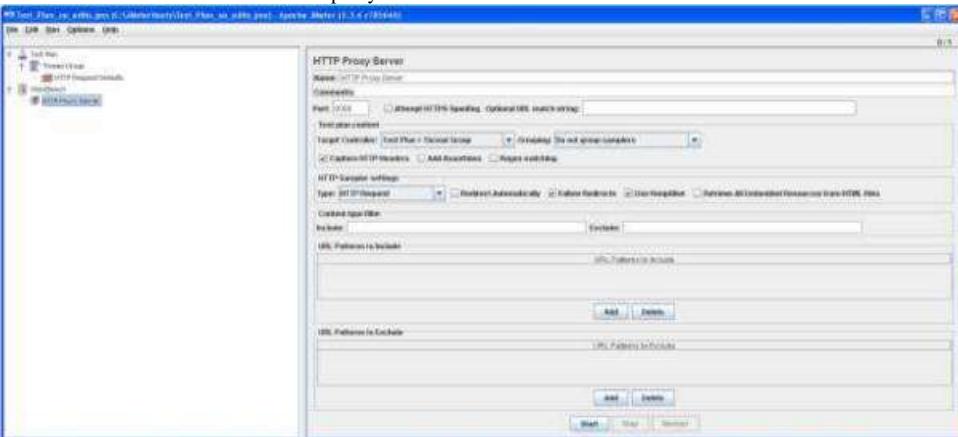
10. Click the Connections tab.



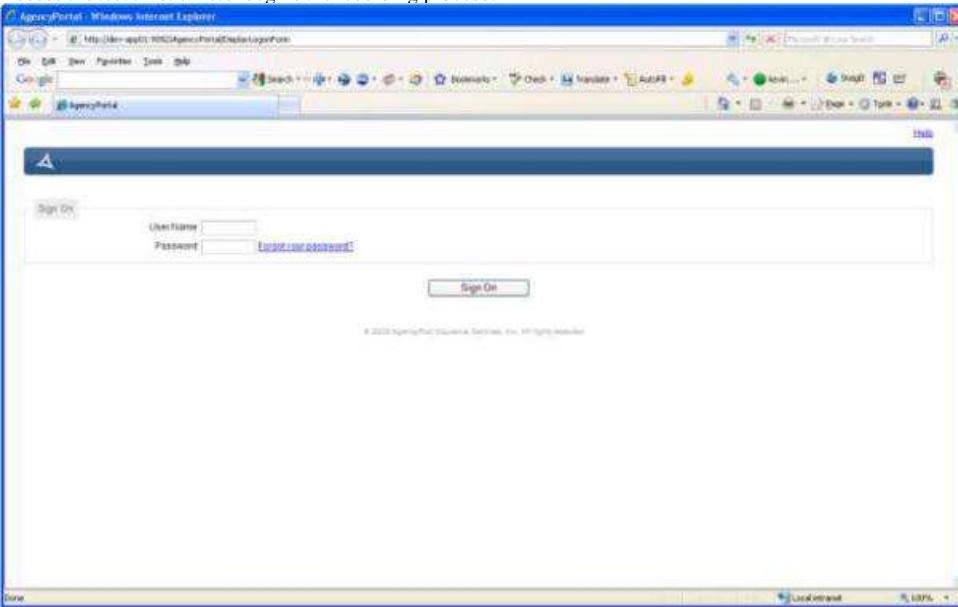
11. Click the LAN settings button. The Local Area Network (LAN) Settings window displays.



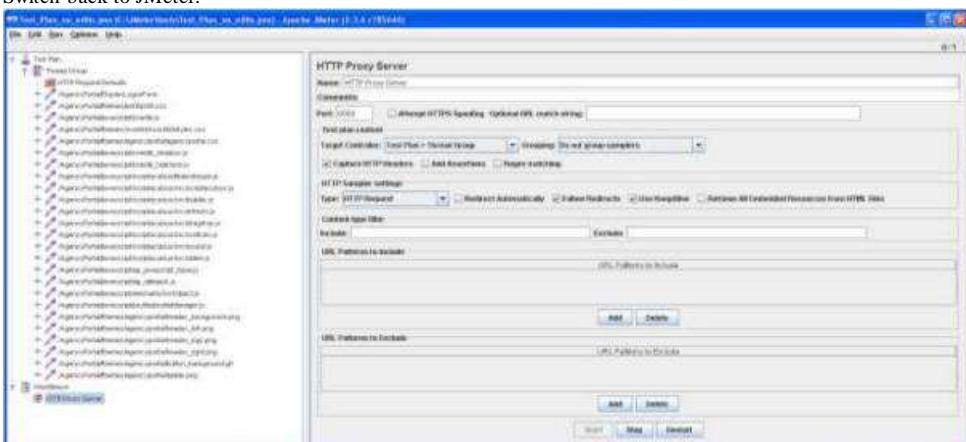
12. Create a proxy server for your environment.
  13. Keep Internet Explorer open.
  14. Switch back to JMeter and turn on the proxy server.



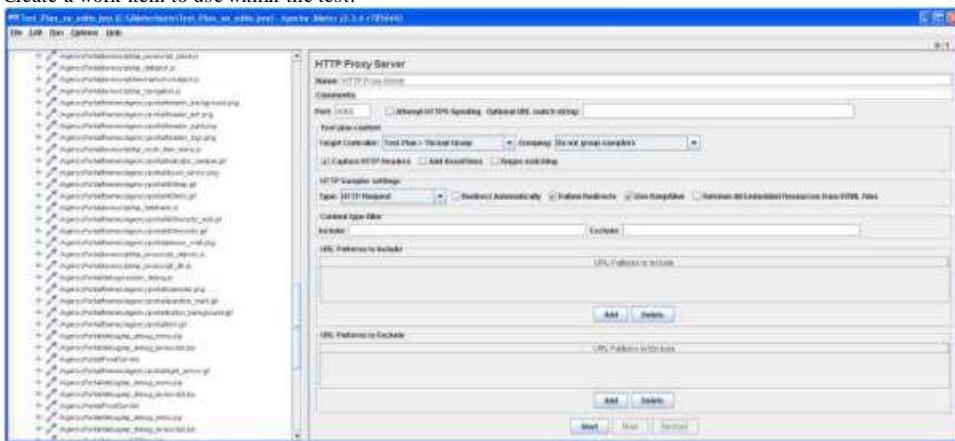
15. Access the test time. JMeter begins the recording process.



16. Switch back to JMeter.

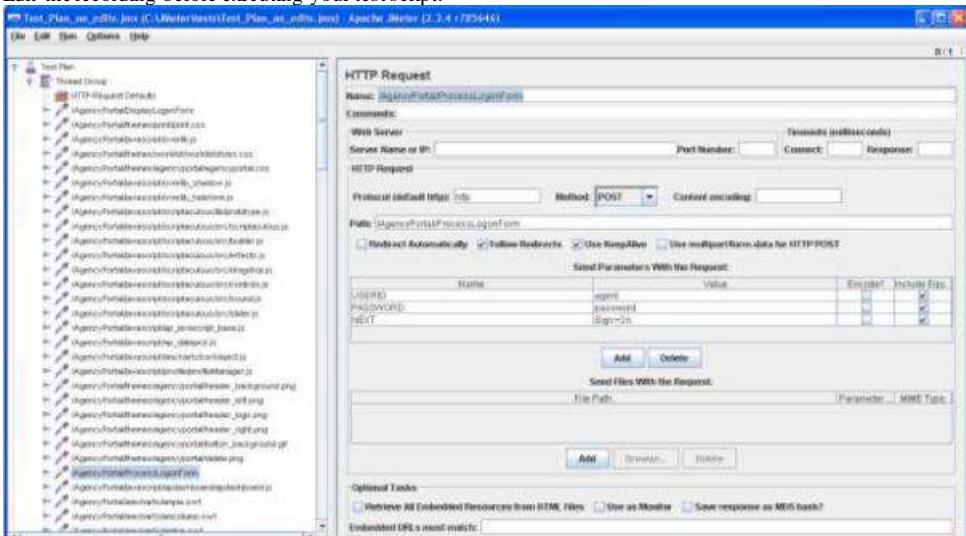


17. Create a work item to use within the test.



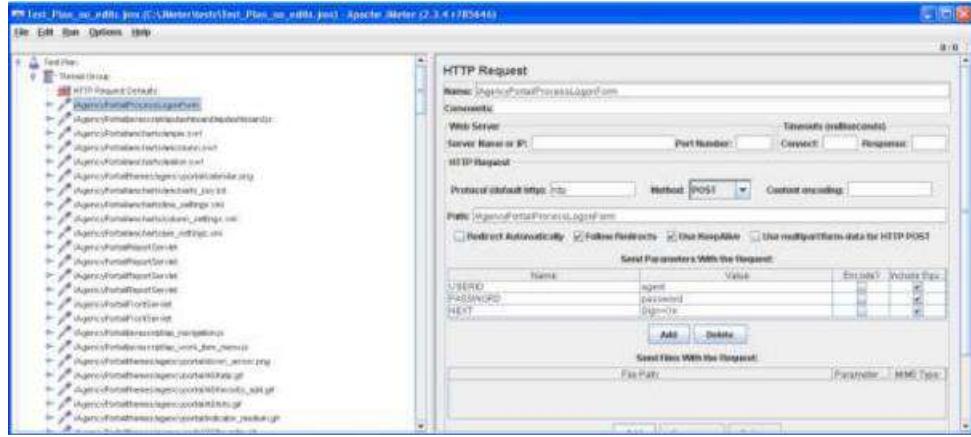
18. Click Stop. You can now disable the proxy server within Internet Explorer.

19. Edit the recording before executing your test script.

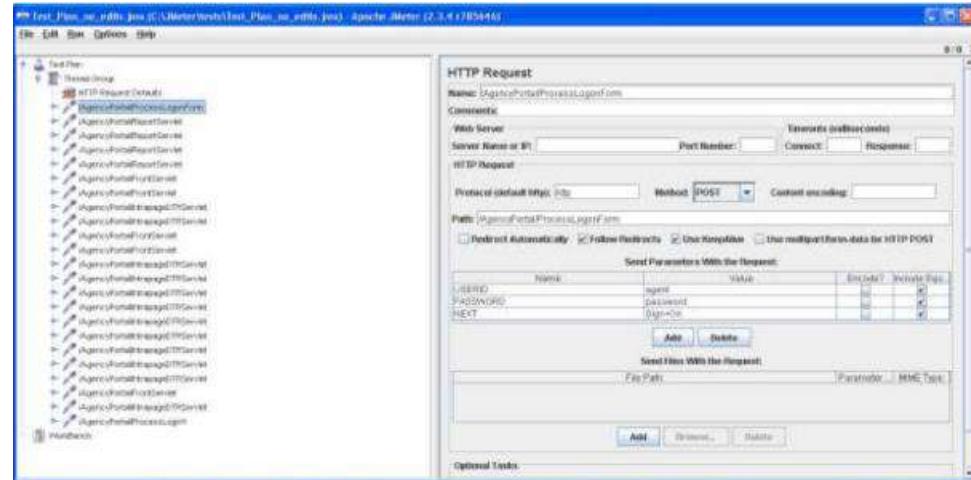


Look for a HTTP request ending with ProcessLogonForm (i.e., /AgencyPortal/ProcessLogonForm) and delete everything above that

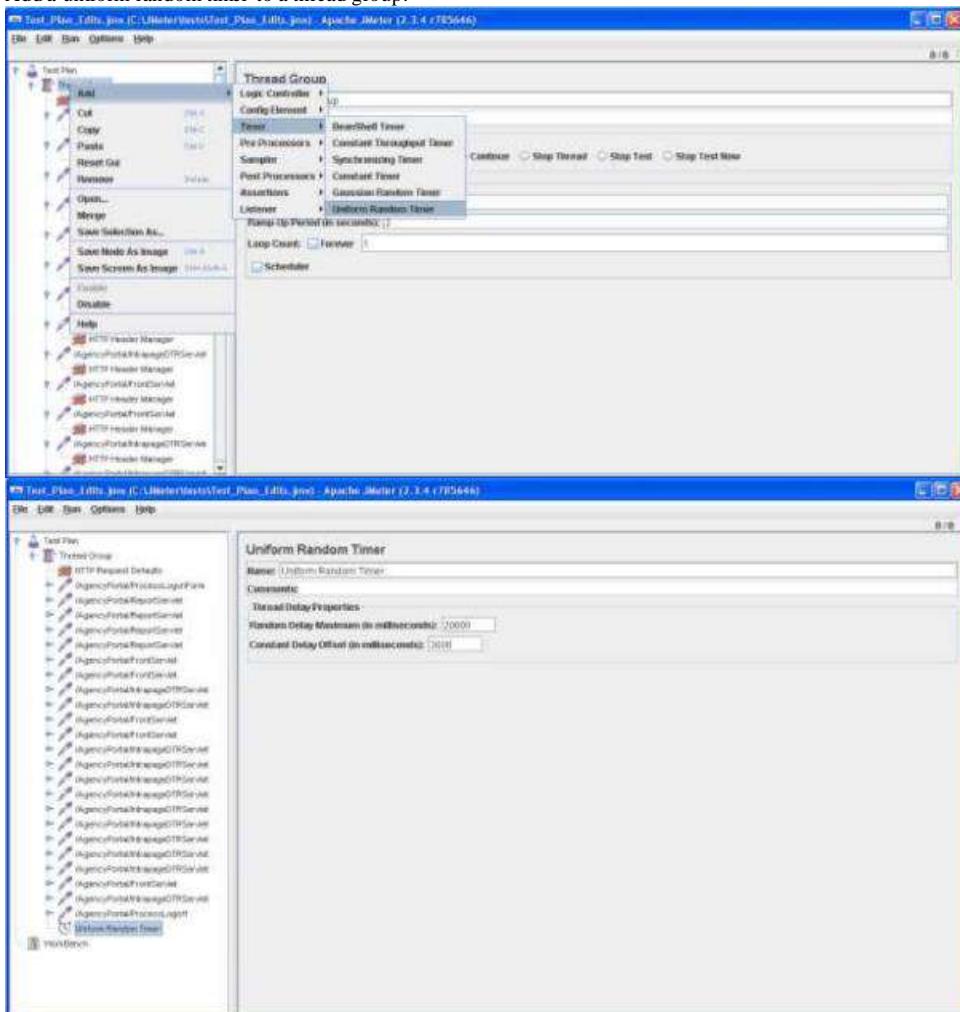
line.



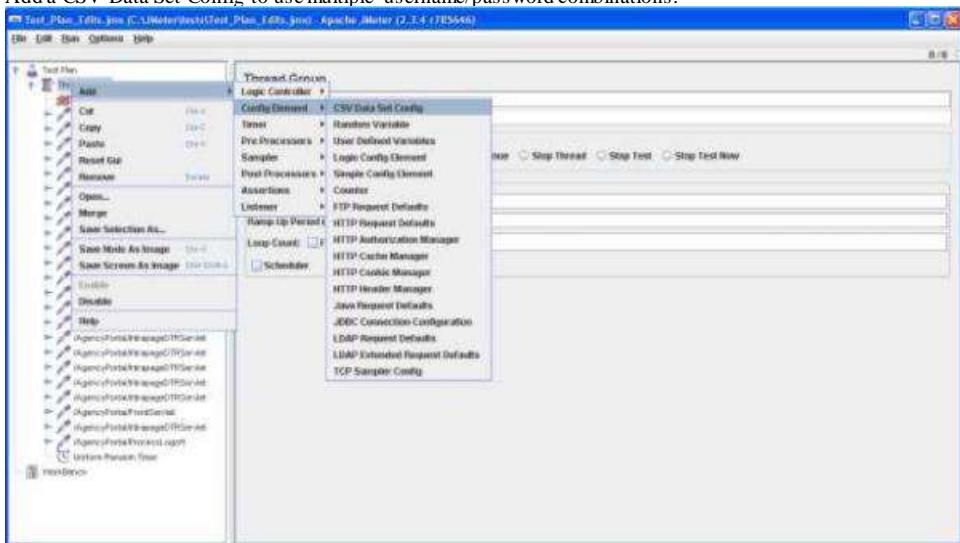
20. Remove all references to .css/.gif/.js/.swf/.txt/.xml files within the recorded file. Only servlet calls and the ProcessLogout call should remain.

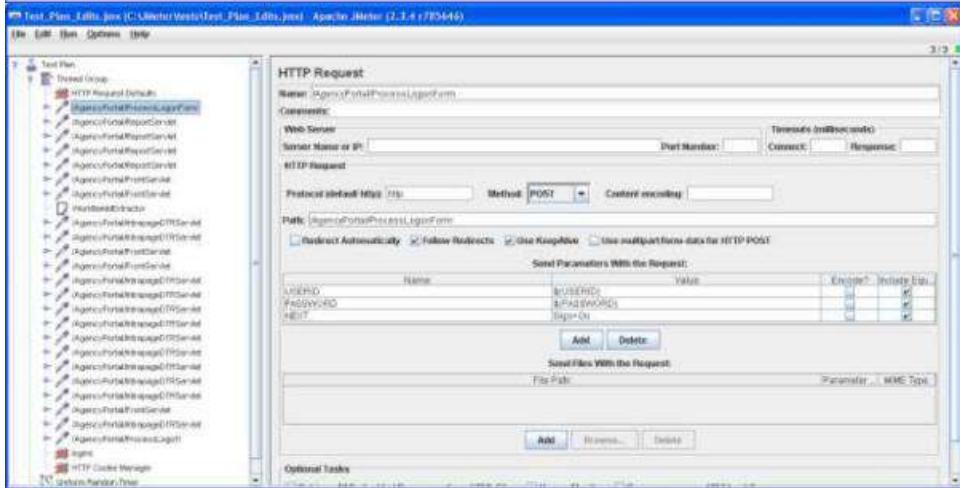
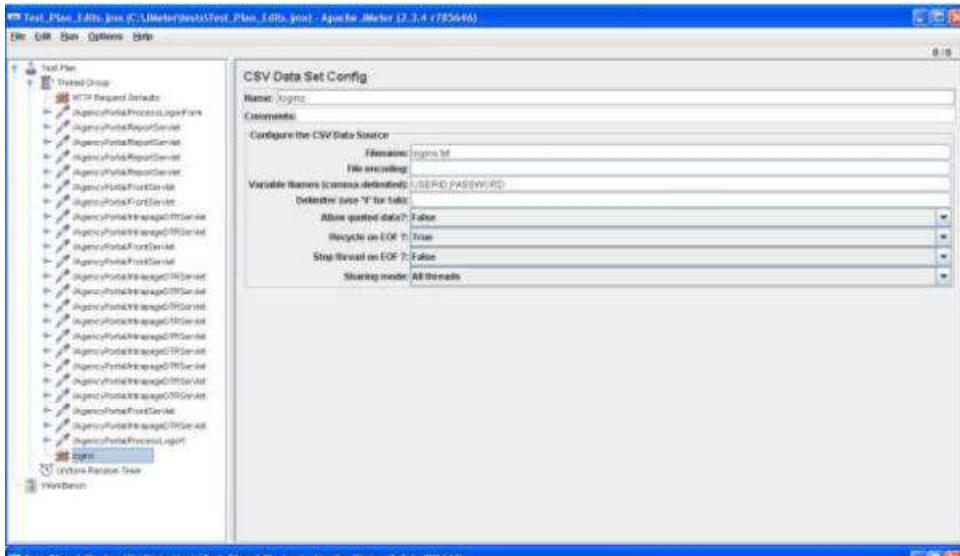


21. Add a uniform random timer to a thread group.

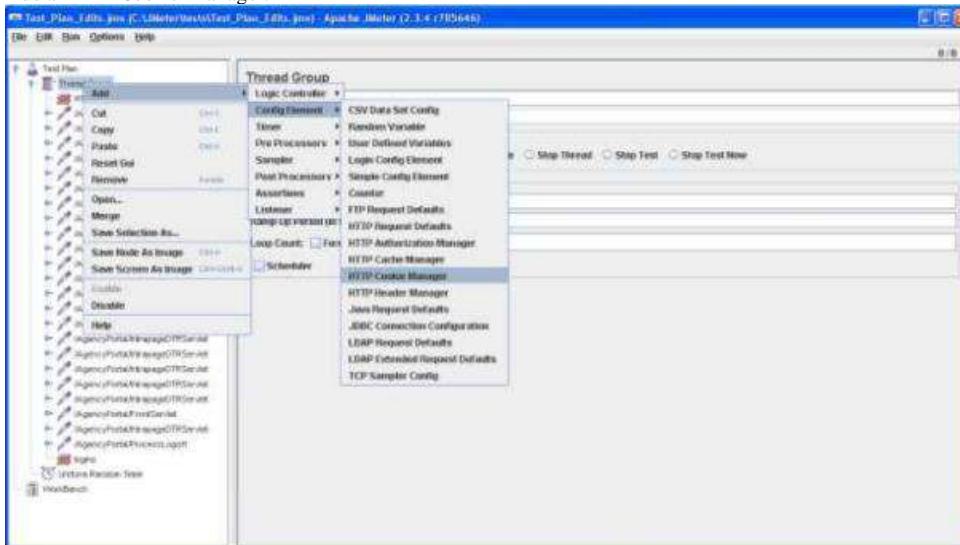


22. Add a CSV Data Set Config to use multiple username/password combinations.

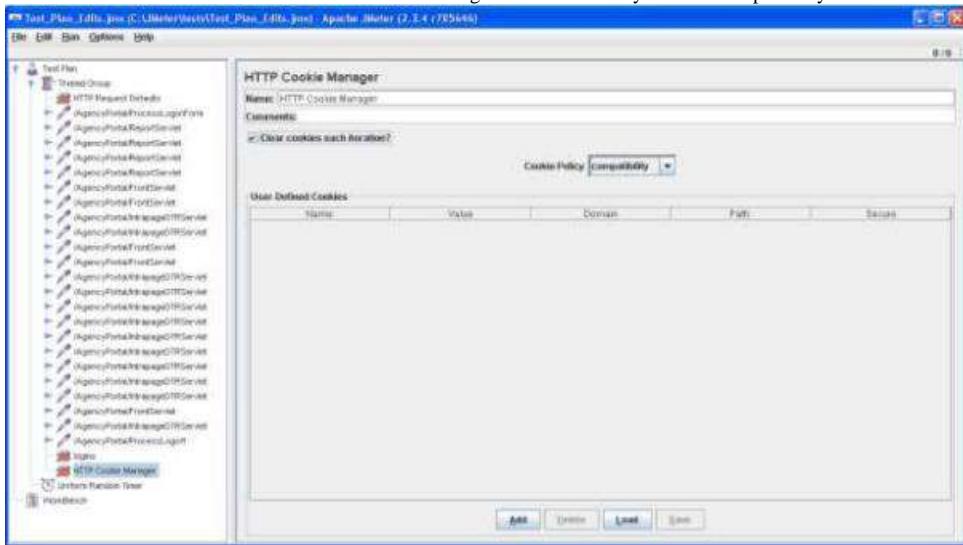




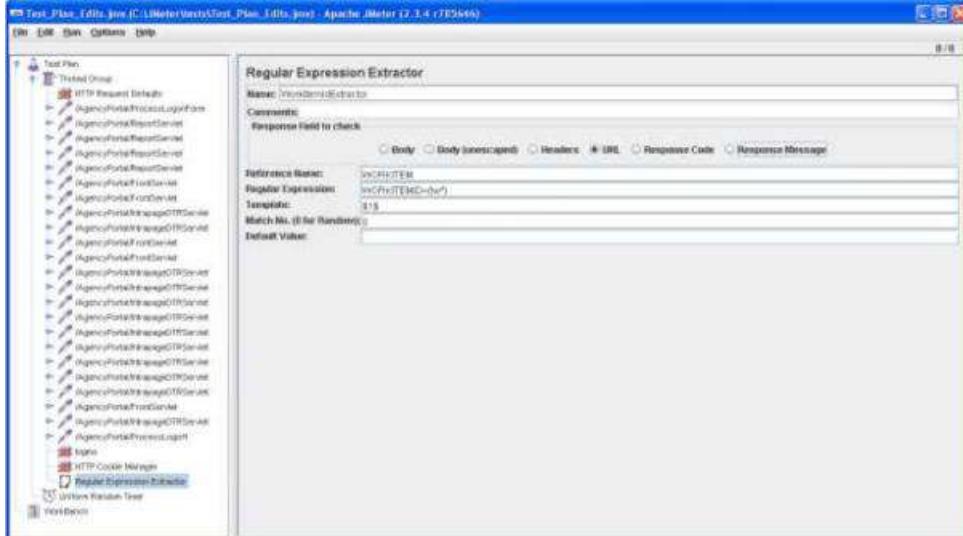
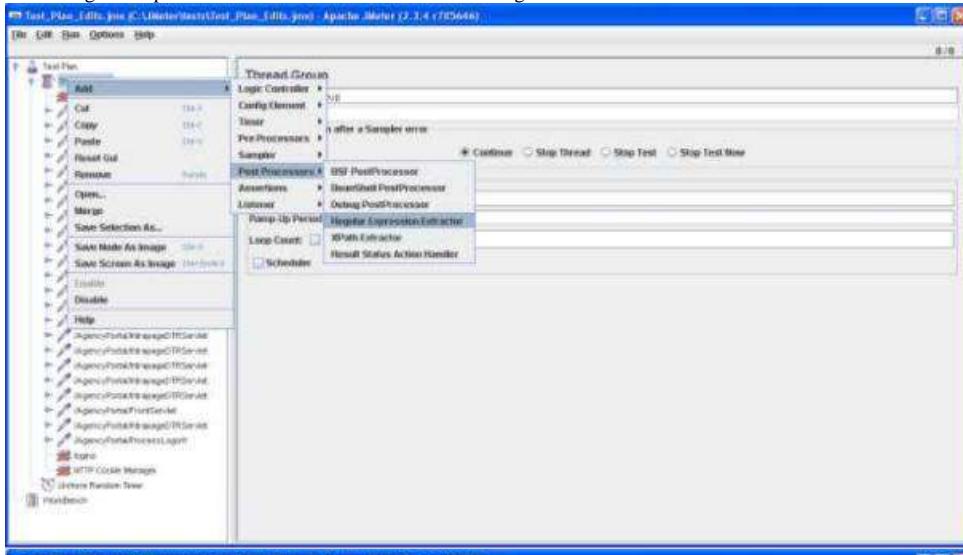
23. Save the logins.txt file in the same location as the .jmx file.
  24. Add a HTTP Cookie Manager.



25. Check the Clear cookies each iteration? field and change the Cookie Policy field to compatibility.



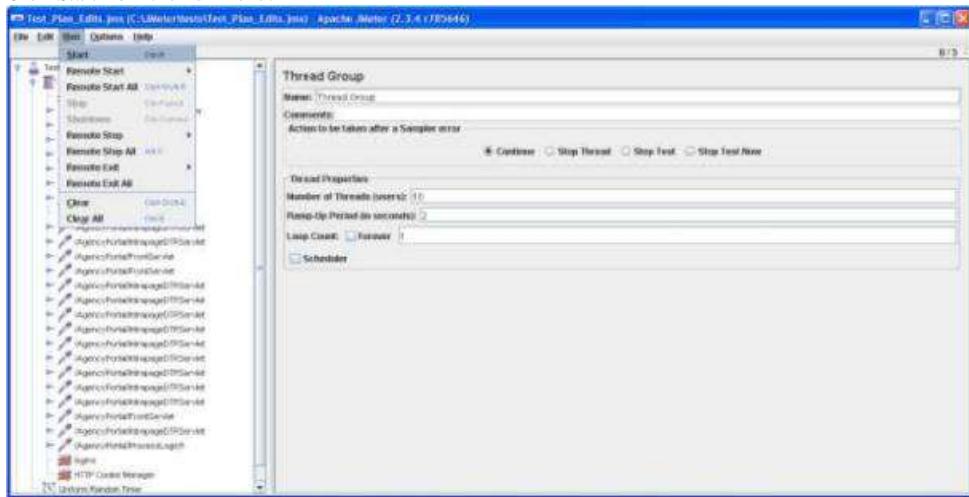
26. Add a Regular Expression Extractor to account for the change in work item number for iteration of the test.



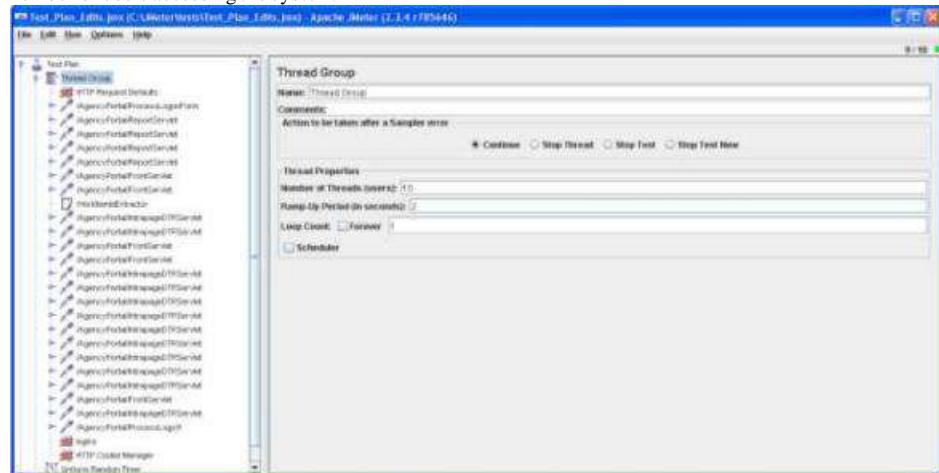
27. Verify that work item number referenced within the script is replaced with \${WORKITEM}.
28. Replace random numbers recorded in the script with a function call to create a random number. You can do this via the JMeter UI or by editing the .jmx file in a text editor:

```
<elementProp name="rnd" elementType="HTTPArgument"> <boolProp name="HTTPArgument.always_encode">false</boolProp>
<stringProp name="Argument.name">rnd</stringProp> <stringProp
name="Argument.value">${__Random(0,9999999)}</stringProp> <stringProp name="Argument.metadata">=</stringProp>
</elementProp>
```

29. Click Start from the Run menu.

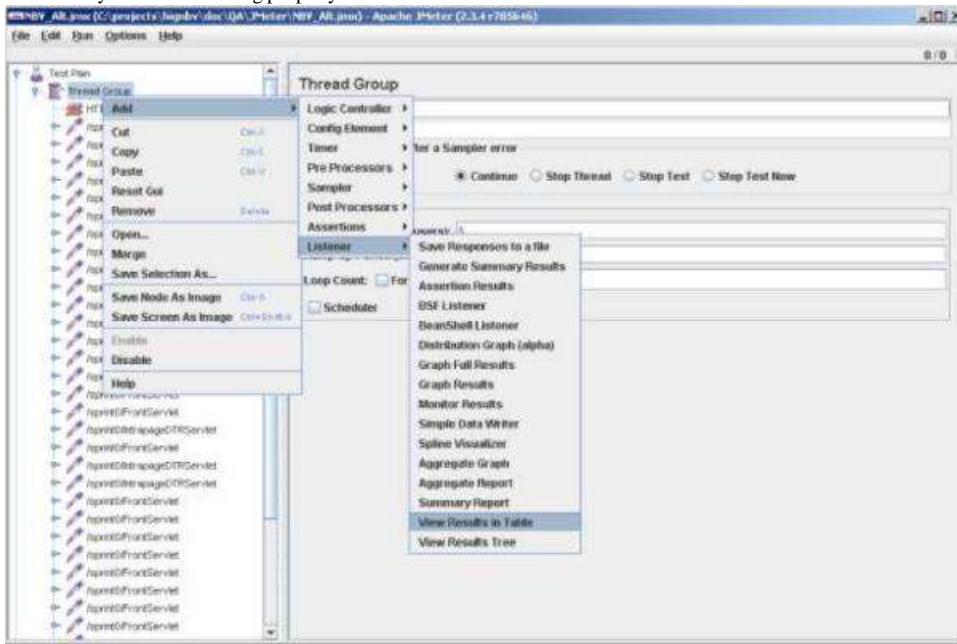


- When the test is running, a green light displays in the upper right corner of the UI. A number also displays to indicate the number of users accessing the system.

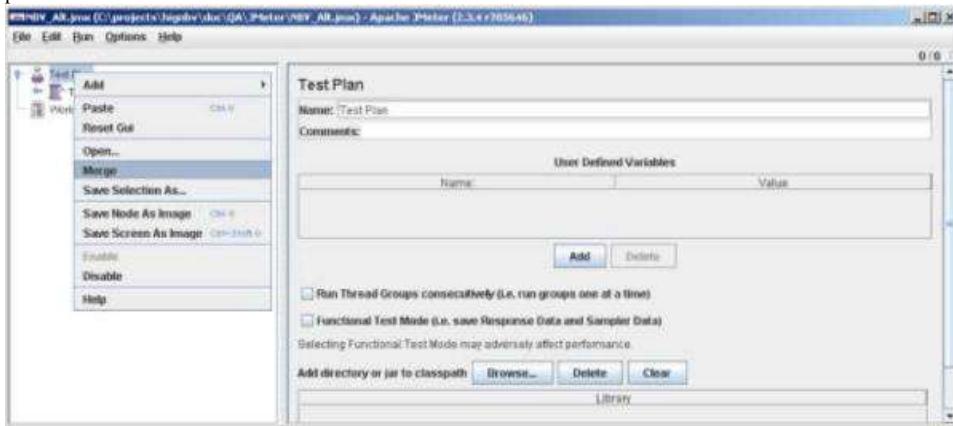


- When the test completes, the green light is no longer visible.

30. Add a View Results in Table listener as a debugging tool, which shows every call to and response from the server. This allows you to make sure your test is running properly.



31. Create each transaction and save it as a .jmx file to run multiple tests (i.e., two different transactions). Open the first and, with the merge command, open the second. You can configure each with a different number of threads and different delays to simulate a realistic usage pattern.



# Reports

Reporting establishes a standard framework for delivering summary chart reports to the AgencyPortal user community. The underlying architecture of AgencyPortal reports uses a graphics component called AM charts (<http://www.amcharts.com/>) to render reports.

## Report Basics

The basic reporting engine is driven by configurations stored in a set of reporting-oriented database tables. One of the basic design principals behind this feature is to offer a reporting engine that can be extended without the need for custom Java or JavaScript coding on the part of the application team. The nature of summary reporting raises the bar on the developer in the realm of SQL development due to the nature of the SQL result sets required by the AM charts.

There are two basic chart types supported by this infrastructure:

- Pie
- Line/Bar/Column

Each chart type has its own specific requirement for the composition of the SQL result sets and chart configurations. The result sets from SQL are converted to AM chart JSON data and combined with the chart configuration. The charts are then rendered by the reporting engine without the need for any customJava code.

### Pie Chart

Pie charts are driven by a result set where each result row is comprised of at least two columns: a slice title and a slice value. The slice title provides the name of the slice, while the slice value provides the raw data value for that pie slice.

Consider the following Total Submissions report SQL:

**SQL:**

```
SELECT w.LOB AS "Slice Title", COUNT(1) AS "Slice Value" FROM work_item w WHERE w.effective_date >= '01/01/2014' AND w.effective_date <= '12/31/2014' AND w.user_group_id IN (200) AND w.STATUS > 55 AND w.commit_flag > 0 GROUP BY w.LOB
```

The count of work items is aggregated by line of business for those records passing the where clause.

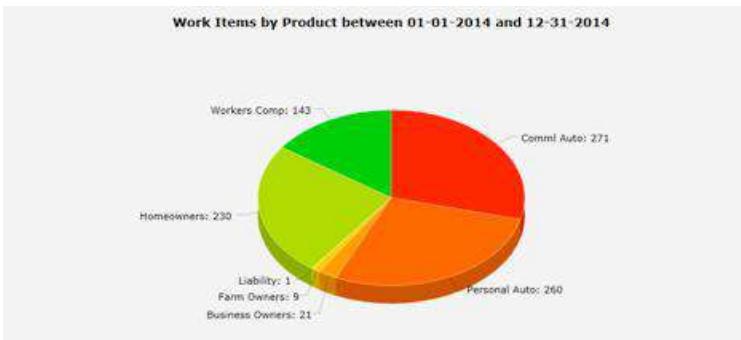
#### Chart Configuration:

```
<settings><type>pie</type><theme>none</theme> <valueField>value</valueField> <titleField>title</titleField>
<decimalSeparator>${rb.dashboard.format.DecimalSeparator}</decimalSeparator> <precision>-1</precision>
<balloonText><![CDATA[[[title]]
[[value]] ([[percents]]%)></balloonText>
<angle>30</angle> <depth3D>20</depth3D> <outlineAlpha>0.4</outlineAlpha> <chartCursor>
<categoryBalloonEnabled>false</categoryBalloonEnabled> <cursorAlpha>0</cursorAlpha> <zoomable>false</zoomable> </chartCursor>
<pathToImages>${my_portal_app}/themes/dashboard/</pathToImages> <amExport> <top>21</top> <right>21</right>
<buttonColor>#EFEFEF</buttonColor> <buttonRollOverColor>#DDDDDD</buttonRollOverColor> <exportPNG>true</exportPNG>
<exportJPG>true</exportJPG> <exportPDF>false</exportPDF> <exportSVG>true</exportSVG> <amExport> <legend>
<markerType>square</markerType> <position>right</position> <marginRight>80</marginRight> <autoMargins>false</autoMargins>
<borderAlpha>0.3</borderAlpha> <horizontalGap>10</horizontalGap> <valueText>[[none]]</valueText> </legend> <titles>
<text><![CDATA[${rb.dashboard.title.report100}]]></text> <size>14</size> <bold>true</bold> </titles> <labelRadius>20</labelRadius>
<labelsEnabled>true</labelsEnabled> <labelText>[[title]]: [[value]]</labelText> </settings>
```

The result set and the chart configuration are combined to generate the following JSON data:

```
{ "response": { "status": 0, "message": "Status[name=SUCCESS]", "results": { "chartData": { "LabelsEnabled": true, "precision": -1, "dataProvider": [{ "title": "Comm Auto", "value": 271 }, { "title": "Personal Auto", "value": 260 }, { "title": "Business Owners", "value": 21 }, { "title": "Farm Owners", "value": 9 }, { "title": "Liability", "value": 1 }, { "title": "Homeowners", "value": 230 }, { "title": "Workers Comp", "value": 143 }], "theme": "none", "type": "pie", "decimalSeparator": ".", "angle": 30, "chartCursor": { "zoomable": false, "cursorAlpha": 0, "categoryBalloonEnabled": false }, "labelText": "[[title]]: [[value]]", "amExport": { "exportPNG": true, "buttonRollOverColor": "#DDDDDD", "exportJPG": true, "exportSVG": true, "right": 21, "buttonColor": "#EFEFEF", "exportPDF": false, "top": 21 }, "balloonText": "[[title]]
[[value]] ([[percents]]%)</balloonText>, "legend": { "position": "right", "valueText": "[[none]]", "autoMargins": false, "marginRight": 80, "horizontalGap": 10, "borderAlpha": 0.3, "markerType": "square" }, "titleField": "title", "valueField": "value", "depth3D": 20, "outlineAlpha": 0.4, "titles": { "bold": true, "text": "Work Items by Product between 01-01-2014 and 12-31-2014", "size": 14 }, "pathToImages": "http://dev-app10:8137/agencyportal/themes/dashboard/", "labelRadius": 20 } } }
```

The JSON data produces a pie chart as follows:



## Line/Bar/Column Charts

Line/bar/column data is a little more complex because of the notion of a series. Each result set row is composed of three columns: a series value, a graph identifier (aka GIB) and a graph value. These reports involve plotting aggregated data values against a series that, many times, is a reporting date period.

### Example 1 - Work Items by Line of Business

#### SQL:

The following monthly submissions dashboard reporting SQL is an example of a SQL server specific line/bar/column data:

```
WITH date_range(start_date,end_date)AS (SELECT cast('01/01/2014' AS datetime) start_date,dateadd(dd,1,cast('12/31/2014' AS datetime)) end_date UNION ALL SELECT dateadd(mm,1,start_date), end_date FROM date_range WHERE dateadd(mm,1,start_date) < end_date)
SELECT substring(dataname(mm, date_range.start_date), 1, 3) + '-' + substring(cast(datepart(yy, date_range.start_date) AS varchar), 3, 2) AS "Series Value", LOB AS "Graph Id", count(LOB) AS "Graph Value" FROM date_range LEFT JOIN work_item w ON (date_range.start_date = dateadd(dd,-day(w.effective_date) + 1, w.effective_date) AND w.user_group_id IN (200) AND w.STATUS > 55 AND w.commit_flag > 0)
GROUP BY date_range.start_date,LOB ORDER BY date_range.start_date,LOB
```

The count of work items is aggregated by line of business by month within a particular date range.

One of the challenges with this report is the "by month" part of the requirement and how to create the series portion of the result set, even if there are no data points for that series point. Furthermore, the technique for achieving this differs by database vendor type. The above result set is mapped to AM chart XML by the report engine without the need for any customJava code.

#### Chart Configuration:

```
<settings><type>serial</type> <theme>none</theme> <decimalSeparator>${rb:dashboard:format.DecimalSeparator}</decimalSeparator>
<precision>1</precision> <angle>30</angle> <depth3D>20</depth3D>
<pathToImages>${my_portal_app}/themes/dashboard/</pathToImages> <amExport> <top>21</top> <right>21</right>
<buttonColor>#EFEFEF</buttonColor> <buttonRollOverColor>#DDDDDD</buttonRollOverColor> <exportPNG>true</exportPNG>
<exportJPG>true</exportJPG> <exportPDF>false</exportPDF> <exportSVG>true</exportSVG> <buttonTitle>Save chart as an image</buttonTitle> </amExport> <categoryField>value</categoryField> <categoryAxis> <gridPosition>start</gridPosition>
<axisAlpha>1</axisAlpha> <gridAlpha>1</gridAlpha> <position>left</position> <minorGridEnabled>true</minorGridEnabled>
<labelRotation>90</labelRotation> <axisThickness>2</axisThickness> <title><![CDATA[${rb:dashboard:title.report101}]]></title>
<categoryAxis> <valueAxes> <stackType>regular</stackType> <axisAlpha>1</axisAlpha> <gridAlpha>1</gridAlpha>
<axisThickness>2</axisThickness> </valueAxes> <legend> <useGraphSettings>true</useGraphSettings> <markerType>square</markerType>
<position>right</position> <marginRight>80</marginRight> <autoMargins>false</autoMargins> <borderAlpha>0.3</borderAlpha>
<horizontalGap>10</horizontalGap> </legend> <graphSetting>
<balloonText><![CDATA[${rb:dashboard:balloonText.report101}]]></balloonText> <fillAlphas>0.8</fillAlphas>
<labelText>[[value]]</labelText> <lineAlpha>0.3</lineAlpha> <type>column</type> <color>#000000</color> </graphSetting> </settings></pre>

```

The above result set is converted to AM chart JSON data and combined with the chart configuration by the reporting engine without the need for any customJava code:

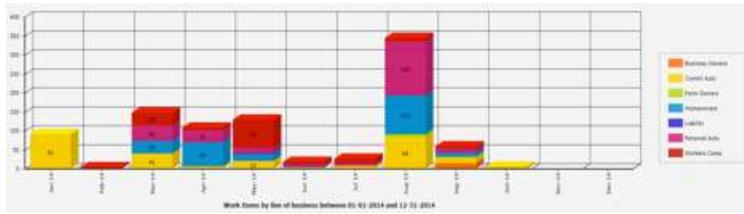
```
{ "response": { "status":0, "message": "Status[name=SUCCESS]", "results": { "chartData": { "precision": -1, "valueAxes": [{ "gridAlpha": 1, "axisThickness": 2, "axisAlpha": 1, "stackType": "regular" }], "dataProvider": [{ "value": "Jan-14", "AUTOB": "91" }, { "value": "Feb-14", "WORK": "4" }, { "HOME": "33", "AUTOP": "41" }, { "value": "Mar-14", "WORK": "33", "AUTOB": "41" }, { "HOME": "64", "AUTOP": "32", "value": "Apr-14"}, { "WORK": "6", "AUTOB": "7" }, { "HOME": "19", "AUTOP": "16", "CGL": "1", "value": "May-14", "WORK": "72", "BOP": "2", "AUTOB": "19" }, { "HOME": "1", "AUTOP": "8", "value": "Jun-14", "WORK": "5", "AUTOB": "4" }, { "AUTOP": "7", "value": "Jul-14", "WORK": "11", "AUTOB": "9" }, { "HOME": "103", "AUTOP": "143", "value": "Aug-14", "WORK": "8", "CFRM": "7", "AUTOB": "84" }, { "HOME": "10", "AUTOP": "13", "value": "Sep-14", "WORK": "4", "BOP": "16", "CFRM": "2", "AUTOB": "14" }, { "value": "Oct-14", "BOP": "3", "AUTOB": "2" }, { "value": "Nov-14" }, { "value": "Dec-14" }], "theme": "none", "guides": { "guide": { } }, "type": "serial", "axis": { "x": { "label": "Month", "values": ["Jan-14", "Feb-14", "Mar-14", "Apr-14", "May-14", "Jun-14", "Jul-14", "Aug-14", "Sep-14", "Oct-14", "Nov-14", "Dec-14"] } } } }}
```

```

"decimalSeparator": ".", "angle": 30, "categoryField": "value", "amExport": { "buttonTitle": "Save chart as an image", "exportPNG": true, "buttonRollOverColor": "#DDDDDD", "exportJPG": true, "exportSVG": true, "right": 21, "buttonColor": "#EFEFEF", "exportPDF": false, "top": 21 }, "legend": { "position": "right", "autoMargins": false, "marginRight": 80, "useGraphSettings": true, "horizontalGap": 10, "borderAlpha": 0.3, "markerType": "square" }, "depth3D": 20, "categoryAxis": { "position": "left", "gridAlpha": 1, "title": "Work Items by line of business between 01-01-2014 and 12-31-2014", "axisThickness": 2, "axisAlpha": 1, "gridPosition": "start", "minorGridEnabled": true, "labelRotation": 90 }, "graphs": [{ "lineAlpha": 0.3, "title": "Business Owners", "labelText": "[[value]]", "color": "#000000", "balloonText": "[[value]] work items ([[percents]]%) are [[title]] for [[category]]", "valueField": "BOP", "type": "column", "fillAlphas": 0.8 }, { "lineAlpha": 0.3, "title": "Comml Auto", "labelText": "[[value]]", "color": "#000000", "balloonText": "[[value]] work items ([[percents]]%) are [[title]] for [[category]]", "valueField": "AUTOB", "type": "column", "fillAlphas": 0.8 }, { "lineAlpha": 0.3, "title": "Farm Owners", "labelText": "[[value]]", "color": "#000000", "balloonText": "[[value]] work items ([[percents]]%) are [[title]] for [[category]]", "valueField": "CFRM", "type": "column", "fillAlphas": 0.8 }, { "lineAlpha": 0.3, "title": "Homeowners", "labelText": "[[value]]", "color": "#000000", "balloonText": "[[value]] work items ([[percents]]%) are [[title]] for [[category]]", "valueField": "HOME", "type": "column", "fillAlphas": 0.8 }, { "lineAlpha": 0.3, "title": "Liability", "labelText": "[[value]]", "color": "#000000", "balloonText": "[[value]] work items ([[percents]]%) are [[title]] for [[category]]", "valueField": "CGL", "type": "column", "fillAlphas": 0.8 }, { "lineAlpha": 0.3, "title": "Personal Auto", "labelText": "[[value]]", "color": "#000000", "balloonText": "[[value]] work items ([[percents]]%) are [[title]] for [[category]]", "valueField": "AUTOP", "type": "column", "fillAlphas": 0.8 }, { "lineAlpha": 0.3, "title": "Workers Comp", "labelText": "[[value]]", "color": "#000000", "balloonText": "[[value]] work items ([[percents]]%) are [[title]] for [[category]]", "valueField": "WORK", "type": "column", "fillAlphas": 0.8 }], "pathToImages": "http://dev-app10:8137/agencyportal/themes/dashboard/" } } }

```

The following is then rendered by the AM chart:



## Example 2 - Premium by Month

**SQL:**

```

with date_range (start_date,end_date) as (select cast(?) as datetime) start_date, cast(?) as datetime) end_date union all select
dateadd(mm,1,start_date - day(start_date)+1),end_date from date_range where dateadd(mm,1,start_date - day(start_date)+1) < end_date) select
substring(dataname(mm, date_range.start_date), 1, 3) + '-' + substring(cast(datepart(yy,date_range.start_date)as varchar), 3, 2) series,LOB as
gid,sum(premium) as value from date_range left join work_item w on (date_range.start_date <= w.effective_date and date_range.end_date >=
w.effective_date and dateadd(mm,1,date_range.start_date - day(date_range.start_date)+1)-1 >= w.effective_date and w.user_group_id in (200)
and w.status <> 55 and w.commit_flag <> 0) group by date_range.start_date,LOB order by date_range.start_date,LOB

```

**Chart Configuration:**

```

<settings><type>serial</type> <theme>none</theme> <decimalSeparator>${rb.dashboard.format.DecimalSeparator}</decimalSeparator>
<pathToImages>${my_portal_app}/themes/dashboard/</pathToImages> <amExport> <top>21</top> <right>21</right>
<buttonColor>#EFEFEF</buttonColor> <buttonRollOverColor>#DDDDDD</buttonRollOverColor> <exportPNG>true</exportPNG>
<exportJPG>true</exportJPG> <exportPDF>false</exportPDF> <exportSVG>true</exportSVG> </amExport>
<categoryField>value</categoryField> <chartCursor> <fullWidth>true</fullWidth> <cursorAlpha>0.1</cursorAlpha> </chartCursor>
<categoryAxis> <minorGridEnabled>true</minorGridEnabled> <gridAlpha>1</gridAlpha> <axisAlpha>1</axisAlpha>
<title><![CDATA[${rb.dashboard.title.report102}]]></title> </categoryAxis> <valueAxes> <gridAlpha>1</gridAlpha>
<axisAlpha>1</axisAlpha> <position>left</position> </valueAxes> <legend> <useGraphSettings>true</useGraphSettings>
<position>right</position> <marginRight>80</marginRight> <autoMargins>false</autoMargins> <borderAlpha>0.3</borderAlpha>
<horizontalGap>10</horizontalGap> <valueText>[[none]]</valueText> </legend> <graphSetting> <lineColor>#FF6600</lineColor>
<bullet>round</bullet> <bubbleBorderThickness>1</bubbleBorderThickness> <fillAlphas>0</fillAlphas> <balloonText>[[title]]:
[[value]]</balloonText> </graphSetting> <chartScrollbar> <selectedBackgroundColor>#99CCFF</selectedBackgroundColor>
<scrollbarHeight>30</scrollbarHeight> <color>#FFFFFF</color> <autoGridCount>true</autoGridCount> </chartScrollbar>
<mouseWheelZoomEnabled>true</mouseWheelZoomEnabled> </settings>

```

The above result set is converted to AM chart JSON data and combined with the chart configuration by the reporting engine without the need for any customJava code:

```

{ "response": { "status": 0, "message": "Status[name=SUCCESS]", "results": { "chartData": { "valueAxes": [{ "position": "left", "gridAlpha": 1, "axisAlpha": 1 }], "dataProvider": [{ "value": "Jan-14", "AUTOB": "3000.00" }, { "value": "Feb-14", "WORK": "1000.00" }, { "value": "HOME": "8250.00", "AUTOP": "10250.00" }, { "value": "Mar-14", "WORK": "7750.00", "AUTOB": "10250.00" }, { "value": "HOME": "15500.00", "AUTOP": "8000.00" }, { "value": "Apr-14", "WORK": "1500.00", "AUTOB": "1750.00" }, { "value": "HOME": "4750.00", "AUTOP": "4000.00", "CGL": "250.00" }, { "value": "May-14", "WORK": "3750.00", "BOP": "250.00", "AUTOB": "4750.00" }, { "value": "HOME": "250.00", "AUTOP": "2000.00", "value": "Jun-14", "WORK": "1000.00", "AUTOB": "750.00" }, { "AUTOP": "1750.00", "value": "Jul-14", "WORK": "2750.00", "AUTOB": "2250.00" }, { "HOME": "3500.00", "AUTOP": "7250.00", "value": "Aug-14", "WORK": "750.00", "CFRM": "250.00", "AUTOB": "2250.00" }] } }

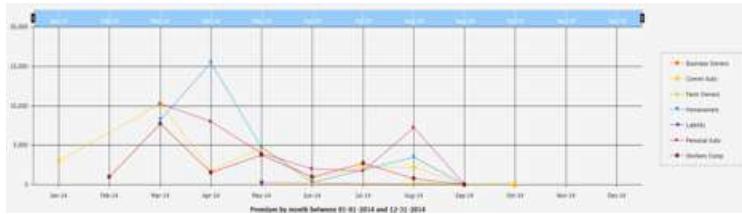
```

```

"HOME": "0.00", "AUTOP": "0.00", "value": "Sep-14", "WORK": "0.00", "BOP": "0.00", "CFRM": "0.00", "AUTOB": "0.00" }, { "value": "Oct-14", "BOP": "0.00", "AUTOB": "250.00" }, { "value": "Nov-14" }, { "value": "Dec-14" }], "theme": "none", "guides": { "guide": {} }, "type": "serial", "decimalSeparator": ".", "categoryField": "value", "chartCursor": { "cursorsorAlpha": 0.1, "fullWidth": true }, "chartScrollbar": { "color": "#FFFFFF", "scrollbarHeight": 30, "selectedBackgroundColor": "#99CCFF", "autoGridCount": true }, "amExport": { "exportPNG": true, "buttonRollOverColor": "#DDDDDD", "exportJPG": true, "exportSVG": true, "right": 21, "buttonColor": "#EFEFEF", "exportPDF": false, "top": 21 }, "legend": { "position": "right", "valueText": "[none]", "autoMargins": false, "marginRight": 80, "useGraphSettings": true, "horizontalGap": 10, "borderAlpha": 0.3 }, "mouseWheelZoomEnabled": true, "categoryAxis": { "gridAlpha": 1, "title": "Premium by month between 01-01-2014 and 12-31-2014", "axisAlpha": 1, "minorGridEnabled": true }, "graphs": [{ "title": "Business Owners", "lineColor": "#FF6600", "balloonText": "[[title]]: [[value]]", "valueField": "BOP", "bulletBorderThickness": 1, "fillAlphas": 0, "bullet": "round" }, { "title": "Comm Auto", "lineColor": "#FC2D20", "balloonText": "[[title]]: [[value]]", "valueField": "AUTOB", "bulletBorderThickness": 1, "fillAlphas": 0, "bullet": "square" }, { "title": "Farm Owners", "lineColor": "#B0DE09", "balloonText": "[[title]]: [[value]]", "valueField": "CFRM", "bulletBorderThickness": 1, "fillAlphas": 0, "bullet": "triangleUp" }, { "title": "Homeowners", "lineColor": "#0D8ECF", "balloonText": "[[title]]: [[value]]", "valueField": "HOME", "bulletBorderThickness": 1, "fillAlphas": 0, "bullet": "triangleDown" }, { "title": "Liability", "lineColor": "#2A0CD0", "balloonText": "[[title]]: [[value]]", "valueField": "CGL", "bulletBorderThickness": 1, "fillAlphas": 0, "bullet": "triangleLeft" }, { "title": "Personal Auto", "lineColor": "#CD0D74", "balloonText": "[[title]]: [[value]]", "valueField": "AUTOP", "bulletBorderThickness": 1, "fillAlphas": 0, "bullet": "triangleRight" }, { "title": "Workers Comp", "lineColor": "#CC0000", "balloonText": "[[title]]: [[value]]", "valueField": "WORK", "bulletBorderThickness": 1, "fillAlphas": 0, "bullet": "bubble" }], "pathToImages": "http://dev-app:8137/agencyportal/themes/dashboard/" } } } }

```

The following is rendered by AM chart:



### Example 3 - Cumulative Premium by Month

SQL:

```

with date_range(start_date,end_date) as (select cast(? as datetime) start_date,cast(? as datetime) end_date union all select dateadd(mm,1,start_date - day(start_date)+1),end_date from date_range where dateadd(mm,1,start_date - day(start_date)+1)< end_date) select substring(datename(mm, a.start_date), 1, 3) + '-' + substring(cast(datepart(yy,a.start_date) as varchar), 3, 2) series , a.gid, sum(b.value) as value from (select date_range.start_date,w.LOB as gid,sum(w.premium) as value from date_range left join work_item w on (date_range.start_date <= w.effective_date and date_range.end_date >= w.effective_date and dateadd(mm,1,date_range.start_date - day(date_range.start_date)+1)-1 >= w.effective_date and w.user_group_id in (200) and w.status <> 55 and w.commit_flag <> 0) group by date_range.start_date,w.LOB) a left join (select date_range.start_date,w.LOB as gid,sum(w.premium) as value from date_range left join work_item w on (date_range.start_date <= w.effective_date and date_range.end_date >= w.effective_date and dateadd(mm,1,date_range.start_date - day(date_range.start_date)+1)-1 >= w.effective_date and w.user_group_id in (200) and w.status <> 55 and w.commit_flag <> 0) group by date_range.start_date,w.LOB) b on (a.start_date >= b.start_date and a.gid = b.gid) group by a.start_date,a.gid order by a.start_date,a.gid

```

#### Chart Configuration:

```

<settings><type>serial</type> <theme>none</theme> <decimalSeparator>${rb.dashboard.format.DecimalSeparator}</decimalSeparator>
<pathToImages>$[my_portal_app]/themes/dashboard/</pathToImages><amExport> <top>21</top> <right>21</right>
<buttonColor>#EFEFEF</buttonColor> <buttonRollOverColor>#DDDDDD</buttonRollOverColor> <exportPNG>true</exportPNG>
<exportJPG>true</exportJPG> <exportPDF>false</exportPDF> <exportSVG>true</exportSVG> </amExport> <chartCursor>
<cursorPosition>mouse</cursorPosition> <chartCursor> <categoryField>value</categoryField> <categoryAxis> <gridAlpha>1</gridAlpha>
<axisAlpha>1</axisAlpha> <labelRotation>45</labelRotation> <title><![CDATA[$[rb.dashboard.title.report106]]]></title> </categoryAxis>
<valueAxes> <gridAlpha>1</gridAlpha> <axisAlpha>1</axisAlpha> <position>left</position> </valueAxes> <legend>
<useGraphSettings>true</useGraphSettings> <position>right</position> <marginRight>80</marginRight> <autoMargins>false</autoMargins>
<borderAlpha>0.3</borderAlpha> <horizontalGap>10</horizontalGap> <valueText>[[value]]</valueText> </legend> <graphSetting>
<lineColor>#FF6600</lineColor> <bullet>round</bullet> <bulletBorderThickness>1</bulletBorderThickness> <fillAlphas>0</fillAlphas>
<balloonText>[[title]]: [[value]]</balloonText> </graphSetting> </settings>

```

The above result set is converted to AM chart JSON data and combined with the chart configuration by the reporting engine without the need for any customJava code:

```

{ "response": { "status": 0, "message": "Status[name=SUCCESS]", "results": { "chartData": { "valueAxes": [{ "position": "left", "gridAlpha": 1, "axisAlpha": 1 }], "dataProvider": [{ "value": "Jan-14", "AUTOB": "3000.00" }, { "value": "Feb-14", "WORK": "1000.00" }, { "HOME": "8250.00", "AUTOP": "10250.00" }, { "value": "Mar-14", "WORK": "8750.00", "AUTOB": "13250.00" }, { "HOME": "23750.00", "AUTOP": "18250.00" }, { "value": "Apr-14", "WORK": "10250.00", "AUTOB": "15000.00" }, { "HOME": "28500.00", "AUTOP": "22250.00" }, { "CGL": "250.00", "value": "May-14", "WORK": "14000.00", "BOP": "250.00", "AUTOB": "19750.00" }, { "HOME": "28750.00", "AUTOP": "24250.00" }, { "value": "Jun-14", "WORK": "15000.00", "AUTOB": "20500.00" }, { "AUTOP": "26000.00", "value": "Jul-14", "WORK": "1000.00" }] } } }

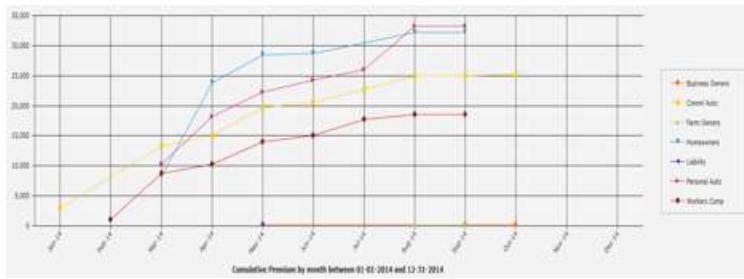
```

```

 "17750.00", "AUTOB": "22750.00" }, { "HOME": "32250.00", "AUTOP": "33250.00", "value": "Aug-14", "WORK": "18500.00", "CFRM": "250.00", "AUTOB": "25000.00" }, { "HOME": "32250.00", "AUTOP": "33250.00", "value": "Sep-14", "WORK": "18500.00", "BOP": "250.00", "CFRM": "250.00", "AUTOB": "25000.00" }, { "value": "Oct-14", "BOP": "250.00", "AUTOB": "25250.00" }, { "value": "Nov-14" }, { "value": "Dec-14" }], "theme": "none", "guides": { "guide": { } }, "type": "serial", "decimalSeparator": ".", "categoryField": "value", "chartCursor": { "cursorPosition": "mouse" }, "amExport": { "exportPNG": true, "buttonRollOverColor": "#DDDDDD", "exportJPG": true, "exportSVG": true, "right": 21, "buttonColor": "#EFEFEF", "exportPDF": false, "top": 21 }, "legend": { "position": "right", "valueText": "[[none]]", "autoMargins": false, "marginRight": 80, "useGraphSettings": true, "horizontalGap": 10, "borderAlpha": 0.3 }, "categoryAxis": { "gridAlpha": 1, "title": "Cumulative Premium by month between 01-01-2014 and 12-31-2014", "axisAlpha": 1, "labelRotation": 45 }, "pathToImages": "http://dev-app108137/agencyportal/themes/dashboard/", "graphs": [{ "title": "Business Owners", "lineColor": "#FF6600", "balloonText": "[[title]]: [[value]]", "valueField": "BOP", "bulletBorderThickness": 1, "fillAlphas": 0, "bullet": "round" }, { "title": "Comml Auto", "lineColor": "#FCD202", "balloonText": "[[title]]", "valueField": "AUTOB", "bulletBorderThickness": 1, "fillAlphas": 0, "bullet": "square" }, { "title": "Farm Owners", "lineColor": "#B0DE09", "balloonText": "[[title]]: [[value]]", "valueField": "CFRM", "bulletBorderThickness": 1, "fillAlphas": 0, "bullet": "triangleUp" }, { "title": "Homeowners", "lineColor": "#0D8ECF", "balloonText": "[[title]]: [[value]]", "valueField": "HOME", "bulletBorderThickness": 1, "fillAlphas": 0, "bullet": "triangleDown" }, { "title": "Liability", "lineColor": "#2A0CDO", "balloonText": "[[title]]: [[value]]", "valueField": "CGL", "bulletBorderThickness": 1, "fillAlphas": 0, "bullet": "triangleLeft" }, { "title": "Personal Auto", "lineColor": "#CDOD74", "balloonText": "[[title]]: [[value]]", "valueField": "AUTOP", "bulletBorderThickness": 1, "fillAlphas": 0, "bullet": "triangleRight" }, { "title": "Workers Comp", "lineColor": "#CC0000", "balloonText": "[[title]]: [[value]]", "valueField": "WORK", "bulletBorderThickness": 1, "fillAlphas": 0, "bullet": "bubble" }] } } }

```

The following is then rendered by AM chart:



# Report Configuration

The configuration supporting the AgencyPortal reports feature is contained in a set of relational tables. For each report, there is one record in the `report` and `report_sql` tables with one or more supporting records in the `report_params` and `report_results` tables. The latter two tables are linked to the report table by the `report_id` foreign key. The `report_sql` table is linked to the report table by the `sql_id` foreign key.

This table is what holds onto the actual SQL that is applied against the database when the report is run. The reason the `report_sql` table is modeled separately from the report table is that there are some situations where the same SQL can be used for different reports by applying parameter substitutions dynamically at runtime.

## Configuring a Report

This section details the configuration of a report that ships, by default, with the AgencyPortal application. The configuration of the total submissions report is used throughout this section as a reference.

### Report Table

The report table contains the basic properties that describe the report (as a reference, the total submissions report):

Column	Value	Description
<code>report_id</code>	100	This is the primary key to this report. Its value is not important; it just must be unique by report.
<code>is_drill_down</code>	0	This signifies whether this is a drill down report. Use a value of 0 (zero) when the report is not a drill down and 1 when it is a drill down report. The total submissions report is not a drill down report.
<code>sql_id</code>	100	This is the foreign key to the record on the <code>report_sql</code> table that supports this report.
<code>default_chart_id</code>	1	The default charting type for chart based reports if the application does not specify. This is <code>NULL</code> for list based reports.
<code>drill_down_report_id</code>	104	Specifies the <code>report_id</code> of the drill down report. This is <code>NULL</code> if the report does not have a drill down report.
<code>report_name</code>	Total Submissions	Contains the report name.
<code>report_description</code>	Submissions by LOB	Contains the report description.
<code>permission</code>	<code>canViewTotalSubmissions</code>	Contains the name of the permission necessary to contain in a user's security profile for them to have authority to run the report. <code>NULL</code> implies that all users can exercise a report. For this report to engage, the security profile for a given user must contain a permission with the name of <code>canViewTotalSubmissions</code> .

### Report SQL Table

This table contains the raw SQL that is prepared and normalized before running the report.

### Note

Although you can write the SQL for this particular report in a database independent fashion, it is important to note that many reports require SQL that is not portable across database vendors. This is especially true when there is aggregation across date reporting periods.

The total submissions report example is designed as a pie chart based report that reports on the number of submissions by line of business within a particular date range. A submission in this report is defined as any work item with a status that is something other than one of the deleted

(status code of 55) and with the commit flag is set to on. The SQL must filter work items for the user group(s) in which the current user has authority.

Column	Date	Description
sql_id	100	Primary key
report_sql	SELECT w.LOB, COUNT(1) FROM \${db_table_prefix}work_item w where w.effective_date >= ? and w.effective_date <= ? and w.user_group_id in (\${user_groups}) and w.status > 55 and w.commit_flag > 0 GROUP BY w.LOB	The two return columns in the result set conform to the pie chart requirements of slice title and slice value.

## Tip

Make sure all of your table references in the SQL contain table prefixes.

### Report Parameters Table

You must account for each parameter in the SQL with one record in the report\_params table for any report. For example:

report_id	param_num	param_name	param_type	param_value
100	1	start_date	date	NULL
100	2	end_date	date	NULL
100	-1	user_groups	user_groups	NULL

You can configure parameters in the SQL either as a question mark (?) or substitutable variable (\${var}). Parameters configured with a question mark must have a param\_num value that aligns with its relative order found in the SQL statement. There are two bound parameters in the SQL for this report, which define the effective date range of the result set. Parameters expressed as substitutable variables have a -1 in the param\_num column. In those cases, the param\_name value must match the substitutable variable found in the SQL. The only substitutable variable in this report is the user\_groups variable.

You can pass parameter values as part of the inbound HTTP request. These override any default values configured or calculated by the reporting engine. Those parameter names match the param\_name values for a report. Values passed on the request have precedence over the default\_value value set on this record. When there is no request parameter passed, the default\_value value (if not NULL) is applied to the SQL when prepared for execution. There are two special system-calculated defaults for the param\_name values of start\_date and end\_date engaged. They are 01/01/<current year> 12:00AM and 12/31/<current year> 11:59:59PM.

The following table lists the supported valid param\_type values supported by the dashboard reporting engine:

Parameter Type Value	Description	System default value	Display value format
date	Denotes a parameter value represented as a date. Bound parameters are mapped as SQL date values.	Special rules above regarding start_date and end_date	MM-DD-YYYY
string	Denotes a parameter value represented as a string. Bound	N/A	As is

Parameter Type Value	Description	System default value	Display value format
	parameters are mapped as SQL string values.		
numeric	Denotes a parameter value represented as a whole number (integer). Bound parameters are mapped as SQL int values.	N/A	As is
decimal	Denotes a parameter value represented as a decimal number (double). Bound parameters are mapped as SQL double values.	N/A	As is
user_groups	Denotes a parameter value as an in list of user group IDs. This is not supported as a bound parameter, but only as a substitutable variable.	All of the user groups that the current user participates in directly (or indirectly), as returned by <code>ISecurityProfile.getUserGroups()</code> .	The name of the user (work) group as in <code>IUserGroup.getName()</code>
LOB	Denotes a line of business code value.	Defaults to the value of ALL, which signifies all lines of business, as in <code>TransactionDefinitionManager.getAllLOBs()</code>	If LOB value is ALL, then "all products," else <code>LOBCode.getDescription()</code>

### Report Results Table

This table applies to chart based reports only. For each column in the result set, there is a corresponding row in the `report_results` table. Pie chart based reports have two rows per report while line/bar/column based reports have three rows.

report_id	result_num	result_type	is_series	is_gid	is_value	default_value
100	1	LOB	NULL	1	0	NULL
100	2	numeric	NULL	0	1	NULL

The `result_num` must correspond to the `column` order of the result set entity. Pie charts always have a `NULL` value in the `is_series` value. The slice title column has a 1 in the `is_gid` column and the slice value has a 1 in the `is_value` column. This may seem a little odd since the table describes a result set destined for a line/bar/column chart rather than a pie chart; nevertheless, the concept is not too difficult to get used to. Gid stands for "graph identifier."

The parameter types in the previous section are all accepted as valid result type values; however, there is one more result type value that is supported on the results.

<b>Result Type Value</b>	<b>Description</b>	<b>Display value format</b>
status	Denotes a work item status value.	The status's "after" title [as in <code>IWorkItemStatus.getAfterChangeStatusTitle()</code> ]

The `default_value` column is currently not used; set it to `NULL`.

### Chart Table

The chart table contains the properties needed to support the creation and initialization of the AM charting object.

<b>Column</b>	<b>Value</b>	<b>Description</b>
chart_id	1	Primary key
chart_type	pie	Supported types are: pie, line, column

Reports are written in a way to support a particular category of charting. Pie charts are one category with slice titles and slice values while line and column charts require a series, gid and graph value. You can probably "rechart" a report designed to use a line chart to a bar chart without changing the report itself. That is as far as you can go before you start redesigning the SQL query and how the particular result sets are mapped. Generally speaking, you do not need to create chart records unless you need to, for some reason, change the chart settings stored in the settings file.

# User Workflow

This section provides detailed examples and workflows to help non-technical analysts learn more about how AgencyPortal works and/or more information about a specific feature within the application. The screen shots and workflow diagrams included in the following sections describe how AgencyPortal looks and functions out of the box.

- [Getting Started](#) - The Getting Started topic lists the different sections of the home page and describes how it looks out of the box.
- [Work List](#) - The Work List section includes information on the features and functionality available from the My Work page.
- [Search and Filter](#) - The Search and Filter topic details the unified search and filter functionality provided throughout the application.
- [Accounts](#) - The Accounts section includes information and workflow topics on the features and functionality available from the My Accounts page, including how to add and manage accounts within the application.
- [Work Items](#) - The Work Items section details the components that make up a work item, such as transaction and summary pages and the Work Item Assistant and Timeline features. This section also includes workflow topics on how to add and manage work items within the application.
- [Reports](#) - The Reports topic describes the default reports that display on the My Reports page within the application.
- [Upload](#) - The Upload topic provides details on the two different types of upload features available out of the box.

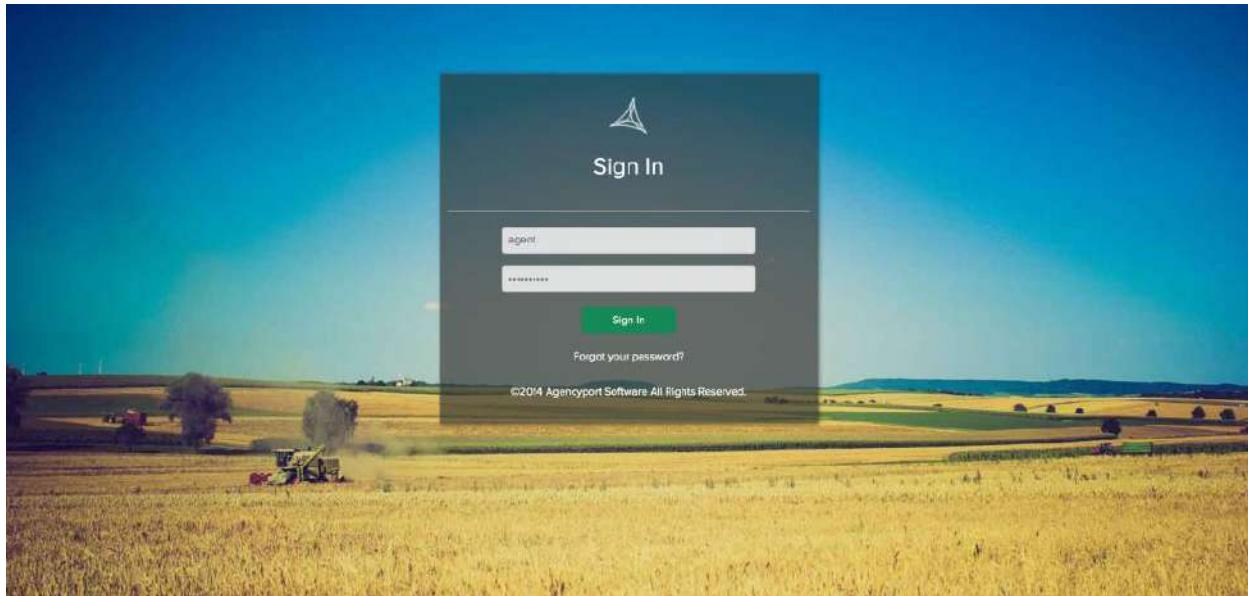
For information on setting up the backend components of these features, refer to the [Developer Guide](#) section.

## Note

The material in the Developer Guide section is geared towards software developers. Refer to your development team for information on setting up any particular feature.

# Getting Started

After you are set up with login credentials for a carrier's AgencyPortal application, you are ready to log into the application.



The login page includes designated fields to enter your username and password. Enter your login credentials and click the Sign In button to log into the application. If you've entered any incorrect credentials, the following message displays: *Please make sure that your user name and password were entered correctly.* Click the Forgot your password? link if you've forgotten your password.

## Home Page

After you have logged into the application, the home page displays. The AgencyPortal home page is the forefront component of the application and provides quick access to all of the functions you need to create and manage quotes, policies and accounts.

This page is broken out into the following features and functionality:

### Navigation Menu

The navigation menu displays at the top of the home page and every other page within the application. It provides a quick and easy way to access the components of the application.

This menu includes the following options:

- Home - This option directs you back to the home page.
- My Accounts - This option displays the My Accounts page to work with accounts.
- My Work - This option displays the My Work page to work with work items.
- My Reports - This option displays the My Reports page to work with reports.
- Get Started - This option displays the Get Started drop-down, which displays the options to create a work item from an ACORD form or create a quick quote or full application for a line of business.
- User menu - This option displays who is logged into the application and provides the option to log out.

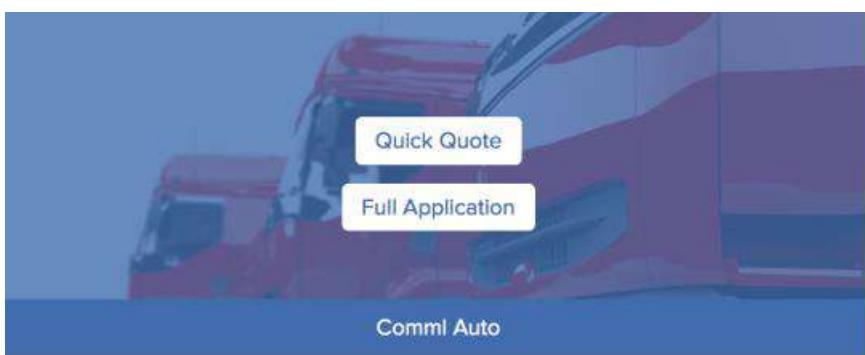
#### Get Started

The get started section includes thumbnails for each available LOB. These thumbnails display the available LOBs for the carrier, as well as the option to create a quick quote or full application directly from the home page. The LOB thumbnails that display are based on the carrier's project implementation specifications.

#### Get Started



After you click a thumbnail, two options display to create a quick quote or full application.



- Create a quick quote - Selecting the Quick Quote option displays a subset of the full application, which contains key information required to rate and provide a quote to the customer.
- Create a full application - Selecting the Full Application option displays a more substantial application, which includes all information required to submit the policy for issuance.

#### Queue

The queue section lists work items in your userqueue (based on user group) with the most recent activity and those waiting for underwriter decision.

## Queue

### Most Recent

Name	Status	Type	Duration
Darla Crane	In-Progress	WORK	1 days
John Doe	In-Progress	WORK	4 days
Charles Smith	In-Progress	AUTOB	4 days
John Doe	In-Progress	WORK	4 days
Charles Smith	In-Progress	AUTOB	4 days

### Waiting for Underwriter

Name	Status	Type	Duration
test test	Referred	AUTOP	5 days
Betty Boop	Referred	AUTOP	20 days
Catrlone Morierty	Referred	HOME	186 days
Donal Murphy	Referred	HOME	186 days
Damien Tusk	Referred	HOME	186 days

- Most Recent - This table lists the five most recently updated work items in your queue that are in any status except referred (referred indicates it is waiting on underwriter review).
- Waiting for Underwriter - This table lists the five most recently updated work items in a referred status, which are pending underwriter review.

The following information displays for each listed item:

- Name - The name of the insured.
- Status - The status of the work item.
- Type - The transaction line of business.
- Effective - The transaction effective date.
- Duration - The duration, in days, since the transaction was last updated.

Click on the name associated with the item to navigate directly to the work item. The work item page displays to continue working with the work item.

Refer to the [Work Item Queues](#) topic in the Developer Guide section for information on how to configure these queues.

## Quotes

The quotes section includes a statistical breakdown of your userquote activity and the total number of work items in a particular status.



The following statuses display:

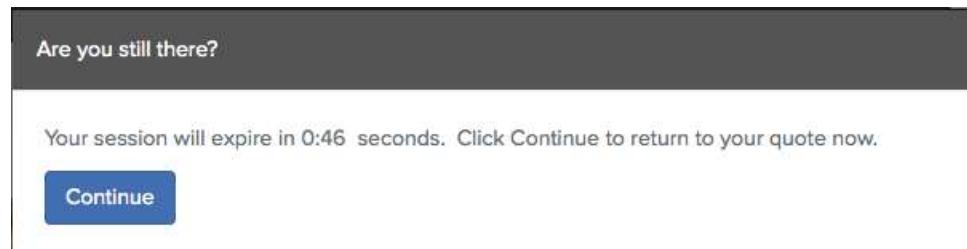
- In Progress
- Approve
- Bind
- Decline

The colors associated with each status box correlates to the same colors that display for work items on the card view version of the My Work page (e.g., the card for a declined work item displays in red). The number included in the status box corresponds to the number of work items in that particular status. Click on a status box to display all work items in that status. Refer to the [Work Item Quotes](#) topic in the Developer Guide section for information on how to customize this section.

## Session Timeout

The AgencyPortal application has a built-in feature that helps safeguard sensitive customer data in the event you walk away from your desk and/or leave your session unattended.

When your session becomes inactive for a configurable amount of time, an alert dialog will display to confirm whether you want to continue working within the application. A countdown displays to let you know how long you have to confirm before you are logged out of the system. By default, this dialog displays for 60 seconds.



If you are not available or choose not to click Continue, you will be automatically logged out of the system and the login page will redisplay for you to re-enter your username and password.

Refer to the Session Timeout section in the Miscellaneous Properties topic for information on how to configure this feature.

# Work List

The work list, or My Work page, provides access to all the work items you have been authorized to view. From this page, you can create new work items. In addition, the page contains all the functions necessary to manage your work items (delete, copy, endorse, etc). By default, work items display on the page sorted by the last updated date. The option to search, filter and sort work items is also available to display particular results (refer to the [Search and Filter](#) topic for more information).

## Page View

There are two ways you can view the My Work page: card view or classic view. The view is toggled by clicking on the  and  icons in the top upper right hand side of the page. When you log out or navigate away from the work list, the system remembers the last selected work list view.

The number of items that display when the page is loaded is based on the user's authentication settings, any entered search terms or selected filters, and any parameters set during project implementation. At the bottom of each page, a pagination bar displays to navigate through multiple pages of work items.

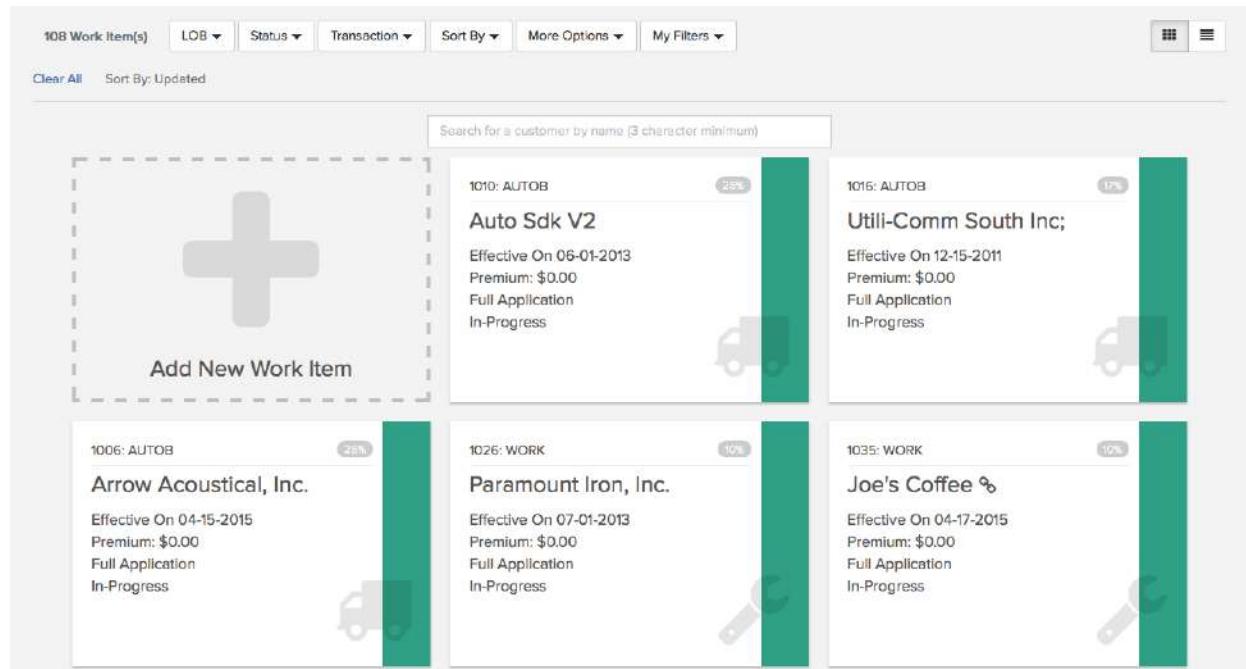


The number of work items that displays per page is determined during project implementation. The pagination bar only displays if there are more than the configured number of work items per page. For example, if there are only 12 work items to display and the page is configured to display up to 20 work items, the pagination bar will not display. Refer to the [Work List Configuration and Customization](#) topic in the Developer Guide section for more information.

The functionality on both page views works the same, but the work item information displays differently. The project team can set the default work list view for the application via configuration. Refer to the [Work List](#) topic in the Developer Guide section for more information.

## Card View

 Click the  button to toggle to the card view version of the page. Viewing work items in the card view displays work item data on a card.



108 Work Item(s) LOB ▾ Status ▾ Transaction ▾ Sort By ▾ More Options ▾ My Filters ▾

Clear All Sort By: Updated

Search for a customer by name (3 character minimum)

Work Item ID	Description	Status	Progress (%)	Icon
1010: AUTOB	Auto Sdk V2	Effective On 06-01-2013 Premium: \$0.00 Full Application In-Progress	25%	
1015: AUTOB	Utili-Comm South Inc;	Effective On 12-15-2011 Premium: \$0.00 Full Application In-Progress	17%	
1006: AUTOB	Arrow Acoustical, Inc.	Effective On 04-15-2015 Premium: \$0.00 Full Application In-Progress	25%	
1026: WORK	Paramount Iron, Inc.	Effective On 07-01-2013 Premium: \$0.00 Full Application In-Progress	10%	
1035: WORK	Joe's Coffee %	Effective On 04-17-2015 Premium: \$0.00 Full Application In-Progress	10%	

The card format shows information pertaining to the work item in one place. The following information displays, by default, for each work item on the card view of the My Work page:

ID	The unique number assigned to the work item when it is created.
LOB	Line of business ACORD code. Each LOB is assigned an icon that displays on the card (e.g., the work item in the above image is for a commercial auto LOB. Since the truck icon is assigned to the commercial auto LOB, a truck icon displays on the card).
Complete	Percent complete for the work item.
Name	<p>Name of applicant.</p> <p>Note</p> <p>A link icon ( ) displays next to the work item if it is not associated with an account.</p>
Effective Date	Effective date of the policy.
Premium Amount	Quoted premium amount for the policy.
Transaction Type	Type of transaction (full application, quick quote, endorsement, etc.).
Status	Status of the work item (in progress, referred, declined, bound, etc). The status determines the color that displays on the card (e.g., the work item in the above image is in an In-Progress status, so the card displays in green since that is the color assigned to that status). The colors assigned to statuses are determined during project implementation.
Updated	Date and time when the work item was last updated.

Classic View

 Click the  button to toggle to the classic view version of the page. Viewing work items in the classic view displays the work item data in a traditional list that is sortable by columns.

108 Work Item(s)		LOB	Status	Transaction	Sort By	More Options	My Filters		
<a href="#">Clear All</a> Sort By: Updated									
Search for a customer by name (3 character minimum)									
Get Started With ▾									
ID	LOB	Name	Status	Updated	Effective	Premium	Transaction Type	Complete %	
1010	Commercial Auto	Auto Sdk V2	In-Progress	2015-04-16 12:38:44	06-01-2013	0	Full Application	25	
1016	Commercial Auto	Utili-Comm South Inc.	In-Progress	2015-04-16 14:21:41	12-15-2011	0	Full Application	17	
1006	Commercial Auto	Arrow Acoustical, Inc.	In-Progress	2015-04-16 15:08:59	04-15-2015	0	Full Application	25	
1026	Workers Comp	Paramount Iron, Inc.	In-Progress	2015-04-16 15:48:55	07-01-2013	0	Full Application	10	
1035	Workers Comp	Joe's Coffee %	In-Progress	2015-04-17 10:34:00	04-17-2015	0	Full Application	10	
1030	Commercial Auto	Heron Pest Control, Inc. %	In-Progress	2015-04-17 10:42:14	10-01-2012	0	Full Application	25	
1037	Commercial Auto	LoJac Holdings Corporation, Inc.	In-Progress	2015-04-17 10:46:08	09-30-2015	0	Full Application	25	
1041	Workers Comp	XeriscapeS Unlimited, Inc. %	In-Progress	2015-04-17 11:09:42	04-01-2013	0	Full Application	10	

These column headers organize the data on the page and can be easily sorted by clicking the header name. The columns that display, by default, on the classic view of the My Work page are as follows:

ID	Unique number assigned to the work item when it is created.
LOB	Line of business ACORD code.
Name	<p>Name of the applicant.</p> <p>Note</p>  A link icon displays next to the work item if it is not associated with an account.
Effective Date	Effective date of the policy.
Premium Amount	Quoted premium amount for the policy.
Transaction Type	Type of transaction (full application, quick quote, endorsement, etc).
Status	Status of the work item (in progress, referred, declined, bound, etc).
Updated	Date and time when the work item was last updated.
Complete	Percent complete for the work item.

Columns can be adjusted and/or customized during project implementation.

## Work Item Actions

After you [create a work item](#), you can make modifications and perform other actions. The main action you can perform after a work item is created is open the work item for edit, which displays as an Open button when you select the work item. The same work item actions display on both the card and classic view versions of the My Work page.

1184: AUTOP

**John Doe %**

Effective On 04-26-2015

Premium: \$0.00

Quick Quote

In-Progress

Open Other Actions▼

1185	Workers Comp	John Doe	In-Progress	04-27-2015 10:36:01	04-27-2015	0	Quick Quote	83
1194	Commercial Auto	Express Label Company	In-Progress	04-27-2015 10:31:09	05-11-2012	0	Full	Open Other Actions▼

Additional work item actions display when you click the Other Actions drop-down.

Work item actions can include the following, depending on the transaction type and user permissions. Project teams can implement other actions as required by the business requirements.

<b>Open</b>	Click the Open button to view and/or edit a work item.
<b>Delete</b>	Click the Delete option to delete a work item. This option displays on the Other Actions drop-down button. Refer to <a href="#">Managing Work Items</a> for more information.
<b>Copy</b>	Click the Copy option to copy a work item in its entirety to create a new work item with a unique work item number. This option displays on the Other Actions drop-down button. Refer to <a href="#">Managing Work Items</a> for more information.
<b>Link</b>	Click the Link option to assign the work item to an account. This option displays on the Other Actions drop-down button and only when a work item is not already assigned to an account. Refer to <a href="#">Managing Work Items</a> for more information.
<b>Move</b>	Click the Move option to move a work item already assigned to an account to another account. This option displays on the Other Actions drop-down button and only if the work item is already assigned to an account. Refer to <a href="#">Managing Work Items</a> for more information.
<b>Claim</b>	Click the Claim option to create a claim for the work item. This option displays on the Other Actions drop-down button.
<b>Endorse</b>	Click the Endorse option to create an endorsement for the work item. This option displays on the Other Actions drop-down button.

# Search and Filter

AgencyPortal includes a powerful search feature that allows you to search and filter accounts and work items throughout multiple areas within the application. The search and filter behavior, which allows you to perform a search using a minimum of three characters, automatically filters or auto-suggests results (depending on where within the application you are using the feature), and select from a variety of filter options.

The search and filter feature is included within the following pages and components of the application:

- My Accounts (both card and classic view)
- My Work (both card and classic view)
- Account merge
- Work item move
- ACORD form upload

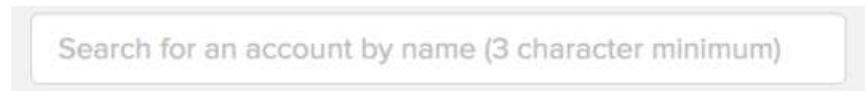
## Searching for Accounts and Work Items

When using the unified search and filter feature on the My Accounts or My Work pages, accounts and work items automatically filter when at least three characters are entered in the search box. This works consistently on both the card and classic view versions of these pages.

### Note

The basic search box uses "contains" logic and is not case sensitive. However, you must enter the exact string of characters, including symbols. For example, a search for ABC will find ABC Industries, but not A.B.C. Industries.

#### My Accounts Search Box



The search box on the My Accounts page allows you to search for accounts by name.

The search feature searches across three data elements: First Name (personal account), Last Name (personal account) and Company Name (commercial account). This allows you to search across personal and commercial accounts simultaneously.

Enter at least the three characters of a name to begin filtering the accounts on the page. You can also enter the whole name, if known, to find all accounts with the name and/or to find fewer results. Backspacing in the search box re-engages active filtering and displays results based on the value in the search box (as long as three characters are entered in the box).

#### Example 1

A search is performed for "joh," which is part of the name "John Doe." The search finds five accounts with a name on an account that contains "joh."

5 Account(s) Type ▾ Sort By ▾ More Options ▾ My Filters ▾

Clear All Sort By: Updated

**Job**

<b>Add New Account</b>	John Doe Commercial 1226 2 Archer Lane Andover MA 01810	Johnson Industries Commercial 1172 25 Main St Northborough MA 01532
<b>Joe Johnson</b> Personal 1173 55 Sleeper St Boston MA 02210	<b>John Doe</b> Personal 1171 55 State Street Boston MA 02210	<b>John's Chainsaws</b> Commercial 1143 55 Story St. Boston MA 93232

## Example 2

A search is performed for the full name "Jane Green." The search finds three accounts with the full name "Jane Green."

3 Account(s) Type ▾ Sort By ▾ More Options ▾ My Filters ▾

Clear All Sort By: Updated

**Jane Green**

<b>Add New Account</b>	Jane Green Personal 1176 56 Commonwealth Ave Cambridge MA 02215	Jane Green Personal 1175 393 Oak St Kittery ME 03941
<b>Jane Green</b> Personal 1174 22 Main St Boston MA 02210		

## My Work Search Box

Search for a customer by name (3 character minimum)

The search box on the My Work page allows you to search by customer name. Enter at least the three characters of a customer's name to begin filter the work items on the page. You can also search for the entire customer name, if known, to find all work items associated with the customer.

### Example 1

A search is performed to find all work items with "moo," which is in the company name IRMOO, Inc. The search finds three work items with a customer name that contains "moo."

The screenshot shows the 'My Work' page interface. At the top, there is a search bar containing the text 'moo'. Below the search bar, the page displays three work items:

- 1180: AUTOP**  
Janet Moore %  
Effective On 04-26-2015  
Premium: \$0.00  
Quick Quote  
In-Progress
- 1182: HOME**  
Samuel Moore %  
Effective On 04-26-2015  
Premium: \$0.00  
Full Application  
In-Progress
- 1181: WORK**  
IRMOO, Inc %  
Effective On 04-26-2015  
Premium: \$0.00  
Full Application  
In-Progress

### Example 2

A search is performed to find all work items with a customer named "John Doe." The search finds three work items with a customer named "John Doe."

The screenshot shows the 'My Work' page interface. At the top, there is a search bar containing the text 'John Doe'. Below the search bar, the page displays three work items:

- 1183: HOME**  
John Doe %  
Effective On 04-27-2015  
Premium: \$0.00  
Quick Quote  
In-Progress
- 1185: WORK**  
John Doe %  
Effective On 04-27-2015  
Premium: \$0.00  
Quick Quote  
In-Progress
- 1184: AUTOP**  
John Doe %  
Effective On 04-26-2015  
Premium: \$0.00  
Quick Quote  
In-Progress

If you want to narrow down your search results even further, use the Advanced Search drop-down on both the My Accounts and My Work pages.

The following advanced search options are available on the My Accounts page:

The screenshot shows a search interface with the following fields and buttons:

- More Options ▾
- My Filters ▾
- Account Number: [Text input field]
- City: [Text input field]
- State: [Text input field] with a dropdown menu labeled "Select State"
- Zip Code: [Text input field]
- Search [Blue button]
- Save Filter As: [Text input field] with a Save [Blue button]

The following displays when incorrect data is entered into the fields or no results are found:

- If an account number is entered in the Account Number field that is less than the number of digits for an account, the following message displays: *Number should be an exact match*.
- When a value is entered in the Account Number field that does not match any accounts, no results display on the My Account page.
- If less than three characters are entered in the City or Zip Code fields, the following message displays: *Please provide at least 3 characters*.
- When a value is entered in the City, State or Zip Code field that does not match any accounts, no results display on the My Account page.

The following advanced search options are available on the My Work page:

The screenshot shows a search interface with the following fields and buttons:

- More Options ▾
- My Filters ▾
- Work Item ID: [Text input field]
- Effective: [Text input field] with a dropdown menu labeled "Select Operator" and a date input field "MM-dd-yyyy" with a calendar icon, followed by a Search [Blue button]
- Save Filter As: [Text input field] with a Save [Blue button]

The Effective field includes the following search options:

- on the effective date
- after the effective date
- not on the effective date
- between effective dates (a second date field displays)

- before the effective date

If a work item ID is entered in the Work Item ID field that is less than the number of digits for a work item, the following message displays:  
*Number should be an exact match.*

#### Note

To clear any entered advanced search criteria, click on the My Work tab to reset the page. Any information entered in the advanced search options fields are saved when using the save filter functionality. Refer to the [Saving Work Item Filters](#) section below.

### Sorting and Filtering Accounts and Work Items

You can also sort and filter the account or work item results on their respective pages. The sort and filter feature can be used in combination with the search functionality mentioned above or by itself.

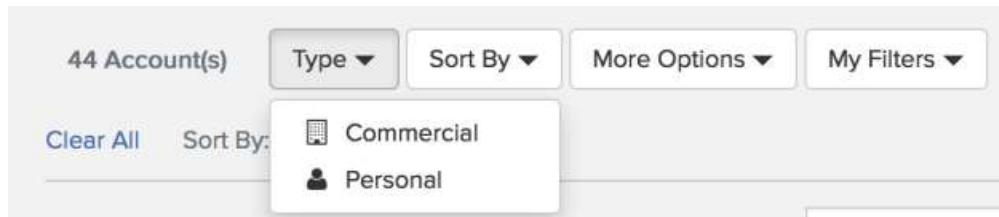
The My Accounts and My Work pages have their own set of filters to display accounts and work items, and they work the same way on both the card and classic views of the pages.

#### Sorting and Filtering Accounts

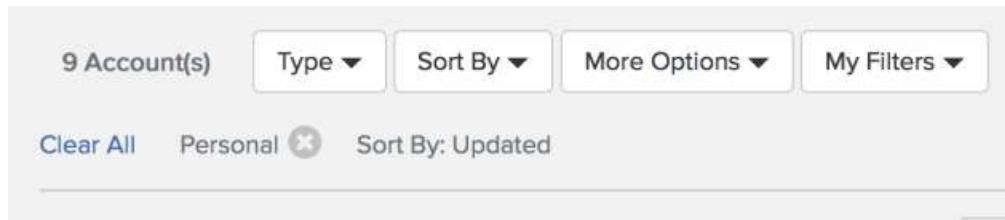
You can filter the display of accounts on the My Accounts page by type and/or sort the results by selected criteria. The Sort by: Updated filter is the default sort for the work list and cannot be removed (only changed).

#### Example

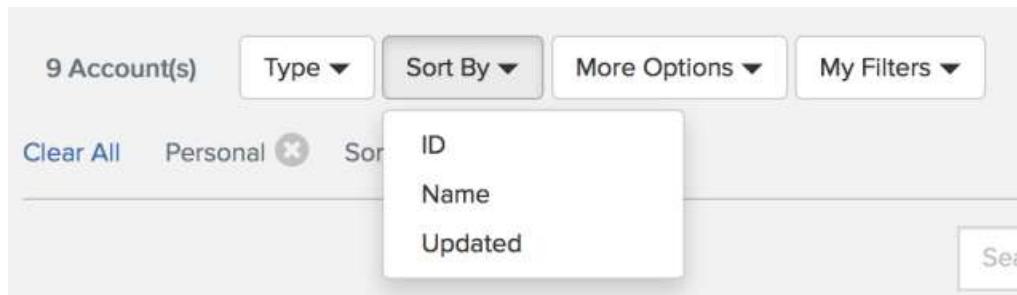
You want to display all personal accounts, so you select the Personal option on the Type drop-down.



The results on the page change to display only personal accounts. To indicate this filter is in use, Personal displays under the account options.



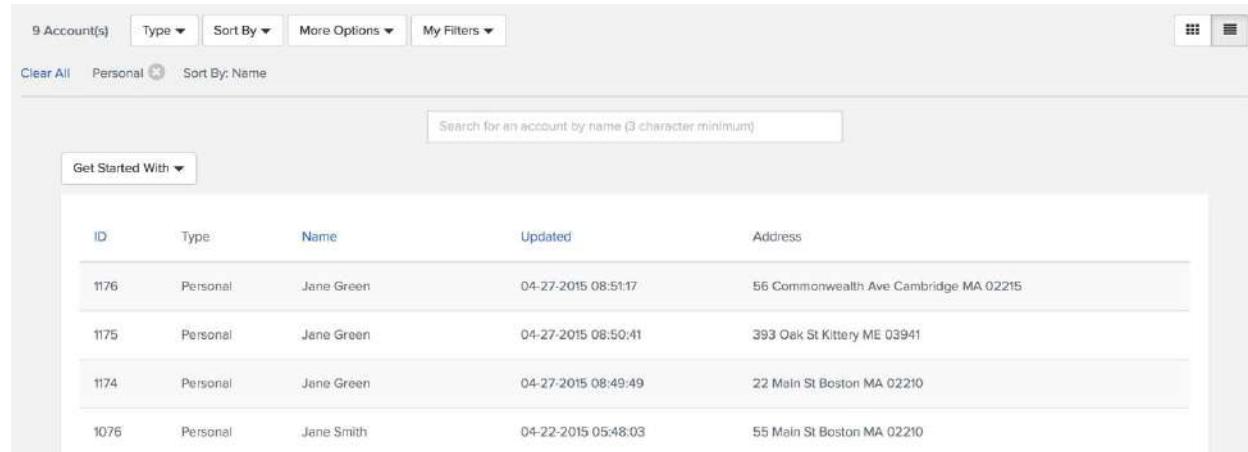
You also decide you want to sort the results by name, so you select Name from the Type drop-down.



The accounts then display in alphabetic order by name.

After you review your results, you have the option to add any additional filters and change the sort order to any of the available options. To remove a filter, simply click the  icon next to the filter.

If you are using the classic view of the My Accounts page, you also have the option to sort the items on the page by simply clicking the column header. Available sort options are ID, Name and Updated.



ID	Type	Name	Updated	Address
1176	Personal	Jane Green	04-27-2015 08:51:17	56 Commonwealth Ave Cambridge MA 02215
1175	Personal	Jane Green	04-27-2015 08:50:41	393 Oak St Kittery ME 03941
1174	Personal	Jane Green	04-27-2015 08:49:49	22 Main St Boston MA 02210
1076	Personal	Jane Smith	04-22-2015 05:48:03	55 Main St Boston MA 02210

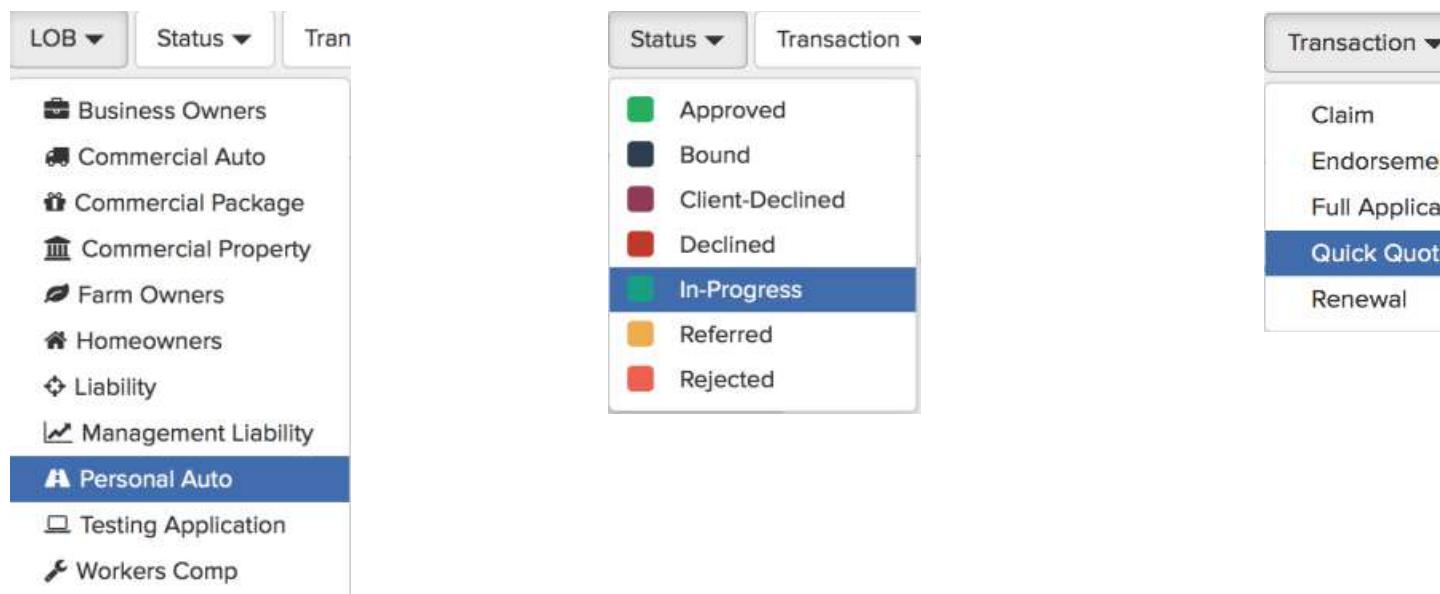
## Note

Filter selections remain when you switch from card to classic view (or vice versa) or navigate to another page within the application. To remove any selected filters, you must click the Clear All hyperlink.

## Sorting and Filtering Work Items

Filtering work items works the same way as filtering accounts except it includes different criteria. You can filter work items by LOB, status and/or transaction type, as well as sort by selected criteria.

For example, you only want to display work items that belong to the Personal Auto LOB, are in an in-progress status and are quick quotes. To display only these work items, you select the following:



LOB ▾

-  Business Owners
-  Commercial Auto
-  Commercial Package
-  Commercial Property
-  Farm Owners
-  Homeowners
-  Liability
-  Management Liability
-  Personal Auto
-  Testing Application
-  Workers Comp

Status ▾

- Approved
- Bound
- Client-Declined
- Declined
- In-Progress
- Referred
- Rejected

Transaction ▾

- Claim
- Endorsement
- Full Application
- Quick Quote
- Renewal

The number of results updates and the filters selected display under the work item options.

3 Work Item(s) LOB ▾ Status ▾ Transaction ▾ Sort By ▾ More Options ▾ My Filters ▾

Clear All Personal Auto In-Progress Quick Quote Sort By: Updated

Search for a customer by name (3 character minimum)

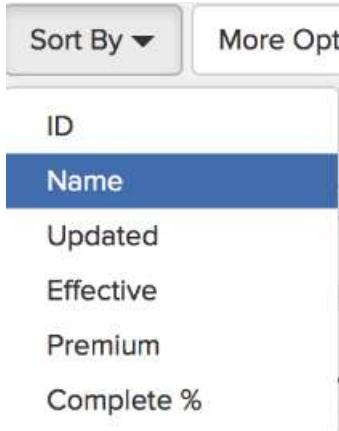
1184: AUTOP John Doe % Effective On 04-26-2015 Premium: \$0.00 Quick Quote In-Progress

1180: AUTOP Janet Moore % Effective On 04-26-2015 Premium: \$0.00 Quick Quote In-Progress

1029: AUTOP Betty White % Effective On 04-15-2015 Premium: \$0.00 Quick Quote In-Progress

Add New Work Item

You then want to sort the results by customer name, so you select the Name option on the Sort By drop-down.



The filtered work items then display in alphabetic order by name.

After you review your results, you have the option to add any additional filters and change the sort order to any of the available options. To remove a filter, simply click the icon next to the filter. However, the Sort by: Updated filter is the default sort for the work list and cannot be removed (only changed).

If you are using the classic view of the My Work page, you also have the option to sort the items on the page by simply clicking the column header. Available sort options are ID, Name, Updated, Effective, Premium and Complete %.

ID	LOB	Name	Status	Updated	Effective	Premium	Transaction Type	Complete %
1190	Personal Auto	Janey Jenkins	In-Progress	04-27-2015 10:03:52	04-27-2015	0	Quick Quote	88
1184	Personal Auto	John Doe %	In-Progress	04-27-2015 09:16:47	04-26-2015	0	Quick Quote	88
1180	Personal Auto	Janet Moore %	In-Progress	04-27-2015 08:59:45	04-26-2015	0	Quick Quote	88

## Note

Filter selections remain when you switch from card to classic view (or vice versa) or navigate to another page within the application. To remove any selected filters, you must click the Clear All hyperlink.

## Saving Work Item Filters

AgencyPortal includes the ability for users to save filter criteria they use most often, which allows them to quickly access work items that fit the selected criteria with the simple click of an option.

A saved filter can include LOB, status, transaction and sort by criteria. This feature does not allow you to save criteria entered in the work item ID or effective date fields.

### Note

Any search criteria entered at the time a filter is saved is not included in the saved filter. This functionality only saves selected filters; it does not save any criteria entered in any of the search boxes.

When filters are active (either saved or unsaved), they display under the filter menu.



If you change your mind and do not want to use a particular filter, click the icon next to the filter to remove it. If you want to remove all selected filters, click the Clear All hyperlink.

After you make a filter selection, click the More Options drop-down.

This drop-down includes work item ID and effective date criteria and is also the location where you can name and save your filter.

To save your filter criteria, enter a name in the Save Filter As field and click the Save button. Your filter criteria is now saved and is available for selection from the My Filters drop-down.

If you later decide you no longer want a saved filter, simply click the icon next to the filter in the My Filters drop-down to delete it.

### Searching for Accounts within Other Components of the Application

When using the unified search feature within the merge accounts, move work items or upload ACORD forms components, the search auto-suggests the results as soon as at least three characters are entered into the search box.

## Example 1

You want to merge John Doe's personal account with his wife's personal account. Her name is Jane Doe.

A screenshot of a software window titled "Merge Accounts". At the top, it says "Search for an account to merge with "John Doe – Account # 1425"". Below is a search bar with the placeholder "Search for a source account by name (3 character minimum)". A magnifying glass icon is to the right of the search bar. The background shows a blurred list of account names.

You enter "Jan," the first three characters of Jane's name. The first four suggested accounts display within the auto-suggest results.

Since there are more than four results, "4 others" displays to indicate there are more results.

A screenshot of the "Merge Accounts" window. The search bar now contains "Jan|". Below it, a dropdown menu lists four accounts: "Jane Doe (#1421 - Personal)", "Jane Green (#1176 - Personal)", "Janson Builder Co (#1189 - Commercial)", and "JAN Jeans, Inc (#1188 - Commercial)". To the right of these entries, a vertical column shows dates: "015-05-14", "015-05-14", and "015-05-14". Below the list, the text "4 others" is visible. A magnifying glass icon is at the end of the dropdown menu.

There are eight accounts that match "Jan"; however, you don't see the Jane Doe account you want in the first four results that display. Click "4 others" in the list of results to display all accounts that match the entered criteria.

Merge Accounts

Search for an account to merge with " John Doe – Account # 1425 "

8 Account(s) Found

	Jane Doe	#1421	<input type="button" value="Select"/>
	Jane Green	#1176	<input type="button" value="Select"/>
	Janson Builder Co	#1189	<input type="button" value="Select"/>
	JAN Jeans, Inc	#1188	<input type="button" value="Select"/>
	Janey Jenkins	#1187	<input type="button" value="Select"/>
	Jane Doe	#1186	<input type="button" value="Select"/>
	Jane Green	#1175	<input type="button" value="Select"/>
	Jane Green	#1174	<input type="button" value="Select"/>

The Jane Doe account you are looking for now displays in the results and is available for selection.

## Example 2

As an alternative, you can enter the entire name (if known) to find the account associated with the customer.

Merge Accounts

Search for an account to merge with " John Doe – Account # 1425 "

2 Account(s) Found

	Jane Doe	#1421	<input type="button" value="Select"/>
	Jane Doe	#1186	<input type="button" value="Select"/>

As you can see, this is a little different than the search feature on the My Accounts and My Work pages; however, the results and functionality work in the same way.

## Filterlist Search and Filter

AgencyPortal includes a filterlist search and filter feature that allows you to quickly search for and filter on a field when the list of possible values is extensive. We've used a nifty suggestive search plug-in called [Select2](#) to implement this feature. This feature works just like the search suggestions common in most web browsers or search engines.

When you click the field box to select a value, a search box displays.

* Class Code	Select One
me Employees	<input type="text"/>
me Employees	<b>0005 Farm: Nursery Employees and Drivers</b>
inual Exposure	0008 Farm: Gardening-Market Or Truck and Drivers
oluntary Comp	0016 Farm: Orchard and Drivers
	0034 Farm: Poultry Or Egg Producer and Drivers
	0035 Farm: Florist and Drivers

As soon as a number or character is entered into the field search box, the list of valid values automatically filters based on the criteria entered. When a match is found, the criteria entered is underlined.

* Class Code	Select One
me Employees	<input type="text"/> 0042
me Employees	<b>0042 Landscape Gardening and Drivers</b>
* Class Code	Select One
ne Employees	<input type="text"/> gard
ne Employees	<b>0008 Farm: Gardening-Market Or Truck and Drivers</b>
	0042 Landscape <u>Gardening</u> and Drivers

This makes it easier to find and select a desired value instead of having to scroll through the list to find the one you want. Out of the box, this feature is available for the class code field in the Workers Compensation line of business. Refer to the filterlist subtopic in the [Transaction Definition/Page Library](#) topic for information on how to configure this feature for other fields and lines of businesses.

# Work Items

The My Work tab allows you to add and modify work items. This section includes details on how to perform these tasks, as well as additional information about work items.

# About Work Items

This section details a number of topics related to work items.

- [Transaction Types](#)
- [Transaction Pages](#)
- [Page Types](#)
- [Page Elements](#)
- [Work Item Assistant](#)
- [Timeline](#)

# Transaction Types

Each line of business is configured to display a certain set of transaction pages. These pages are set up during project implementation, and determine what you see and must enter when creating a work item.

Transaction pages can be created for any of the following transaction types:

- new business (full application)
- quick quote
- claim
- endorsement
- renewal

The data entry for a work item varies, depending on the line of business and transaction type. The amount and nature of information differs for each type. For instance, quick quotes have fewer pages than a new business application and a claim transaction requires a different data set than new business.

Each transaction type has its own set of transaction pages and each page is configured to display a certain set of fields and elements. Refer to the following for more information:

- [Page Types](#)
- [Page Elements](#)

# Page Types

The pages that make up a transaction depend on the information required for the transaction and how they are configured. Transaction pages can be one of the following types:

## Data Entry

A type of page used to collect basic, non-repeating data.

### Example

The agency/applicant information page is an example of a data entry transaction page. This page is used to enter basic information for the agency and applicant associated with the work item

## Roster

A type of page that collects a group of repeating data.

### Example

The locations transaction page is an example of a roster page. This page is used to add one to multiple locations associated with the transaction.

Roster pages allow you to add, edit and delete repeating data associated with the transaction.

## Custom

A custom page created by developers to display data that does not fit in the roster or data entry page type.

### Example

A summary page is an example of a custom page that can display custom data. Refer to [Summary Pages](#) below for more information.

The screenshot shows a custom "Summary" page for a "Personal Auto QQ (ID: 1184 - John Doe)" application. At the top, there are tabs for "Normal" and "Side-By-Side". A green box labeled "In-Progress" is visible. The page displays three payment options: \$1838 (Monthly), \$2468 (Bi-Annual), and \$1390 (Payment In Full). Below these are sections for "Producer Code" (Agency Customer ID), "John Doe" (Co-Applicant, Policy Term: 04-27-2015 to 04-27-2016), and "Documents" (View Acord 90).

## Messages

When working on a transaction page and continuing on to the next page, the following messages can display for a work item:

- Validation – This message indicates you have performed an action that is invalid. You cannot continue when this message displays.  

① Applicant Information.Date Business Started must be a valid date entered in MM-dd-yyyy format  
Applicant Information.Tax ID must be a valid FEIN with a format of nn-nnnnnnnn
- Warning – This message indicates that the data you entered in the application requires review. You can continue creating or editing the work item when this message displays.  

② Effective Date cannot be prior to today. This work item will be referred to an underwriter.
- Information – This message displays informational text based on an action you performed. You can continue creating or editing the work item when this message displays.

These messages display based on the XARC rules established during project implementation. Refer to the [XARC Rules](#) topic in the Developer Guide section for more information on configuring XARC rules and messages.

## Work Item Links

On each transaction page, available work item links display in the upper right corner. These options can include the following actions:

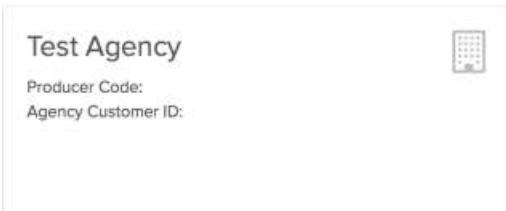
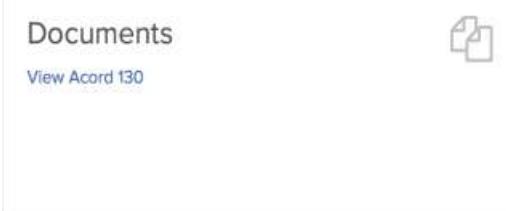
Account Name and Number  <a href="#">ABC Company Account # 8781</a>	Click this hyperlink to view the account information page associated with the work item. This hyperlink only displays if an account is associated with the work item.
History  <a href="#">① History</a>	Click this hyperlink to view work item history and events. Refer to the <a href="#">Timeline</a> topic for more information.

Upload   <a href="#">Upload</a> ▾	Click this hyperlink to view the source information for the uploaded work item and any relevant help information. This hyperlink only displays if the work item was uploaded via the Turnstile integration feature. Refer to the <a href="#">Upload</a> topic for more information.
Corrections   <a href="#">Corrections</a> ▾	Click this hyperlink to view any automatic corrections for an uploaded work item. This hyperlink only displays if the work item was created from an upload and automatic corrections were made.
My Defaults   <a href="#">My Defaults</a> ▾	Click this hyperlink to apply defaults to the selected work item, manage available defaults, view available fields to set default values and view help on how to set defaults. This hyperlink and corresponding drop-down only display when My Defaults is configured for the line of business. Refer to the <a href="#">My Defaults</a> topic for more information.

## Summary Pages

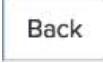
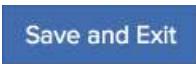
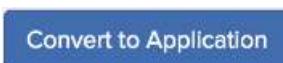
Summary pages are custom pages used to display basic, non-repeating information. Most commonly, they are used to display quote/rating results and submission summary pages. The details and information that displays on a summary page is customized per project during implementation.

The following are some common summary page sections, which are included within the AgencyPortal application by default:

Agency    <b>Test Agency</b> Producer Code: Agency Customer ID:	A summary of the agency/producer information for the submission. Details of this section can be customized per project.
Applicant    <b>John Doe</b> Co-Applicant: Policy Term: 04-28-2015 to 04-28-2016	A summary of the applicant information. Details of this section can be customized per project.
Documents    <b>Documents</b> <a href="#">View Acord 130</a>	A list of documents attached to the submission. Each document is a hyperlink.  ACORD forms can be grouped together and viewable from a single hyperlink or listed individually. Refer to the <a href="#">PDF Generation Utility</a> topic in the Developer Guide for more information on how ACORD forms can be grouped on the policy summary page.

Status	The color-coded representation of the work item status. Colors and statuses are customized per project. The status and status color that displays on the summary page correlates to the status and status color that displays on the My Work page.
Policy details summarized by screen	This section provides a page-by-page summary of the data entered for the transaction.

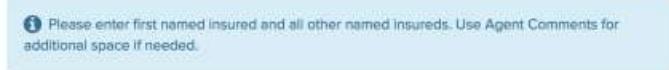
Also included on the summary page are the following options:

 Back	Back	Returns you to the last visited transaction page.
 Save and Exit	Save and Exit	Saves any changes to the work item and returns you to the My Work page if the work item is not associated with an account or the account information page if the work item is associated with an account.
 Convert to Application	Convert to Application	Displays for a quick quote application. Allows you to convert the quick quote to a full application.

# Page Elements

A page element is the part of the transaction page that displays a group of items (e.g., fields) in a section.

Transaction page can consist of the following page elements:

Fieldset	<p>Each page is comprised of fieldsets. A fieldset, or section, is a group of fields logically grouped by their relation to one another. The label on a fieldset is typically indicative of the subject for the grouping. Details of a fieldset can be customized during project implementation.</p> <p><b>Example</b></p> <p>For example, the Agency Information fieldset contains fields that relate to the agency.</p> <p><b>Agency Information</b></p> <p>* Agency Name <input type="text"/></p> <p>* Address Line 1 <input type="text"/></p> <p>Address Line 2 <input type="text"/></p> <p>* City <input type="text"/></p> <p>* State <input type="select"/> Select One</p> <p>* Zip Code <input type="text"/></p>
Tip	<p>A page or fieldset can include a tip element. Tips elements are used to display informational messages regarding a field or set of fields.</p> <p><b>Example</b></p> <p>The following is an example of a tip that displays for the Applicant Information fieldset:</p> <p></p> <p><b>Applicant Information</b></p> <p>* Applicant Name <input type="text"/></p> <p>Doing Business As <input type="text"/></p> <p>Tax Id Type <input type="select"/></p> <p>Tax Id <input type="text"/></p>
Questionnaire	<p>Data entry or custom pages can include questionnaire field elements. A questionnaire includes a list of questions with Yes or No radio buttons.</p>

	<p><b>Example</b></p> <p>The following is an example of a questionnaire:</p> <p><b>Questionnaire</b></p> <table border="1"> <thead> <tr> <th></th><th style="text-align: center;">Yes</th><th style="text-align: center;">No</th></tr> </thead> <tbody> <tr> <td>* Any farming or other business conducted on premises? (including day/child care)</td><td style="text-align: center;"><input checked="" type="radio"/></td><td style="text-align: center;"><input type="radio"/></td></tr> <tr> <td>* Any residence employees?</td><td style="text-align: center;"><input type="radio"/></td><td style="text-align: center;"><input checked="" type="radio"/></td></tr> <tr> <td>* Any flooding, brush, forest fire hazard, landslide, etc?</td><td style="text-align: center;"><input type="radio"/></td><td style="text-align: center;"><input checked="" type="radio"/></td></tr> <tr> <td>* Any other residence owned, occupied or rented?</td><td style="text-align: center;"><input type="radio"/></td><td style="text-align: center;"><input checked="" type="radio"/></td></tr> <tr> <td>* Any other insurance with this company?</td><td style="text-align: center;"><input type="radio"/></td><td style="text-align: center;"><input checked="" type="radio"/></td></tr> <tr> <td>* Has insurance been transferred within agency?</td><td style="text-align: center;"><input type="radio"/></td><td style="text-align: center;"><input checked="" type="radio"/></td></tr> <tr> <td>* Any coverage declined, cancelled, or non-renewed during the last three (3) years?</td><td style="text-align: center;"><input type="radio"/></td><td style="text-align: center;"><input checked="" type="radio"/></td></tr> </tbody> </table> <p>Each field is a question with a Yes and No radio button.</p>		Yes	No	* Any farming or other business conducted on premises? (including day/child care)	<input checked="" type="radio"/>	<input type="radio"/>	* Any residence employees?	<input type="radio"/>	<input checked="" type="radio"/>	* Any flooding, brush, forest fire hazard, landslide, etc?	<input type="radio"/>	<input checked="" type="radio"/>	* Any other residence owned, occupied or rented?	<input type="radio"/>	<input checked="" type="radio"/>	* Any other insurance with this company?	<input type="radio"/>	<input checked="" type="radio"/>	* Has insurance been transferred within agency?	<input type="radio"/>	<input checked="" type="radio"/>	* Any coverage declined, cancelled, or non-renewed during the last three (3) years?	<input type="radio"/>	<input checked="" type="radio"/>
	Yes	No																							
* Any farming or other business conducted on premises? (including day/child care)	<input checked="" type="radio"/>	<input type="radio"/>																							
* Any residence employees?	<input type="radio"/>	<input checked="" type="radio"/>																							
* Any flooding, brush, forest fire hazard, landslide, etc?	<input type="radio"/>	<input checked="" type="radio"/>																							
* Any other residence owned, occupied or rented?	<input type="radio"/>	<input checked="" type="radio"/>																							
* Any other insurance with this company?	<input type="radio"/>	<input checked="" type="radio"/>																							
* Has insurance been transferred within agency?	<input type="radio"/>	<input checked="" type="radio"/>																							
* Any coverage declined, cancelled, or non-renewed during the last three (3) years?	<input type="radio"/>	<input checked="" type="radio"/>																							
Script	<p>Lines of conditional code, such as JavaScript, that is active within the page to display conditional information.</p> <p><b>Example</b></p> <p>The following is an example of a field with a script element:</p> <p><b>Policy Term</b></p> <table border="1"> <tr> <td>* Effective Date</td> <td>10-10-2014</td> <td></td> </tr> <tr> <td>* Expiration Date</td> <td>10-10-2015</td> <td></td> </tr> </table> <p>In this example, the expiration date of a policy is scripted to be one year from the effective date so you do not have to enter in the expiration date for every policy created.</p>	* Effective Date	10-10-2014		* Expiration Date	10-10-2015																			
* Effective Date	10-10-2014																								
* Expiration Date	10-10-2015																								

## Fields

Each page element contains individual field elements. The fields on a transaction page can be any of the following:

- text
- date
- selectlist
- radio
- textarea
- hidden
- message
- file
- filterlist
- check box

## Field Validation

Field validation is available for each field type set up within the application, and is determined during the implementation process. Available field validations vary by field element. The validation determines whether the value is required (i.e., you must enter a value in the field before continuing). Field validations can include the following, based on field element:

Field Element	Available Validations	Purpose
Text only	ACORD Date Format	The date entered must be in the ACORD date format of YYYY-MM-DD.
Text only	Alphanumeric	The value entered must be comprised of only letters and numbers.
Text, Date, Selectlist, Radio and Filterlist	Custom	The value entered is compared against a script (e.g., com.agencyport.shared.validators.YearValidator)
Text and Date	Date Compare*	If selected, the date entered in the application is verified against the date entered here. The valid formats are before('YYYY-MM-DD') and after('YYYY-MM-DD').
Text only	Email	The value entered must be a valid email address string.
Text only	FEIN	The value entered must comply with the following format: 'nn-nnnnnnn.'
Text only	Regular Expression	The value entered must match the pattern expressed in the regular expression.
Text only	Length	The value entered must be exactly this number of characters.
Text only	Max Length	The value entered must be no more than this number of characters.
Text only	Max Value	The value entered must be no more than this value.
Text only	Min Length	The value entered must be no less than this length of characters.
Text only	Min Value	The value entered must be no less than this value.
Text only	Numeric	The value entered must be numeric.
Text only	Phone	If selected, the value must comply with one of the following formats: (nnn) nnn-nnnn or international ACORD phone format, complying with regular expression: "\+\S+\-\S+\-\S+\(\+\ \S+\)\?".
Text, Date, Selectlist, Radio and Filterlist	Required	A value must be entered.
Text only	SSN	The value entered must comply with the following format: 'nnn-nn-nnnn.'
Text only	US Date	The value entered must conform to a mm-dd-yyyy format and must be a valid date. Month and day portions must be left zero padded where needed.
Text only	US Zip Code	The value entered must comply with either of the following formats: 'nnnnn' or 'nnnnn-nnnn.'
Selectlist, Radio and Filterlist	Verify List	This is used with upload when you want to validate a value uploaded against a value in the list.

\* This validation only runs on the server side since there is a comparison between the entered date and a reference date.

## Example

The following is an example of a field validation for Address Line 1 within the applicant's mailing address.

Mailing Address

\* Address Line 1:  This field is required

Address Line 2:

This field validation requires you to enter alphanumeric data in the field before proceeding.

## Field Helpers

Field helpers can also be available for fields. Field helpers provide additional field information that helps you complete a field. This can include a format mask or help text.

### Format Mask

A format mask determines the mask over the field to provide you with the data pattern to enter in the field.

Categories of format mask are:

- fixed length text inputs - the input fields are pre-filled with dashes (e.g., date and tax ID).
- numeric inputs (variable length) (e.g., format for US dollars).

Types of format masks are:

- ACORD date format
- US date format
- phone
- social security number
- FEIN
- US zip code short
- US zip code long
- custom Mask
- US dollars
- integer
- decimal
- custom numeric

## Example

The following is an example of a format mask added to the Business Phone field. This helps you determine the format of the data required for the field.

Business Phone:  (3\_\_\_\_) \_\_\_\_-\_\_\_\_

## Help Text

Help text may also be available for particular fields. The following are the different types of available help text:

	<b>balloon</b>	Used to provide help text at the field level.  <b>Example</b>  The following is an example of a balloon help text type:   Agency Customer ID <input type="text"/> ? Enter the Agency ID for the Insured.
-------------------------------------------------------------------------------------	----------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

		<p>The  icon displays next to the Agency Customer ID field. If you're not sure what type of data to enter in this field, click the icon to read the help text.</p>
	calendar	<p>Used to display a calendar widget to collect a date.</p> <p><b>Example</b></p> <p>The following is an example of a calendar widget:</p>  <p>Click the  icon next to the Date Business Started field. The calendar widget displays to select the date.</p>

# Navigation Menu

Within the transaction workflow, transaction page titles are listed on the left hand side of the page. When you create a line of business transaction, these items are read only (greyed out) until you visit the page and enter the required information; you cannot jump ahead to a page you have not yet visited. After a work item is created and all required fields are entered, you can go back into the work item and navigate through any of the transaction pages.

## Example 1

The following shows a commercial auto transaction page at the time of creation.

The screenshot displays a transaction creation interface. On the left, a vertical navigation menu titled "Agency/Applicant Information" lists several sections: Policy Information, Locations, State Rating Factors, Rating Classifications, Individuals Included/Excluded, General Information, Prior Carrier Information/Loss Information, Billing/Contact Information, and Summary. Most of these items are greyed out, indicating they have not been visited. The "Agency Information" section is the active page, shown in white. It contains fields for Agency Name, Address Line 1, Address Line 2, City, State (a dropdown menu), Zip Code, CS Representative Name, Producer Phone, Mobile Phone, and Producer Fax. Each field has an asterisk (\*) next to it, signifying it is a required field. To the right of the form is a sidebar with "Comments" (an empty text area and a "Comment" button), "Documents" (a section for uploading files with a "Drag file here" area and an "Add File" button), and "Upload A File" (a dropdown menu for selecting document types, with "Select One" currently chosen).

Notice that only the transaction page you are currently on, the agency/applicant information page, is available, and the other pages are greyed out. This is because you have not entered any required information on these pages.

## Example 2

When a work item is created from an upload, there may be missing information for the transaction. Pages that have not been visited yet and do not include information are greyed out; pages that need attention, such as incomplete data or missing data, display in red. You must review all of the pages and click the Continue button to enable all transaction pages in the transaction menu.

The screenshot shows a transaction page for a Commercial Auto policy. On the left is a sidebar menu with sections like General Information, Drivers, Locations, Vehicles & Coverages, Policy Coverages, Applicant Questions, Underwriting Questions, Additional Interests, Prior Carriers, Loss History, and Summary. The main content area is titled "Agency Information" and contains fields for Agency Name (Insurance Office of America - LN), Address Line 1 (26 West State Road 128), Address Line 2 (empty), City (Longwood), State (Florida), Zip Code (32750), Producer Phone ((407) 555-3000), Producer Fax ((407) 555-7933), E-mail Address (empty), Producer Code (empty), and Producer Sub Code (empty). A note at the top says "\* Denotes Required Fields". To the right are three sidebar modules: "Comments" (listing "Great Salesman" notes), "Documents" (listing "#4A.pdf ACORD Form 194 kb 04-27-2015 14:33:30"), and "Upload A File" (with a "Add File" button).

### Example 3

After a work item is created and you select the work item to make changes, the transaction page menu displays all transaction pages as enabled. This allows you to jump from transaction page to transaction page to review or modify any information.

The screenshot shows a transaction page for a Workers Comp QQ policy. The sidebar menu includes sections for Applicant Information, Policy Information, Locations, State Rating Factors, Rating Classifications, and Summary. The main content area is titled "Agency Information" and contains fields for \* Agency Name (Test Agency), \* Address Line 1 (55 Main St), Address Line 2 (empty), \* City (Boston), \* State (Massachusetts), \* Zip Code (02210), CS Representative Name (empty), Producer Phone (empty), Mobile Phone (empty), and Producer Fax (empty). A note at the top says "\* Denotes Required Fields" and a message below says "For help with a specific field or question, click on the question mark icon." To the right are three sidebar modules: "Comments" (listing "Great Salesman" note), "Documents" (empty), and "Upload A File" (with a "Add File" button and a "Drag file here" area).

# Autosave

AgencyPortal's Autosave feature automatically saves data entered in non-encrypted fields after a configured set of time to ensure that your work is saved. This feature is available for both work items and accounts and kicks in when you make changes on a transaction page (excluding the comment section of the Work Item Assistant), but have not manually saved your changes. By default, this occurs every 60 seconds.

## Note

Autosave functionality is only available if it is turned on within your application. Refer to the [Autosave](#) topic in the Developer Guide section for information on how to set up and configure this feature.

To indicate an autosave has occurred, a time stamp displays at the top of the transaction page with the date and time of the last autosave.

**Auto Saved: 02-05-2015 11:03:12**

The time stamp only displays if changes were made to transaction data and continues to display until you manually commit your changes.

## Note

Transaction page changes are not autosaved if you log out of your session before the autosave interval time kicks in (e.g., 60 seconds). In addition, the following are not included in the autosave feature:

- fields designated to include encrypted data are not autosaved
- data entered on the account people or locations roster pages are not autosaved

The last page you visited (regardless of whether you made any changes to the page) displays when you re-open a work item after you log off or your session expires. This is only available if your application is set up to use the Autosave feature.

# Work Item Assistant

Work Item Assistant displays on every transaction page based on user permissions and configuration. The Work Item Assistant feature can be configured to include the following sections:

- comments
- file attachments
- users

This feature allows you to add comments and file attachments on any page within a transaction, which are also available to other users when they are working within the same work item. This element of the feature allows for nearly real-time communication among users that are in the same work item.

The following are a couple of important points about Work Item Assistant:

- The Work Item Assistant feature can be configured to display at the transaction level.
- Each section of the Work Item Assistant is available to all users by default; however, the availability of the feature can be configured by user role and role group (such as agent or underwriter). Sections within Work Item Assistant can be configured to allow some user roles more access and permissions than others. Restrictions can also be put into place within a section so that only a specific set of users can view and access that section. Users associated with a specific role or role group who are not defined for a specific section cannot view that section within the application.

## Example

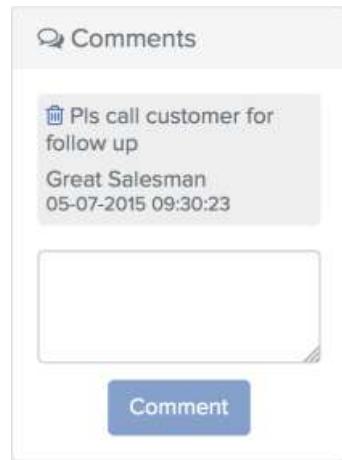
The assistant can be configured so that internal comments are only viewable by underwriters and external comments are viewable by underwriters and agents.

Comments and internal comments sections are included; multiple comment and file attachment sections are set up with unique names and user/role groups.

- Comments, file attachments and active user usernames are added to the assistant within 15 seconds of their entry (i.e., it may take up to 15 seconds for a comment to display, but no longer).
- Multiple sections of the same type can be configured (e.g., a given assistant can have multiple comment sections).

## Comments

The Comments section displays any comments entered for the work item, allows you to create new comments and delete any existing comments.



To leave a comment, enter text in the text box and click the Comment button. The Comment button displays as greyed out until text is entered into the comments text box. Comments are limited to 65,535 characters. If an entered comment is larger than the maximum amount of characters allowed, the comment displays as 'empty.'

After a comment is added, it is added to the Comments section within 15 seconds and includes the following information:

- The username of the user who entered the comment
- The comment text

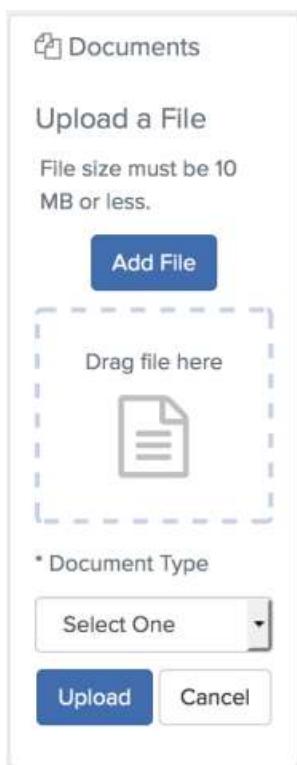
- The date and time when the comment was added

A delete action is also available for each comment via a delete icon (  ). This icon displays next to the comment and is used to delete the comment. A Delete Comment modal displays to confirm the deletion when the icon is selected.



## File Attachments

You can add file attachments to any page within the transaction by clicking the Add File button in the Documents section and selecting a file or by dragging a file from your computer into the 'Drag file here' box. You can attach a file in any file format.



The image shows a file upload interface. It includes a title "Documents", a sub-section "Upload a File", and instructions "File size must be 10 MB or less.". A large dashed rectangular area is labeled "Drag file here" with a paper icon. Below this is a dropdown menu labeled "Select One". At the bottom are two buttons: "Upload" (highlighted in blue) and "Cancel".

If the file you are uploading is large, a progress bar displays to let you know the progress of the file upload.

Documents

Upload a File

File size must be 10 MB or less.

**Add File**

Drag file here

#4A.pdf

\* Document Type

State Suppleme<sup>n</sup>

**Upload** **Cancel**

This screenshot shows a file upload interface. At the top, there's a header with a document icon and the word 'Documents'. Below it, a section titled 'Upload a File' with a note about file size. A large dashed rectangular area is provided for dragging files, with a placeholder 'Drag file here' and a paper icon. A file named '#4A.pdf' is shown within this area, accompanied by a progress bar that is mostly blue. Below this, a field labeled '\* Document Type' has a dropdown menu open, showing 'State Suppleme' as the selected option. At the bottom, there are two buttons: a blue 'Upload' button and a white 'Cancel' button.

After you add a file to the Work Item Assistant, you must select a document type before you click Upload.

Documents

Upload a File

File size must be 10 MB or less.

Add File

Drag file here

#4A.pdf

\* Document Type

Select One

Upload Cancel

The list of available document types can be configured by line of business. If you do not select a document type, the file will not upload and the "Please add a file and select file type" message displays.

- You can only attach one file at a time. If you select a file after you've already put one into the upload queue, it will override the original and the latest one will be uploaded. Click the Cancel button to remove the file from the upload queue.
- If the attached file is larger than the configured maximum size (10MB by default), the file is not saved and an error message displays to the user (e.g., File Attachments must be no larger than 10MB).
- If a file upload fails for a reason other than file size, a generic error message displays (e.g., Your file cannot be uploaded. Please contact your system administrator).

WC\_App\_45

Other, 45 kb

Great Salesman

05-07-2015 16:05:13

After a file is uploaded, it is added to the Documents section and includes the following information:

- The file name
- The document type
- The username of the user who uploaded the file
- The file size
- The date and time when the file was uploaded

#### Note

Uploaded PDF forms (such as via [Turnstile](#)) are automatically added to the Documents section when the work item is created. The document type for these files default to ACORD Form.

The following actions are also available for each uploaded file:

- View the file - The file name is a hyperlink. Click the file name to open the file attachment.
- Edit the file name - An edit icon (  ) displays next to the file name. Click this icon to change the original file name. A File Name modal displays to enter and save the new file name.



A screenshot of a modal window titled "File Name". It contains a text input field labeled "Enter file name" with the placeholder "Enter file name". Below the input field are two buttons: "Save" (blue) and "Cancel" (white).

- Delete the file - A trash can icon (  ) displays next to the file name. This icon displays based on your user permissions (currently, this is only available for users that belong to a group with underwriter permissions). A Delete File modal displays to confirm the deletion.



A screenshot of a modal window titled "Delete File". It contains the text "Are you sure you want to delete this file?". Below the text are two buttons: "Yes" (blue) and "No" (white).

## Other Active Users

If you are in a work item that another user is currently viewing, Work Item Assistant lets you know of the other user's presence.

When other users have the work item open, the username of the other active user displays in the Users section and the following displays at the top of the page: This screen is read-only and cannot be updated. It is locked by another user. Even though the work item is locked by another user, you can still use the Work Item Assistant.



# Secure Fields

Securing sensitive customer data, such as social security, tax ID and driver's license numbers, is critical in any industry, but particularly in insurance. To provide an extra layer of protection to this type of information, we provide the ability to configure which fields contain sensitive data and how to mask them.

Data entry and date fields can be designated as secured fields. When a field is designated as secured, the field is masked with asterisks (\*). If the field has a format mask applied, the formatting (e.g., dashes) also displays.

The following masking options are available for configuration:

- Full masked value (e.g., \*\*\*-\*\*-\*\*\*\*)
- Mask with a prefix (e.g., 23\*-\*\*-\*\*\*\*)
- Mask with a suffix (e.g., \*\*\*-\*\*-\*\*23)

Refer to the [Transaction Definition/Page Library](#) and [Field Security](#) topics in the Developer Guide section for more information on how to configure secure fields.

When you enter data into a secured field, the full value displays and normal validation rules apply. When you click or tab off of the field, field masking is applied and the value no longer displays.

Tax ID      \*\*-\*\*\*\*\*

If you remain on the page, you can click back into the field to view the entire field value and make any changes.

Tax ID      12-3333333

If you click Continue or exit the work item or account, the field is masked and an edit icon displays next to the field.

Tax ID      \*\*-\*\*\*\*\* 

- If you do not have permission to view the secure field, the field value is cleared when the edit icon is clicked. You must re-enter a value into the field; the original value is lost.

Tax ID      | - \_\_\_\_\_

- If you have permission to view the secure field, you can either click or tab on to the field to view and edit the value.

Tax ID      12-3333333

## Uploaded and Endorsed Work Items

Secured field values are not masked on the initial page display for uploaded or endorsed work items. The field is only masked after you click Continue or exit the item.

If a revert option displays for the field, it will only display on initial display. The revert icon will not display after the field is masked.

When a secure field is changed for an endorsement, the changed value does not display on the policy summary page.

Field	Original Value	Current Value	Action Taken
Agency Information.Address Line 2	2nd Floor	2nd Floor DS	Changed
Applicant Information.Tax Id	This value is protected from view		Changed

## Read Only Fields

Secured read only fields always display as masked.

Tax ID      \*\*\*-\*\*-\*\*\*\*

If you have permission to view the secure field, a Show link displays to view the masked value.

Tax ID      \*\*\*-\*\*-\*\*\*\* [Show](#)

# Timeline

Each work item has the option to view the history of the work item, even when the work item is in a locked status. This feature, called Timeline, captures and displays events within the timeline of the work item, as well as who made changes to the work item, rules that were broken and any status changes made over the lifetime of the work item.

The History hyperlink on individual transaction pages gives you access to this feature. This hyperlink, by default, displays as soon you create a work item either manually or via an upload feature, and remains available on each subsequent transaction page for the entire lifetime of the work item.

By default, all users within the administrator, agent and underwriter userroles have permission to access the Timeline. Additional security considerations are available surrounding rule triggered events since underwriters may not want agents to see underwriter-specific rules. Any security or permissions needed for the Timeline feature is determined during project implementation.

## Timeline Events

The events that display within Timeline are listed chronologically on the Work Item History modal when you click the History hyperlink. The order is based on the timestamp of the event. Events are also numbered in order, starting at 1. The timestamp (date and time when the event occurred) also displays under the event title. The earliest event (by default, Work Item Created) displays first at the top of the list.

### Work Item Created

The Work Item Created event always displays in work item history and only displays once. This event is automatically created when you manually create a work item, upload a work item (e.g., via Turnstile) or copy an existing work item. This event is selected by default when you select to view the work item history.

#### Example

The following displays when the work item is created manually:

The screenshot shows a modal window titled "Work Item History". Inside, there's a section titled "Event Summary" with the following details:  
1 Work Item Created  
09-09-2014 08:19:16  
The work item was created  
When: 09-09-2014 08:19:16  
By: Jami Delia

The following displays when the work item is uploaded:

The screenshot shows a modal window titled "Work Item History". Inside, there's a section titled "Event Summary" with the following details:  
1 Work Item Created  
09-09-2014 03:42:49  
About this Upload  
When: 09-09-2014 03:42:49  
By: Great Salesman

Below this, there's a table with columns "System", "Version", and "Vendor":  
System: Agencyport Software Turnstile  
Version: 1.0  
Vendor: Agencyport Software

At the bottom, it says "The work item was created".

## Rule Triggered

The Rule Triggered event displays when a user clicks either Save, Save and Edit, Add (in a roster scenario), Delete (in a roster scenario) or Continue on a transaction page and the work item's XML updates as a result. Only one rule event is created per processing action, even if multiple rules are triggered.

### Example

The following is an example of a rule triggering event:

The screenshot shows a 'Work Item History' timeline. On the left, there is a list of events: '1 Work Item Created' (09-05-2014 09:02:50), '2 Rule Triggered' (09-05-2014 09:13:53), and '3 Rule Triggered' (09-05-2014 09:14:02). The 'Event Summary' tab is selected. It contains the following details:  
A rule was triggered by the values in this work item at the time  
When: 09-05-2014 09:13:53  
By: R G  
A table showing the rule message and severity:  

Page	Rule Message	Severity
Policy Information	Only MA,RI,IL, MD and FL are currently supported. Please select a supported state.	error

## Service Call Made

The Service Call Made event displays at any point after a work item creation event occurs. Timeline displays a service call made event when a user completes an action that causes a server call to be made.

### Status Changed

The Status Changed event displays at any point after the work item created event occurs. By default, Timeline displays a status changed event when a user completes an action that causes the work item status to change.

### Example

The following is an example of a status change event:

The screenshot shows a 'Work Item History' timeline. On the left, there is a list of events: '1 Work Item Created' (09-09-2014 10:24:45) and '2 Status Changed' (09-09-2014 10:26:43). The 'Event Summary' tab is selected. It contains the following details:  
Status Changed by Great Salesman 09-09-2014 10:26:43  
A table showing previous and new status:  

Previous Status	New Status
INPROGRESS	REFER

## Work Item Modified

The Work Item Modified event displays at any point after the first status change event occurs. This cuts down on data entry "noise" for those events that report on the agent's data entry before they submit a work item. Timeline displays this event after a user opens a transaction page, adds new data, changes existing data or deletes existing data and then processes the page. Even though the user may change multiple fields on a page, only one event is created when the user processes the page.

## Event Details

Each event includes details pertaining to the event, and displays when you select the event on the Work Item History modal. The Event Summary tab is the first tab in event details. This tab contains different information for each type of event. For all event types, this tab displays three key items: the event description, the time the event occurred and the user ID of the user who generated the event.

- The event description is located at the top of the Event Summary tab. Only one event description displays per event type and is followed by the name of the user who generated the event.
- The timestamp of when the event took place displays next the event description and user name, and displays in mm-dd-yyyy hh:mm:ss format (e.g., 03-01-2014 12:33:55).
  - For the Work Item Created event, the timestamp is the time the user created the work item.
  - For the Rule Triggered event, the timestamp is the time the user processed the page, which triggered the rule(s).
  - For the Service Call Made event, the timestamp is the time the user completed the action that triggered the service call.
  - For the Status Changed event, the timestamp is the time the user completed the action that triggered the status change.
  - For the Work Item Modified event, the timestamp is the time the user processed the page on which data was modified.

Below the event description, username and timestamp displays the data details of each event. This section contains additional information about the event.

### Work Item Created

For a manually created work item, the event detail section only includes the event summary, which includes when the work item was created and who created the work item.

### Rule Triggered

When a rule is triggered, the event detail section only displays the event summary, which includes the event description, when the rule triggered occurred and the user who triggered the event. Below this information, additional data details also display to include the page where the rule was triggered, the rule message that displayed to the user and the severity of the rule.

### Service Call Made

Service call made events are unique to each carrier and are determined during project implementation. If a service call does not change any data within the work item, the event detail section only includes the event summary, which includes the event description, when the event occurred and the user who triggered the event. If there are any data changes to the work item, a What Changed tab also displays in the detail section and includes the data that the service call changed.

### Status Changed

Status changed events are unique to each carrier and are determined during project implementation. The detail section for a status change event varies based on the type of status change. The following lists the available details that can display for each type of status change:

When the first status change occurs for the work item, the event details include an Event Summary and Snapshot tab. These tabs also display when any subsequent changes are made to the work item, but no work item data is changed since the previous status change.

- The Event Summary tab includes the event description, when the event occurred, the user who created the event and details on the previous status (the status before the event) and the new status (the status after the event).
- The Snapshot tab includes a list all of the data in the work item at the time of the status change event. This information displays in the order it appears in the actual work item (by page and field). A page name (e.g., Agency/Application Information) and fieldset name (e.g., Agency Information) always displays.

When a subsequent status change occurs and the work item data has changed since the previous status change, an Event Summary, What Changed and Snapshot tab displays.

- The Event Summary and Snapshot tabs display the same information as stated in the first status change scenario.
- The What Changed tab details what changed in the work item since the previous status change. This includes a comparison of the data snapshot from the previous status change event to the current status change event. Any differences display on this tab and can include the field name, initial value (value before the event) and final value (value after the event).

### Work Item Modified

The work item modified detail section includes the Event Summary and What Changed tabs. The What Changed tab lists changes made by page and field. This includes the field, initial value (value of the field before the event) and final value (value of the field after the event).

# My Defaults

The My Defaults feature in AgencyPortal allows you to reuse transaction information so you don't have to re-key the same information every time you create similar types of transactions. This can save you a lot of time if you frequently write similar types of policies (e.g., common coverages, locations).

## Note

The My Defaults feature is configured at the transaction level for a line of business. Out of the box, this feature is turned on for the Workers Comp and Commercial Auto lines of businesses, but can be configured to use for any LOB. Refer to the [My Defaults](#) topic in the Developer Guide section for information on how to set up and configure this feature.

## Saving Defaults

You can save available field values for a newly created or an existing transaction that is set up to use the My Defaults feature.

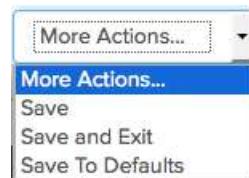
## Note

Click the Available Fields option from the My Defaults drop-down to view all fields available to use with the My Defaults feature.

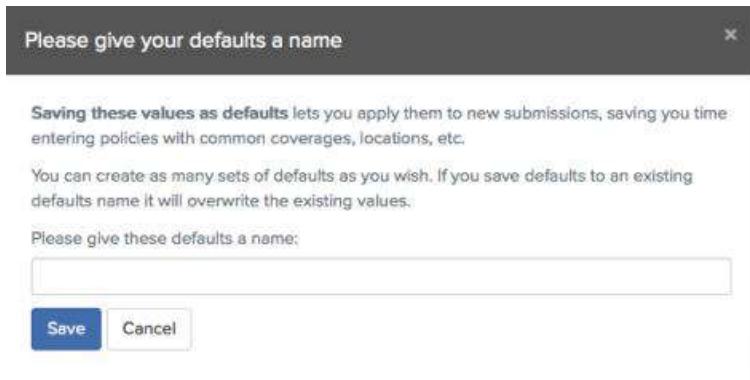
Fields for which Default Values Can Be Set	
Page	Field
Agency/Applicant Information	State
Agency/Applicant Information	Tax Id Type
Agency/Applicant Information	Legal Entity
Agency/Applicant Information	Billing Method
Agency/Applicant Information	Direct Bill Pay Plan
Agency/Applicant Information	Auditable?
Agency/Applicant Information	State
Locations	State
Locations	Located
Rating Classifications	Location
Rating Classifications	Class Code
Questions & TextArea	Billing Method
General Information	Business in Trust

Available fields are defined as corrections in the TDF. Refer to the [My Defaults](#) topic in the Developer Guide section for more information.

After validated fields are entered on the first transaction page and you click Save, the Save to Defaults option displays in the More Actions... drop-down at the bottom of every page within the transaction workflow. This allows you to save entered field information to a default.



The Please give your defaults a name modal displays to enter a name for the default and provides additional information about the feature.



A confirmation window displays after you click Save.



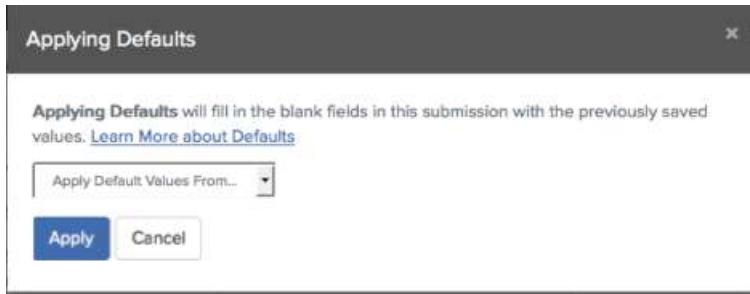
The default is now available for selection from the My Defaults drop-down.

## Applying and Managing Defaults

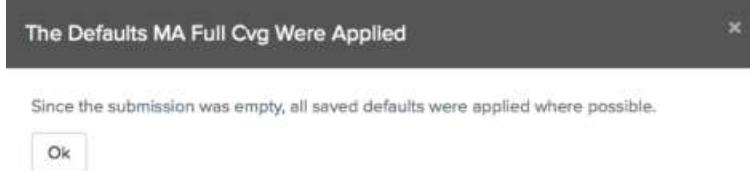
The **★ My Defaults** drop-down displays in the upper right corner of the transaction page when the My Defaults feature is turned on for the transaction. This drop-down displays options to apply defaults to the work item, manage existing defaults, view a list of available fields and refer to the My Defaults help.

### Apply to this Submission

Select the Apply to this Submission option to pre-fill fields with data saved in an already created default. Default values are only pre-filled for new transactions or transactions that do not already have data in the available fields. The Applying Defaults modal displays to select a default.



After you select a default and click Apply, a confirmation displays to let you know changes were made.



### Manage

Select the Manage option to rename or delete saved defaults. The Saved Defaults modal displays.

Saved Defaults:		
	Rename	Delete
CT Full Cvg	<a href="#">Rename</a>	<a href="#">Delete</a>
full app	<a href="#">Rename</a>	<a href="#">Delete</a>
MA Full Cvg	<a href="#">Rename</a>	<a href="#">Delete</a>
test 100	<a href="#">Rename</a>	<a href="#">Delete</a>

[Cancel](#)

Click the corresponding hyperlink to rename or delete the default. A confirmation displays after you complete the action.

## Help with Defaults

Click the Help option from the My Defaults drop-down anywhere within the transaction if you feel like you need a quick refresher on how this feature works. The Help with Defaults modal displays.

Help with Defaults	
<p>If you frequently write similar types of policies, using defaults can save you considerable data entry time. Once you have got the submission where you want it (i.e. you've added some coverages, answered underwriting questions, etc) use the "save to defaults" action in the drop down at the bottom of the page to save a copy (and some keystrokes!) for later.</p> <p>Once you've created a set of defaults, next time you start a new submission from scratch, choose "Apply to this Submission" from the "My Defaults" menu at the top of the screen. Blank fields in the new submission will be filled in with your chosen defaults.</p> <p>When applying default values to submissions in which fields have already been filled out, not all fields will be replaced.</p> <p>The defaults you setup belong only to you, other users can not see or change them.</p>	

# Creating a Work Item

You can create a work item in AgencyPortal 5.1 manually or by using the Turnstile integration upload feature (refer to the Turnstile Integration section of the [Upload](#) topic for more information on this feature).

## Manual Creation

	Select an LOB thumbnail on the Home page and select Quick Quote or Full Application.
	Select one of the LOB options under Quick Quote or Policy Application from the Get Started drop-down menu on the Home page.
	Click the Add New Work Item card on the card view of the My Work page.

<p>Get Started With ▾</p> <ul style="list-style-type: none"> <li>Prefill from an ACORD Form</li> <li>Business Owners Application</li> <li>Business Owners Quick Quote</li> <li>Commercial Auto</li> <li>Commercial Auto QQ</li> <li>Homeowners</li> <li>Homeowners QQ</li> <li>Personal Auto</li> <li>Personal Auto QQ</li> <li>Testing Application</li> <li>Testing Application - AUTOB</li> <li>Workers Comp</li> <li>Workers Comp QQ</li> </ul>	<p>Click an LOB option from the Get Started With drop-down on the classic view of the My Work page.</p>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------

#### Upload Creation

 <p>My Reports    Get Started ▾</p> <p>Prefill from an ACORD Form <span style="border: 1px solid red; padding: 2px;">2</span></p> <ul style="list-style-type: none"> <li>Quick Quote</li> <li>Business Owners Application</li> <li>Commercial Auto</li> <li>Homeowners</li> <li>Personal Auto</li> <li>Workers Comp</li> </ul> <hr/> <ul style="list-style-type: none"> <li>Policy Application</li> <li>Business Owners Application</li> <li>Commercial Auto</li> <li>Farmowners</li> <li>Homeowners</li> <li>Management Liability</li> <li>Personal Auto</li> <li>Workers Comp</li> </ul>	<p>Select the Prefill from an ACORD Form option from the Get Started drop-down on the Home page and upload an ACORD form.</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------

<p><b>Get Started With ▾</b></p> <p>Prefill from an ACORD Form</p> <ul style="list-style-type: none"> <li>Business Owners Application</li> <li>Business Owners Quick Quote</li> <li>Commercial Auto</li> <li>Commercial Auto QQ</li> <li>Homeowners</li> <li>Homeowners QQ</li> <li>Personal Auto</li> <li>Personal Auto QQ</li> <li>Testing Application</li> <li>Testing Application - AUTOB</li> <li>Workers Comp</li> <li>Workers Comp QQ</li> </ul>	<p>Select the Prefill from an ACORD Form option from the Get Started With drop-down on the the classic view of the My Work page and upload an ACORD form.</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------

Work items can be created individually or in bulk (such as using one of the upload features). Work items can also be created within an account. Refer to the Adding a Work Item within an Account section of the [Managing Accounts](#) topic for more information.

After you select one of the above options to create a work item, the first transaction data entry page displays. Transaction pages include all fields defined during project implementation. Required fields display with an asterisk (\*). You must complete these fields to continue. Refer to the [Page Types](#) topic for more information on the fields and elements that display.

If your application is configured to enforce account creation during the work item creation workflow, the following validation message displays: "You must create an account or select an existing account for this work item before continuing." You must then click the Link to an Account hyperlink at the top of the page and either create a new account or associate with an existing account. Refer to the Link to an Account section in the [Managing Work Items](#) topic for more information.

## Percent Complete

After a work item is created, a percent complete indicator displays on the following pages:

- Individual work item transaction page

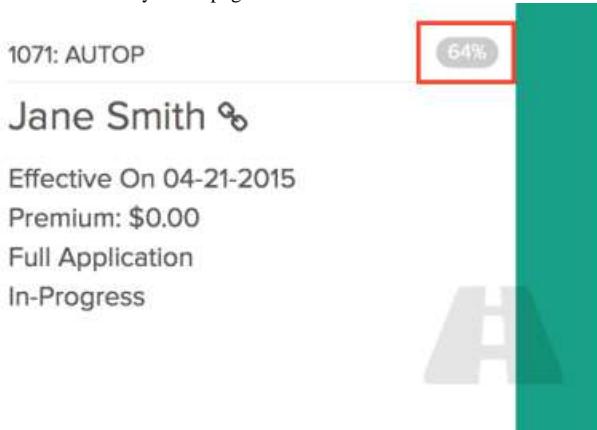
The screenshot shows a work item transaction page. At the top left, there is a progress bar indicating 64% completion, followed by the text "Personal Auto (ID: 1071 - Jane Smith)". To the right of the progress bar is a "Link to an Account" button and a "History" link. On the far right, there is a "Print" icon.

The main content area has a sidebar on the left containing navigation links: "Applicant Information", "Policy Coverages", "Drivers", "Incidents", "Garaging Locations", "Vehicles", "Driver Vehicle Assignments", "General Information" (which is currently selected), "Loss History", "Additional Interests", and "Summary".

The main content area is titled "Questionnaire" and contains a list of questions with radio button options for "Yes" and "No". The questions are:

- \* 1. With the exception of any encumbrances, are any vehicles not solely owned by and registered to the applicant? (radio buttons: Yes, No)
- \* 2. Any car modified/special equipment? (Include customized vans/pickups) (radio buttons: Yes, No)
- \* 3. Any existing damage to vehicle? (Include damaged glass) (radio buttons: Yes, No)
- \* 4. Any other losses incurred? (Not shown in Accident/Conviction area) (radio buttons: Yes, No)
- \* 5. Any car kept at school? (radio buttons: Yes, No)
- \* 6. Any car parked on street? (radio buttons: Yes, No)
- \* 7. Any other auto insurance in household? (Include any provided by employer) (radio buttons: Yes, No)
- \* 8. Any other insurance with this company? (radio buttons: Yes, No)
- \* 9. Any household member in military service? (radio buttons: Yes, No)

- Card view of the My Work page



- Classic view of the My Work page

ID	LOB	Name	Status	Updated	Effective	Premium	Transaction Type	Complete %
1071	Personal Auto	Jane Smith	In-Progress	04-22-2015 05:23:20	04-21-2015	0	Full Application	64

This feature lets you see where you are within a transaction workflow and know the progress of particular work items. The same percentage displays for the work item on all of the pages listed above.

The percent complete that displays is based on the number of pages visited for the particular line of business work item and is consistent throughout the application.

- A page for a manually created work item is considered complete when it has been saved manually (autosave is not considered in to the calculation). If a page is read-only (e.g., the Policy Summary page), it is only factored into the percent complete calculation when it is also saved.
- A page for an uploaded work item is considered completed if no validation errors are detected. The only exception to this rule is the first page, which is always counted as visited.

## Data Pre-fill

For all work item templates, functionality is provided to pre-fill the first location from the mailing address on the applicant information page.

If the work item is created from an account, there is additional functionality to downfill data from the account into the work item and an option to manually select locations and contacts from the account to add to the work item. The reference implementation for this feature is in the Locations and Individuals Included/Excluded roster pages in the Workers Compensation template. Refer to the [Creating an Account](#) topic for more information on leveraging account level data in a work item.

## Using Work Item Assistant

Also included on transaction pages is the work item assistant feature. This feature displays on the right side of the page, and includes the options to:

- Enter a comment for the work item
- Add a file (10MB or less) to the work item

Refer to the [Work Item Assistant](#) topic for more information.

## Navigating through the Transaction Pages

After you complete the information on the page, click one of the following available options:

- Continue to continue on to the next page.
- More Actions..., which includes the option to Save the data entered on the page, Save and Exit (save the data on the page and then exit out of the work item) and Convert to Application (convert a quick quote work item into a full application).

At the end of the transaction, a summary page displays. This page includes data entered within the transaction, as well as premium information. Refer to the Summary Pages section of the [Page Types](#) topic for more information.

The screenshot shows a summary page for a "Personal Auto QO (ID: 1184 - John Doe)" application. At the top, there are links for "Link to an Account" and "History". Below that, there are buttons for "Normal" and "Side-By-Side" view modes, with "Normal" selected. A green box highlights the "In-Progress" status at the top right. The main content area is titled "Summary" and contains a message: "If you have completed the application, click "Save and Exit". Your submission will be saved in your Work in Progress." Below this, three premium amounts are listed: \$1838 (Monthly), \$2468 (Bi-Annual), and \$1390 (Payment In Full). Further down, there are three sections: "Producer Code" (Agency Customer ID), "John Doe" (Co-Applicant, Policy Term: 04-27-2015 to 04-27-2016), and "Documents" (View Acord 90).

# Managing Work Items

After a work item is created, you can [modify](#), [move](#), [copy](#) or [delete](#) the work item.

You can find existing work items on the My Work page. By default, work items display by last updated date. Only the work items related to the user group in which you belong display; you cannot see work items created by a user in a different user group. Use the search and filter feature to search for and select a particular work item. Refer to the [Search and Filter](#) topic for information on how to search for a work item.

If your project is set up to use Account Management, you can also find existing work items by account. Search for and select an account from the My Accounts page, then search for and select the work item from the list of associated work items on the account information page. Refer to the [Search and Filter](#) topic for information on how to search for a work item.

## Modifying a Work Item

Opening a work item to view or modify its contents can be done by double clicking the work item or clicking the Open button on the work item. You can also open a work item from an account information page if the work item is associated with the account. After you open the work item, the first transaction page displays.

The screenshot shows the 'Agency Information' section of a work item. On the left, a sidebar lists 'Applicant Information' with links for Policy Information, Locations, State Rating Factors, Rating Classifications, and Summary. The main area contains fields for Agency Name ('Test Agency'), Address Line 1 ('55 Main St'), Address Line 2 (empty), City ('Boston'), State ('Massachusetts'), Zip Code ('02210'), CS Representative Name (empty), Producer Phone (empty), Mobile Phone (empty), and Producer Fax (empty). To the right, there are two panels: 'Comments' showing a message from 'Great Salesman' about follow-up, and 'Documents' with a section for uploading files.

The percent complete indicator in the upper right corner displays to let you know how far along you are within the transaction workflow (based on the number of pages visited). Review each page and make any needed changes.

- If all transaction pages have been previously visited (page title displays in blue), you can jump around throughout the transaction page menu.
- If a page has not been visited (page title displays in grey), you must complete the required fields on the page before proceeding to the next page.
- If a required field has not been entered or incorrect data is entered in the field, a validation displays and you must correct the field before proceeding.

## Transaction Validation

If a work item was previously uploaded into the application or was a quick quote converted into a full application, an alert message may display at the top of the transaction page when information is missing or invalid.

① Agency Information.Agency Name can't exceed 40 characters in length  
 Applicant Information.Date Business Started must be a valid date entered in MM-dd-yyyy format.  
 Applicant Information.SIC Code value must match one of the drop down values.  
 Applicant Information.Tax ID must be a valid FEIN with a format of nn-nnnnnnnn

AgencyPortal includes validation messaging to alert you when work items are missing information. When you access a work item with missing information, an alert message displays at the top of each data entry transaction page to let you know what data is missing and which fields you should review.

Roster pages display validation messaging a little differently. Instead of the validation message displaying at the top of the roster page, each roster entry that has an issue displays in red. The actual alert message displays when you click the roster item to display the individual roster entry page.

## Rating Information

[Help with this List](#)

For Location	Class Code	Full Time Empl	Part Time Empl	Est. Annual Exposure	
162 Main Street Peabody, MA 01960	9521	4	6	10000	<a href="#">Edit</a> <a href="#">Delete</a>
162 Main Street Peabody, MA 01960	9521	2	3	5000	<a href="#">Edit</a> <a href="#">Delete</a>
11 Summer St Providence, RI 01312	0913	3	2	50000	<a href="#">Edit</a> <a href="#">Delete</a>

[Continue](#) [Add New](#) [More Actions...](#) ▾

## Note

Rating classification roster entries do not require intervention if both the class code and description in an uploaded PDF matches the values in AgencyPortal. If the code is the same and the description is different, then you must re-select a value.

While reviewing field information, the following field validations may display:

- A Select One drop-down option may display when there is no exact match found between a field value in an uploaded file and a required selectlist or filterlist field in AgencyPortal. This indicates you must select a value from a pre-set list of the next closest value.

## Example

① Symbols.Specified Causes of Loss Symbol is required

\* Denotes Required Fields

### Symbols

* Liability Symbol	7
* Medical Payments Symbol	3
* Uninsured/Underinsured Symbol	7
* Comprehensive Symbol	7
* Specified Causes of Loss Symbol	Select One

- The field may display in red when an optional field requires review after a work item is created via an upload. This indicates you should review the field to make sure the correct option is selected.

Applicant Information,Legal Entity value must match one of the drop down values  
Applicant Information,Credit Bureau Name value must match one of the drop down values

Legal Entity	<input style="width: 100%; border: none; border-bottom: 1px solid #ccc; padding: 2px; margin-bottom: 5px;" type="text" value="Select One"/> <span style="color: red; font-size: small;">*</span>
Credit Bureau Name	<input style="width: 100%; border: none; border-bottom: 1px solid #ccc; padding: 2px; margin-bottom: 5px;" type="text" value="Select One"/> <span style="color: red; font-size: small;">*</span>

## Saving Changes

After making changes to a work item, you must save any changes before proceeding to another page or clicking out of the work item. Save options are available at the bottom of every transaction page.

- Continue - The Continue button saves your changes and takes you directly to the next transaction page.
- Save - The Save option on the More Actions... drop-down allows you to save your changes before jumping to another page.
- Save and Exit - The Save and Exit option allows you to save your changes and exit the work item.

If you do not save your changes, the following modal displays.



Click No to return to the page and save any changes; click Yes to disregard your changes and leave the page without saving.

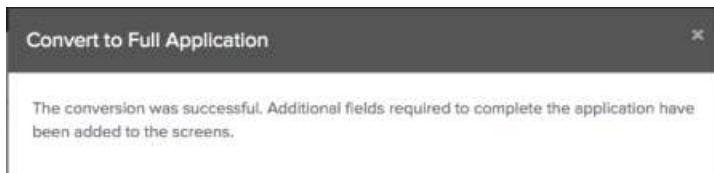
## Converting Quick Quote to Full Application

If you decide to convert a quick quote transaction into a full application, select the Convert to Application option on the More Actions... drop-down. The following confirmation modal displays:



Click the Yes button to confirm the conversion or No to return to the transaction page.

When you click Yes, a second confirmation modal displays to indicate the conversion was successful.



## Linking a Work Item to an Account

If a work item is not assigned to an account, you can associate it with a new account or to an already existing account. The following indicators display to notify you that an account is not linked to an account:

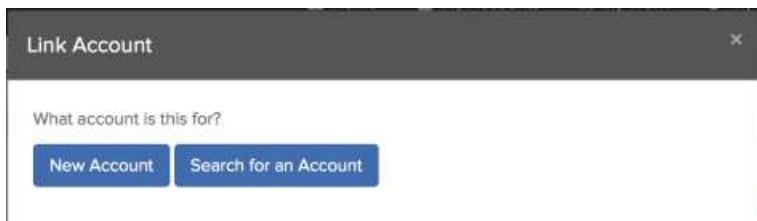
- A link icon ( ) displays for the work item on the My Work page.
- A Link to an Account hyperlink ([Link to an Account](#) ) displays at the top of the work item transaction page.

Placing the cursor over either of these indicators displays the following message: "This work item is not assigned to an account."

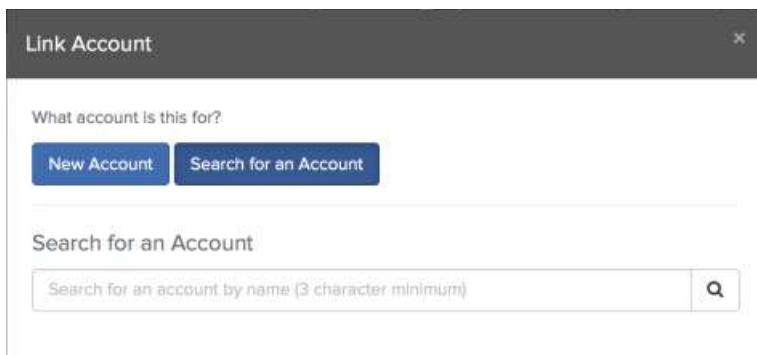
You can link an unassigned account from one of the following options:

- Select the work item on the My Work page and select the Link option from the Other Actions drop-down.
- Open up a work item and click the Link to an Account hyperlink.

The Link Account modal displays to create a new account or search for an existing account.



- Click the New Account option to create a new account. The account transaction page displays to create the account. Refer to [Creating an Account](#) for more information.
- Click Search for an Account to search for an existing account. A Search for Account search box displays.

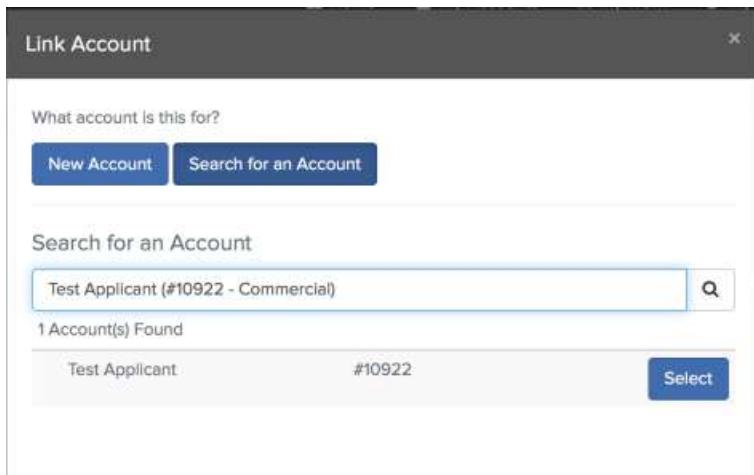


Search for an account by name. Enter an account name into the search box. A list of relevant accounts automatically displays when you enter at least three characters (if you do not enter at least three characters and click the magnifying glass icon, the following message displays: "Please provide at least 3 characters.").

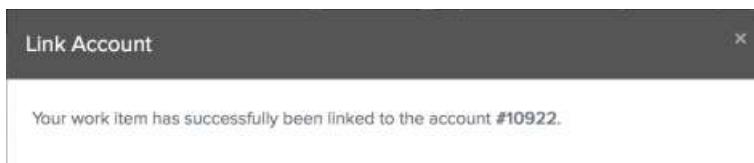
The search feature uses auto suggest when three characters are entered into the search box.

- Auto suggest returns a maximum of four matches. If there are more than four matches, XX others (where XX is the number of additional items that meet the search criteria) displays to link additional results. Click the XX others link or the Select button to display all results.
- The XX others link (where XX is the number of additional items that meet the search criteria) is informational only. If selected, the auto suggest results no longer display and a list of all results display in a scrollable list and are available for selection.

Refer to the [Search and Filter](#) topic for more information on how the search feature works within AgencyPortal.



After you select an account, a confirmation modal displays.



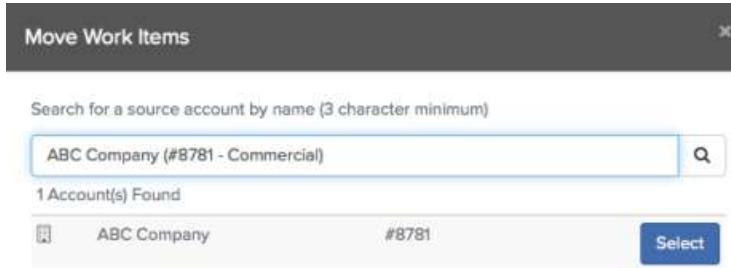
## Moving a Work Item

If a work item is assigned to an account, you can move it to associate it with another account. From the Other Actions drop-down that displays for the work item, select the Move option.

The Move Work Items modal displays to search for an account to move the work item to.



Enter an account name into the search box. A list of relevant accounts automatically displays when you enter at least three characters (if you do not enter at least three characters and click the magnifying glass icon, the following message displays: "Please provide at least 3 characters.").



The search feature uses auto suggest when three characters are entered into the search box.

- Auto suggest returns a maximum of four matches. If there are more than four matches, XX others (where XX is the number of additional items that meet the search criteria) displays to link additional results. Click the XX others link or the Select button to display all results.

- The XX others link (where XX is the number of additional items that meet the search criteria) is informational only. If selected, the auto suggest results no longer display and a list of all results display in a scrollable list and are available for selection.

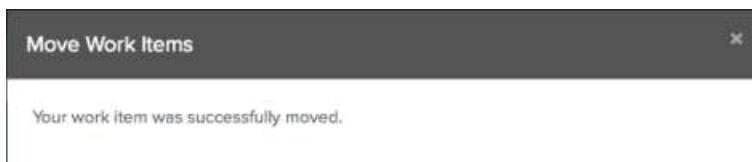
Refer to the [Search and Filter](#) topic for more information on how the search feature works within AgencyPortal.

After you select an account, a confirmation modal displays.



Click the Move button to confirm the move or click Close to cancel the action.

A second confirmation modal displays, indicating the work item was successfully moved.



## Copying a Work Item

The Copy option on the Other Actions drop-down allows you to copy an existing work item in its entirety to create a new work item with a different work item ID. This action can be performed from the My Work or account information page.

As soon as you click the Copy option, the first transaction page for the new work item displays with the information from the initial work item prefilled. You can then review the information on all of the transaction pages and make any necessary changes.

## Deleting a Work Item

To permanently delete a work item from the system, select the Delete option from the Other Actions drop-down on either the My Work or account information page. A confirmation modal displays to confirm the deletion.



Click the Delete button to confirm the deletion or close out of the modal to cancel the action.

# Accounts

The Account Management feature in AgencyPortal is an accounts-based approach to managing policies. When carriers enable this feature within their AgencyPortal, common customer information is captured at the account level and pre-fills into each line of business that is eligible for quoting. Instead of navigating through the portal one work item at a time, accounts-minded users can manage multiple work items across each customer account. Account Management provides the ability for agents and underwriters to view all in-force, quotes and historical policies for all lines of business associated with a specific account.

The My Accounts tab in the navigation menu provides access to view and manage accounts. The My Accounts page displays all available accounts by most recently updated. The following is a list of events that trigger an update to the account last updated date:

- create account
- edit account
- create work item (manual or upload)
- move work item
- change work item status
- merge accounts (both accounts show up as last updated)

You can search, filter and sort accounts to display different criteria. Refer to the [Search and Filter](#) topic for more information.

## Page View

There are two ways you can view the My Accounts page 1) card view 2) classic view. The functionality on both pages works in a similar fashion; however, you can only create accounts from the card view version of the page. The information on both versions of the page are similar (based on the parameters established during project implementation), but displays in a different format.

### Card View

Viewing accounts in the card view displays account information on a card. By default, the account name, type, ID and state/city/zip display on the card. The card view version of this page is the default view of accounts in the AgencyPortal application.

44 Account(s) Type Sort By More Options My Filters

Clear All Sort By: Updated

Search for an account by name (3 character minimum)

Add New Account

Account Name	Type	Address
Yummy Food, Inc	Commercial	1422 161 Atlantic Ave Boston MA 02210
ABC Industries, Inc	Commercial	1420 55 Main St Marlborough MA 01752
Jane Doe	Personal	1421 22 Summer St Boston MA 02210
Penny Smith	Personal	1419 P.O. Box 123 Boston MA 02210
Pradeep Namepally	Personal	1002 1 main st Boston MA 23423-4234

Click the button to toggle to the classic view version of the page.

Classic View

Viewing accounts in the classic view displays account information in a traditional list that is sortable by columns. The columns that display on this version of the page are determined during project implementation. By default, the account ID, type, name, last updated date and address display. All columns, except Address, are clickable to sort results.

ID	Type	Name	Updated	Address
1422	Commercial	Yummy Food, Inc	2015-05-14 09:52:47	161 Atlantic Ave Boston MA 02210
1420	Commercial	ABC Industries, Inc	2015-05-14 09:51:40	55 Main St Marlborough MA 01752
1421	Personal	Jane Doe	2015-05-14 09:51:27	22 Summer St Boston MA 02210
1419	Personal	Penny Smith	2015-05-14 09:49:38	P.O. Box 123 Boston MA 02210
1002	Personal	Pradeep Namepally	2015-05-14 09:48:33	1 main st Boston MA 23423-4234
1381	Commercial	Test	2015-05-14 09:48:33	Street City MA 1111
1328	Personal	Betty Boop	2015-05-14 07:29:26	45 swan st. cambridge MA 03333
1326	Commercial	JJ	2015-05-13 14:55:23	1 Center Street Danvers MA 01923

Click the button to toggle to the card view version of the page.

## Account Types

You can [create accounts](#) for either commercial or personal lines of business. Based on the account type selected, the account template displays with the appropriate fields for the account type. These templates integrate common ACORD standard data elements found within each line of business. The amount and type of information required to create an account is configurable during implementation requirements.

## Account Detail View

The account detail view, or account summary page, is the 'home page' of an individual account. This page gives the user a high-level snapshot of important account information. This view also shows all work items created as part of the account. You can view, search, sort and edit all work items belonging to an account, as well as add new work items to an account.

## Work Items and Accounts

The Work tab includes a work list of work items associated with the account. The work list includes the standard work list functionality, which includes searching, sorting, page views, etc. You can add a new work item or modify any existing work items associated with the account from this tab, depending on the role, status and ownership of the work item.

1 Work item(s) LOB ▾ Status ▾ Transaction ▾ Sort By ▾ More Options ▾

Clear All Sort By: Updated

Search for a customer by name (3 character minimum)

1423: WORK 10%

**Jack Johnson**

Effective On 09-25-2012

Premium: \$0.00

Full Application

In-Progress

Add New Work Item

When you create a new work item at the account level, account level data is automatically pre-filled into the new work item. This eliminates the need to re-key data that is shared across multiple work items. After you set up an account, all account level data (e.g., Application Name) is pre-filled into the work item pages each time you create a new work item from the account summary page.

You can edit or reuse the pre-fill data. However, if you edit the pre-fill data inside a work item, the edited information is specific to the individual work item only. If the data within the line of business differs from the account detail, the account detail information is not updated. No other work items inside the account are updated either.

Refer to Adding a Work Item within an Account section in the [Managing Accounts](#) topic for more information.

## Account Fields

The fields that display on the account transaction and roster pages can be set up with field format masking, validations and field help during project implementation.

### Example

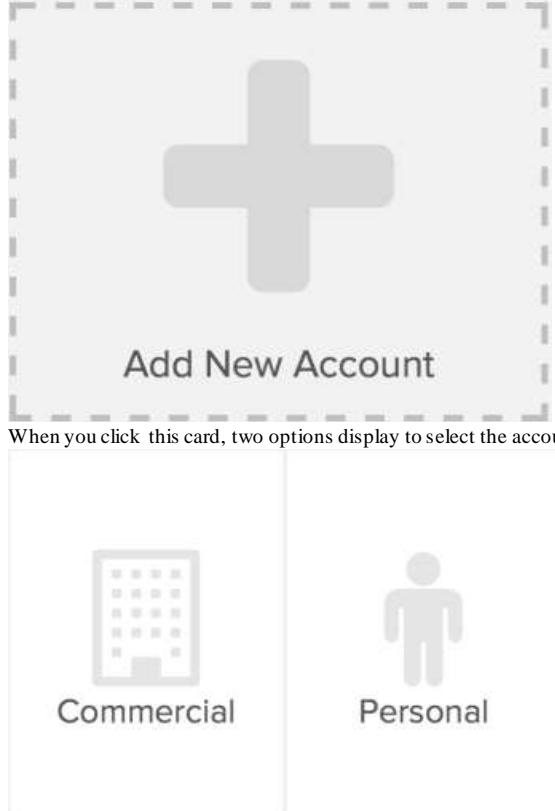
Address Line 1 on the people roster page is set to required. You must enter a value in the Address Line 1 field before adding the roster item. If you do not enter a value, a validation message displays and you cannot continue.

# Creating an Account

You can create new accounts from the card or classic view version of the My Account page. Before you create an account, perform a search to make sure your account is not already created. Refer to the [Search and Filter](#) topic for detailed information on using the search feature.

The following displays to create a new account from the card view version of the My Account page:

- The Add New Account card displays as the first card on the page.



- When you click this card, two options display to select the account type: commercial and personal.

The following displays to create a new account from the classic view of the My Account page:

- Select the Get Started With drop-down.

**Get Started With ▾**

- Two options display to select the account type: Commercial Account and Personal Account.

**Get Started With ▾**

**Commercial**

**Personal**

After you select the appropriate account type option for the account you want to create. The associated account transaction page displays.

\* Denotes Required Fields

**Creating an account allows you to reuse the account information with each new submission.**

### Account Information

\* Account Type  Commercial Personal

\* First Name

\* Last Name

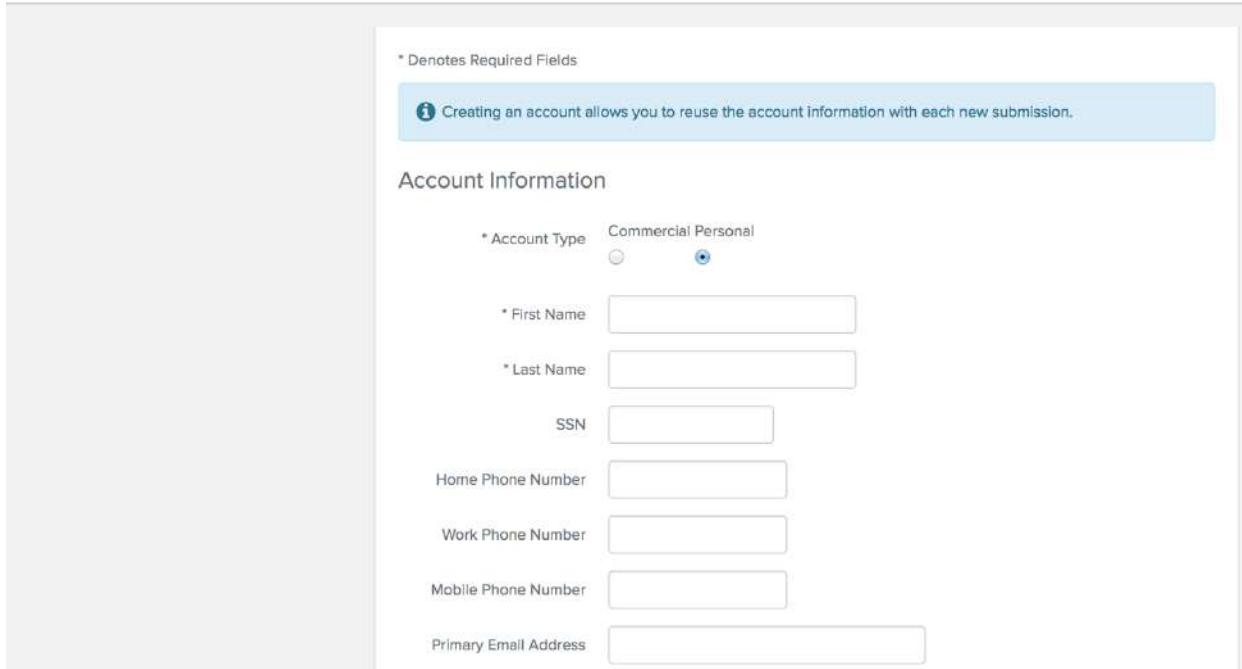
SSN

Home Phone Number

Work Phone Number

Mobile Phone Number

Primary Email Address



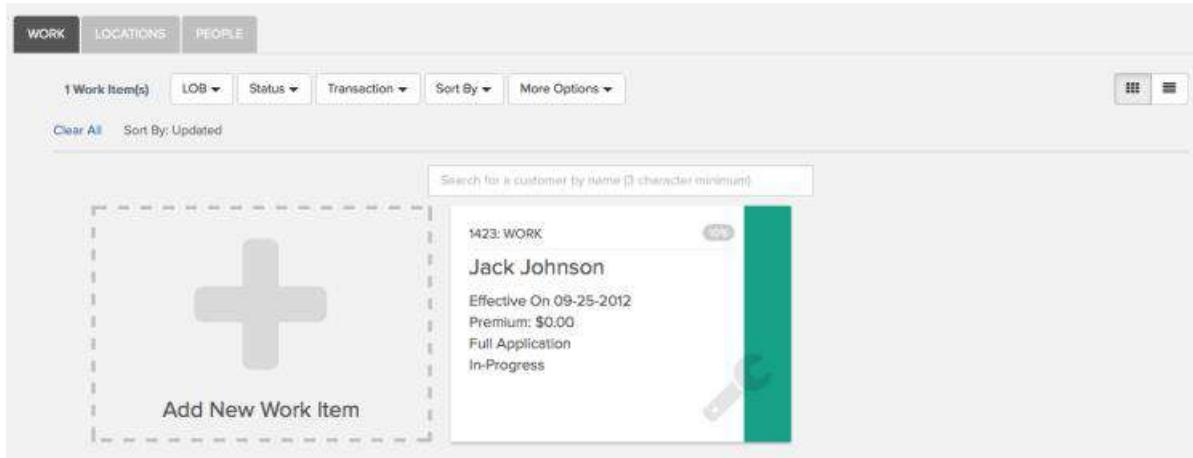
The information and fields that display on the page are based on the account type template determined during project implementation and are associated with the selected account type. The fields that display on the page can differ based on whether the account is commercial or personal. Fields that display an asterisk (\*) next to them indicate the field is required. You must complete all required fields before continuing. If you do not enter information in a required field, the field displays in red and you cannot continue.

## Adding a Work Item within an Account

After an account is created, you can associate existing work items to the account or add a new work item to the account directly from the account summary page.

The Work tab on the account summary page lists any existing associated work items, as well as the option to add any new work items to the account. The options to add a work item to the account display slightly differently on the card and classic view versions of the My work tab.

### Card View



WORK LOCATIONS PEOPLE

1 Work Item(s) LOB ▾ Status ▾ Transaction ▾ Sort By ▾ More Options ▾

Clear All Sort By: Updated

Add New Work Item

Search for a customer by name [ character minimum]

1423: WORK  
Jack Johnson  
Effective On 09-25-2012  
Premium: \$0.00  
Full Application  
In-Progress

An Add New Work Item card displays to add a work item to the account. When you click the card, the following options display:

- Upload Form. Select this option to create a new work item from an ACORD form. The Upload Forms modal displays when you select this option. Refer to [Turnstile Integration](#) for more information on creating a work item from an ACORD form.
- Start a New Work Item. Select this option to manually create a new work item. The option to select a line of business displays when you select this option. Refer to [Creating a Work Item](#) for more information on creating a work item manually.

#### Classic View

The screenshot shows the 'WORK' tab selected in the navigation bar. Below it, there are filters for 'LOB', 'Status', 'Transaction', 'Sort By', and 'More Options'. A search bar at the top right allows searching by customer name. A 'Get Started With' dropdown is visible. The main table lists one work item:

ID	LOB	Name	Status	Updated	Effective	Premium	Transaction Type	Complete %
1423	Workers Comp	Jack Johnson	In-Progress	2015-05-14 09:57:49	09-25-2012	0	Full Application	10

A Get Started With drop-down displays to create a new work item.

The 'Get Started With' dropdown menu lists the following options:

- Prefill from an ACORD Form
- Business Owners Application
- Business Owners Quick Quote
- Commercial Auto
- Commercial Auto QQ
- Homeowners
- Homeowners QQ
- Personal Auto
- Personal Auto QQ
- Testing Application
- Testing Application - AUTOB
- Workers Comp
- Workers Comp QQ

This drop-down lists all available lines of business that are available to create a new work item manually. Select a line of business to begin creating the work item. Refer to [Creating a Work Item](#) for more information on how to create a new work item manually.

There is also an option to upload a work item via an ACORD form. When uploading an ACORD form via the account detail page, the created work item is automatically assigned to the account as long as you continue through the workflow (i.e., process the work item). If you decide to process the PDF at a later time, no account will be tied to the work item. Refer to the [Upload](#) topic for more information on the upload workflow.

When creating a work item manually using either of these create work item options, any relevant account level information from the account is automatically pre-filled into the work item.

#### Down-Filling into a Work Item

When you attach an account to a work item, data entered for the account is down-filled into the work item. This alleviates the need of duplicating data entry. The first location is still pre-filled with the mailing address on the first page of the application.

## Adding Account Level Locations and Contacts

To help reuse account level locations and people, an Add from Account option is included on the Locations and Individuals Included/Excluded rosters in the Workers Compensation template, which can be reused across other templates on applicable pages.

### Locations

The screenshot shows a navigation menu on the left with options: Applicant Information, Policy Information, Locations (which is selected and highlighted in blue), State Rating Factors, Rating Classifications, and Summary. To the right, there is a note about required fields and a text input field for entering locations. Below this is a section titled 'Location Information' with a text input field containing '22 Sleeper St Boston, MA 02210' and buttons for Edit and Delete. At the bottom are Continue, Add From Account, Add New, and More Actions... buttons.

### Individuals Included/Excluded

The screenshot shows a navigation menu on the left with options: Agency/Applicant Information, Policy Information, Locations, State Rating Factors, Rating Classifications, Individuals Included/Excluded (which is selected and highlighted in blue), General Information, and Class Code Questionnaire. To the right, there is a note about including/excluding individuals and a section titled 'Individuals Included/Excluded' with a table header for Location, Class Code, Name, and Inc/Exc. Below this is a message stating 'No Individuals Included/Excluded found for this work item'. At the bottom are Continue, Add From Account, Add New, and More Actions... buttons.

After you create an account, you can add additional locations or people to the account directly from the account summary page. These are available as roster pages. These pages are available from their individual tabs on the account summary page, and allow you to enter and maintain additional locations and/or people associated with the account.

- Click the Locations tab to view locations currently added to the account. After an account is saved, the account address is listed on the location roster. The account address defaults as the first location in the roster and can be edited or deleted. Click the Add Location button to add any additional locations. Locations added at the account level are also available when creating new work items.

- Click the People tab to view people currently added to the account. After an account is created, the company or person name entered on the account is the default account holder and cannot be removed from the account. Click the Add Someone button to add any additional people. People added at the account level are also available when creating new work items.

WORK   LOCATIONS   PEOPLE

### People and Resources

Name	Role(s)	
Yummy Food	Account Holder	<button style="border: none; padding: 0 5px;">Edit</button>
Let's Eat, Inc	Billing Contact	<button style="border: none; padding: 0 5px;">Edit</button> <button style="border: none; padding: 0 5px;">Delete</button>
Jake Rudolph	Driver, Claims Information Contact	<button style="border: none; padding: 0 5px;">Edit</button> <button style="border: none; padding: 0 5px;">Delete</button>

[Add Someone](#)

WORK   LOCATIONS   PEOPLE

### Details

\* Type  Commercial  Personal

Primary Email Address

---

### Address

\* Address Type  Select One   
 Copy account holder address

\* Address Line 1

Address Line 2

\* City

\* State  Select One

\* Zip Code

---

### Roles

Please select at least one role.

Accounting Contact

Additional Insured

Additional Interest

Billing Contact

Claims Information Contact

Inspection Contact

Named Insured

Secondary Contact

[Add](#) [Cancel](#)

A Role(s) field is provided so you can assign one or more roles to the people within the people roster (e.g., additional interest, owner, officer).

# Managing Accounts

After an account is created, you can [modify](#), [merge](#) or [delete](#) the account.

You can find existing accounts on the My Accounts page. By default, accounts display by last updated date. Only the accounts related to the user group in which you belong display; you cannot see accounts created by a user in a different user group. Use the search and filter feature to search for and select a particular account. Refer to the [Search and Filter](#) topic for information on how to search for an account.

## Modifying Account Details

You can easily open an account to view or modify its information by double clicking on the account in the card or classic view version of the My Accounts page or select the account and click the Open button. The account summary page displays.

The screenshot shows the 'Account Information' page for 'Yummy Food, Inc'. At the top, there's a header with icons for edit, merge, and delete. Below the header, the account name 'Yummy Food, Inc' and account number 'Commercial Account # 1422' are displayed. To the right, 'Contact Information' includes '161 Atlantic Ave Boston, MA 02210'. On the far right, 'Created' is listed as '2015-05-14 by Great Salesman' and 'Last Updated' as '2015-05-14 by Great Salesman'. Below this, there are three tabs: 'WORK' (selected), 'LOCATIONS', and 'PEOPLE'. Under the 'WORK' tab, there's a list of work items: '1 Work Item(s)'. The list shows one item: '1423: WORK' for 'Jack Johnson', effective on '09-25-2012', with a premium of '\$0.00', labeled as 'Full Application' and 'In-Progress'. A search bar at the top says 'Search for a customer by name (3 character minimum)'.

From the account summary page, you can select the following options to make any changes to the account:

- Click the icon in the upper right corner to edit the basic account information.

### Note

Fields with an asterisk (\*) are still required when maintaining an existing account. If you do not enter information in a required field, the field displays in red and you cannot continue.

The account type is read-only when editing an account. In addition, the account holder can only be edited; they cannot be removed from the people roster.

- Click the icon to merge the account with another existing account. Refer to [Merging an Account](#) for more information.
- Click the icon to delete the account from the system. Refer to [Deleting an Account](#) for more information.

Below the account information section is three tabs of related account information:

- Work
- Locations
- People

Work

The Work tab lists all work items associated with the account. You can view this list in either the card or classic view by toggling the view from the icons on the right side of the page. Refer to Adding a Work Item within an Account for more information.

## Locations

The Locations tab is a roster of all locations associated with the account.

The screenshot shows a user interface for managing locations. At the top, there are three tabs: WORK, LOCATIONS (which is selected and highlighted in blue), and PEOPLE. Below the tabs, the title "Location Information" is displayed. A table lists three locations:

Location	Actions
15 Essex St Boston, MA 02210	Edit Delete
55 Sleeper St Boston, MA 02210	Edit Delete
27 Boston Post Road Marlborough, MA 01752	Edit Delete

At the bottom left of the table area is a blue button labeled "Add Location".

The following options are provided on this tab:

- Add any new locations to the account.
- Edit any existing locations associated with the account.
- Delete any existing locations associated with the account.

## People

The People tab is a roster of all people associated with the account and their role (e.g., account holder, driver, etc).

The screenshot shows a user interface for managing people. At the top, there are three tabs: WORK, LOCATIONS, and PEOPLE (selected). Below the tabs, the title "People and Resources" is displayed. A table lists three people:

Name	Role(s)	Actions
Yummy Food	Account Holder	Edit
Let's Eat, Inc	Billing Contact	Edit Delete
Jake Rudolph	Driver, Claims Information Contact	Edit Delete

At the bottom left of the table area is a blue button labeled "Add Someone".

The following options are available from this tab:

- Add someone to the account.
- Edit any existing people associated with the account.
- Delete any existing people, except the account holder, from the account.

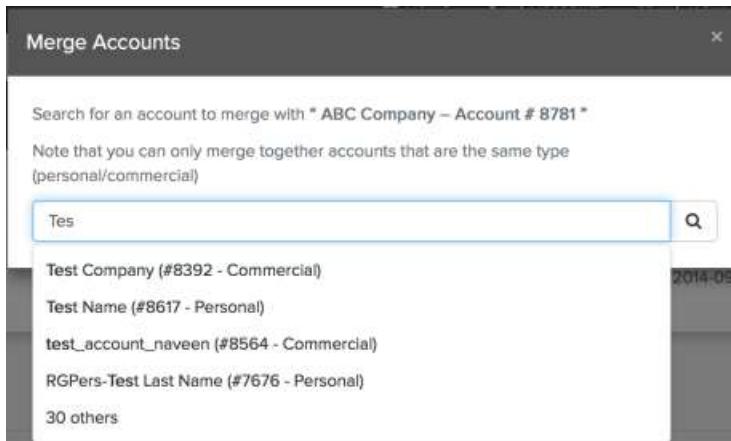
## Merging an Account

Within the account summary page, you can merge two accounts together if you feel there are duplicate accounts in the system or you just want to unify work items and account data into a single account. There is no restriction on account type when merging accounts (you can merge commercial with personal and vice versa). When you merge two accounts:

- the account holder from the source account gets assigned a named insured role in the target account

- any people, locations and work items from the source account are moved to the merged account
- the account holder of the target account remains the account holder
- all people, locations and work items in the target account remain in the merged account

The  icon on the account summary page provides the ability to merge the selected account with an existing account. After you click the merge icon, a Merge Accounts modal displays to search for the account in which to merge the account into.



The screenshot shows a 'Merge Accounts' modal window. At the top, it says 'Search for an account to merge with "ABC Company – Account # 8781"'. Below that is a note: 'Note that you can only merge together accounts that are the same type (personal/commercial)'. A search input field contains 'Tes'. To the right of the input field is a magnifying glass icon. Below the input field, a list of account suggestions is shown:

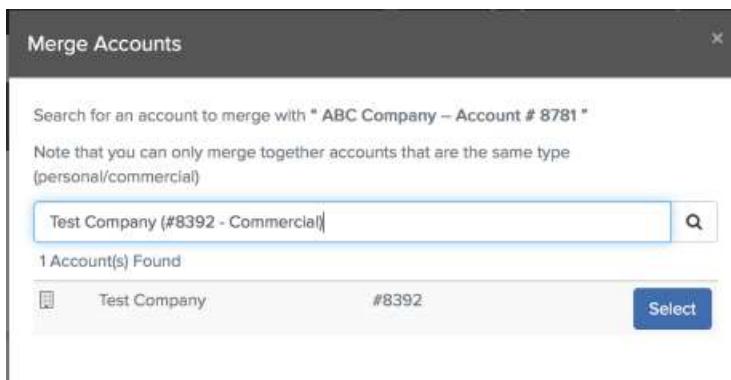
- Test Company (#8392 - Commercial)
- Test Name (#8617 - Personal)
- test\_account\_naveen (#8564 - Commercial)
- RGPers-Test Last Name (#7676 - Personal)
- 30 others

Search for an account by name (three character minimum). A list of relevant accounts automatically displays when you enter at least three characters (if you do not enter at least three characters and click the magnifying glass icon, the following message displays: *Please provide at least 3 characters.*).

The search feature uses auto suggest when three characters are entered into the search box.

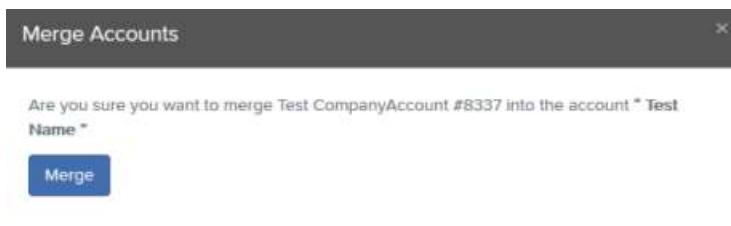
- Auto suggest returns a maximum of four matches. If there are more than four matches, XX others (where XX is the amount of additional items that meet the search criteria) displays to link additional results.
- Click the XX others link (where XX is the amount of additional items that meet the search criteria) or the search icon (magnifying glass) to display all results in a scrollable list.

Select an account and click the Select button.



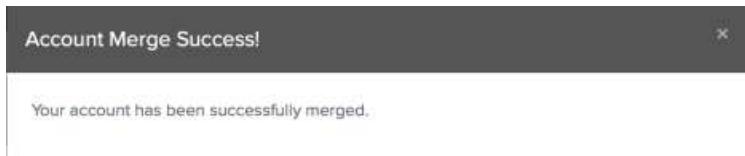
The screenshot shows a 'Merge Accounts' modal window. At the top, it says 'Search for an account to merge with "ABC Company – Account # 8781"'. Below that is a note: 'Note that you can only merge together accounts that are the same type (personal/commercial)'. A search input field contains 'Test Company (#8392 - Commercial)'. To the right of the input field is a magnifying glass icon. Below the input field, a message says '1 Account(s) Found'. A table row shows the account details: 'Test Company' and '#8392'. To the right of the table row is a blue 'Select' button.

A second merge confirmation modal displays with both accounts listed.



The screenshot shows a 'Merge Accounts' modal window. At the top, it says 'Are you sure you want to merge Test CompanyAccount #8337 into the account "Test Name"'. Below the message is a blue 'Merge' button.

Click the Merge button to confirm the merge. A confirmation modal displays to let you know the merge is complete.



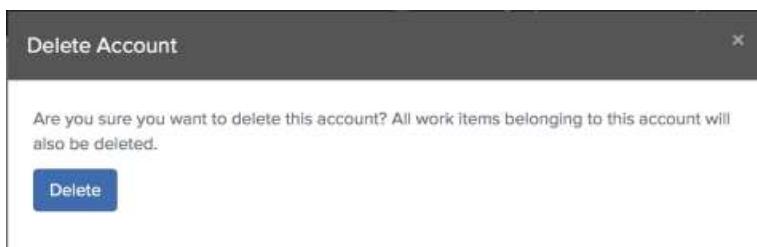
## Deleting an Account

There are two ways to delete an account:

- Click the Delete button directly from the My Accounts page, or
- Click the  icon on the account summary page

These options only display if you have authority (based on user permissions) to delete accounts.

When either option is selected, the Delete Account modal displays. This modal asks you to confirm whether you want to perform the delete action.



Click the Delete button to confirm the deletion. The account is automatically removed from the system, along with any associated work items, locations and people added to the account.

# Reports

The My Reports option on the navigation menu displays the reports configured for the carrier's AgencyPortal application. This can include pie charts or line/bar/column charts.

These reports:

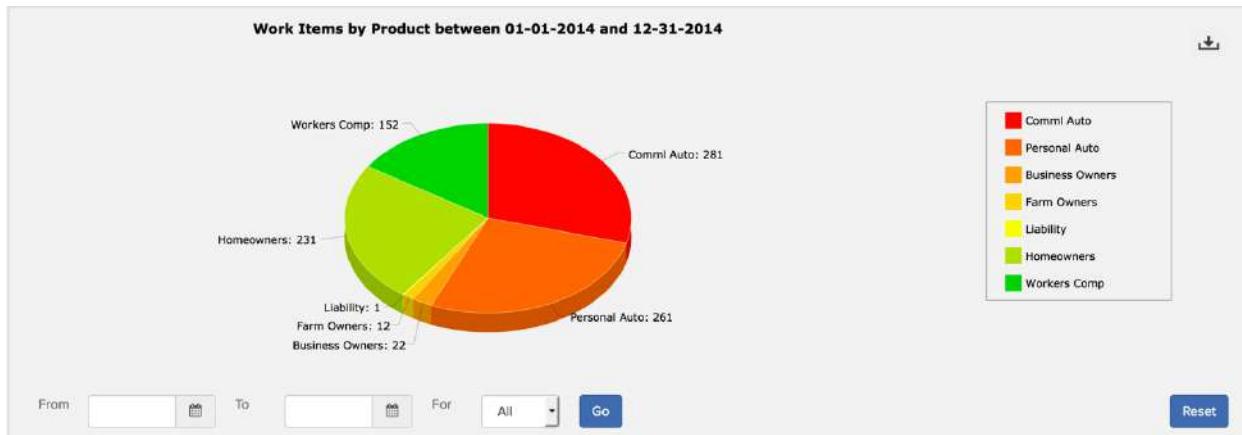
- use the latest version of amCharts
- have no dependency on Flash
- use one technology (amCharts) to render in any browser
- include interactivity to manipulate the data that displays (e.g., you can click on an LOB to remove it from the chart or slide the date rule at the top of a chart to change the date range).

You can perform the following from the Reports page:

- Download a copy of the report in PDF, JPG or SVG format and save it to a desired location.
- Enter specific criteria, such as a to and from date and by agency. A blank in the to/from fields searches all date ranges.
- Reset the report criteria. Click the Reset button to remove any report filters and re-display the default report.

By default, the following reports are included in the AgencyPortal application.

Work Items by Product



This is a pie chart report that displays submissions by line of business and displays what was submitted over a selected period of time.

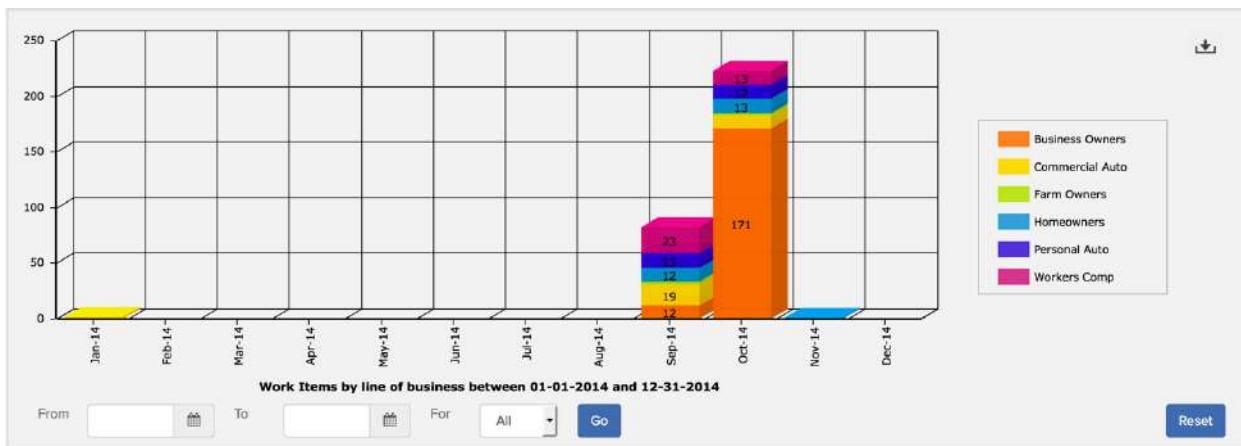
This report includes the following interactivity:

- Display results based on a from and to date and/or for a selected agency.
- Display line of business results by selecting and/or un-selecting the categories from the legend.

Line of Business Specific Report (with work item count by status)

This is a pie chart report that displays work item count by status for a specific line of business. It is a drilled-down report accessible from the Work Items by Product report.

Work Items by Line of Business



This is a stacked bar chart that displays the number of submissions entered in the application each month (by line of business) over a selected period of time.

This report includes the following interactivity:

- Display results based on a from and to date and/or for a selected agency.
- Display line of business results by selecting and/or un-selecting the categories from the legend.

#### Premium by Month

This is a line chart report that displays premiums by month.

This report includes the following interactivity:

- Display results based on a from and to date and/or for a selected agency.
- Display line of business results by selecting and/or un-selecting the categories from the legend.

#### Cumulative Premium by Month

This is a line chart report that displays cumulative premiums by line of business over a selected period of time.

This report includes the following interactivity:

- Display results based on a from and to date and/or for a selected agency. You can use either the from and to fields or drag the toggle bars to a from/to period at the top of the chart.
- Display line of business results by selecting and/or un-selecting the categories from the legend.

The following additional reports are also available:

<b>Top 10 Broken Underwriting Rules</b>	This is a horizontal bar chart that displays the top 10 rules triggered over a selected period of time.
<b>Broken UW Rules Detail Report</b>	This report displays the work item number, userID, agency name and a rule message for all triggered rules. This is a drilled-down report that is accessible from the Top 10 Broken Underwriting Rules report.
<b>System Events</b>	This is a tabular report that displays transaction, toolkit and user auditing events. This report is targeted for system administrators.

Refer to the [Reports](#) topic in the Developer Guide section for information on how to configure these reports.

# Upload

AgencyPortal provides the functionality to upload ACORD forms [manually](#) or automatically via an agency management system.

## Turnstile Integration

AgencyPortal's integration with Turnstile allows you to manually upload ACORD forms. The functionality included in this workflow:

- asynchronizes calls (this means no wait time for processing)
- creates multiple work items from a single upload
- selects your line of business (LOBs) for forms processed

This allows you to easily upload ACORD forms into AgencyPortal and attach the form information to new or existing accounts. If you don't want to attach the information to an account, simply create a work item for later processing.

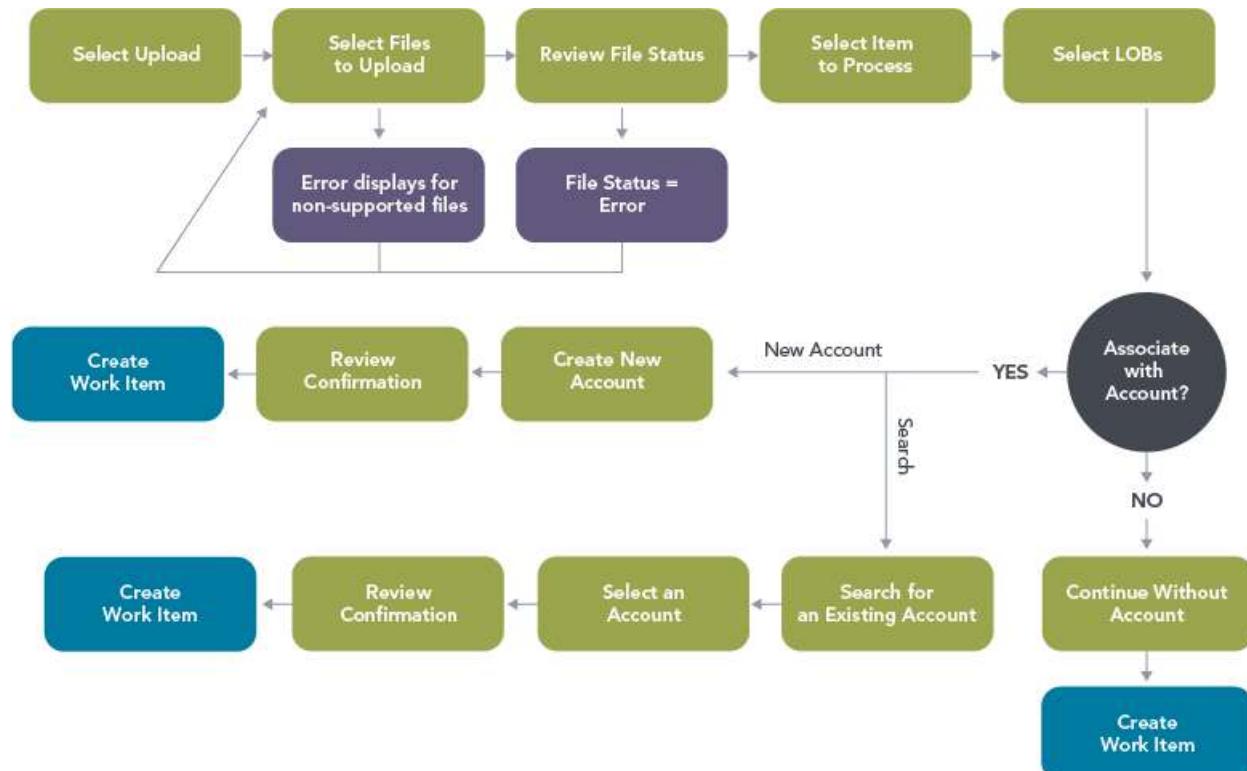
## Uploading Files

The upload functionality allows you to upload one or multiple ACORD forms for an account at a time. Depending on the number of forms included in one file, one or multiple work item transactions (depending on the LOB selected) is created for each uploaded file. The uploaded PDF form is attached to the work item for later viewing if your application is using [Work Item Assistant](#).

You can initiate a file upload from the [Get Started](#) drop-down on the navigation menu or at the account level from the [Upload Form](#) option on the My Accounts page.

### Pre-filling From an ACORD Form

The following details the process and workflow of uploading ACORD forms using the Prefill from an ACORD Form option on the Get Started drop-down on the navigation menu.

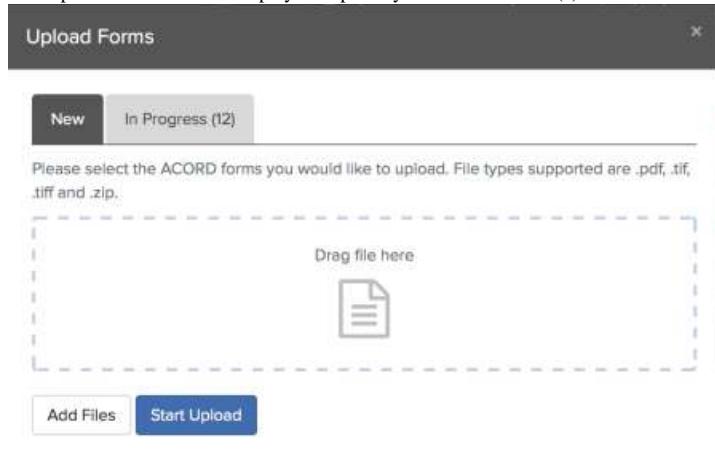


1. Click the Get Started drop-down on the navigation menu.
2. Select the Prefill from an ACORD Form option.

## Note

The number that displays next to the Prefill from an ACORD Form option indicates the number of uploaded files in the queue.

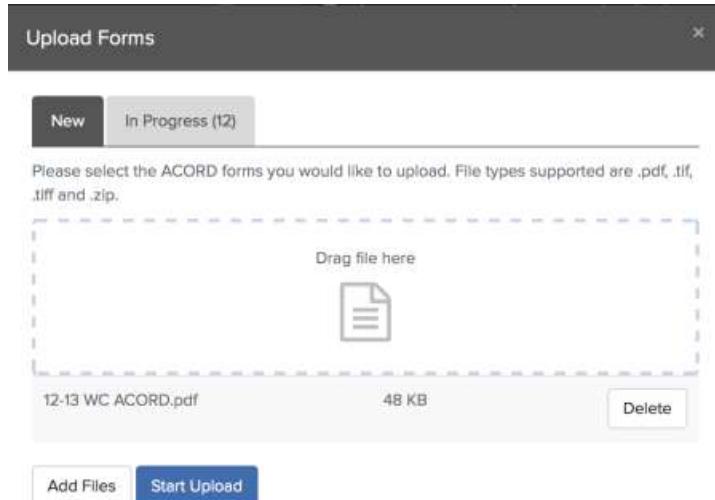
The Upload Forms modal displays to upload your ACORD form(s).



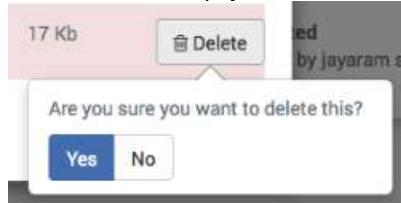
3. Click the Add Files button to add one or multiple files. All files added in a single upload should be for the same customer. You can also drag and drop files from an open file window into the modal.

## Note

If the files included in the upload are not for the same customer, the system will not prevent this action since there is no way to identify prior to the update process whether the files are all for a single customer.



4. If you decide you do not want to include a file (or files), click the Delete button next to the file to remove it from the upload list. A delete confirmation displays:



## Note

If you upload a file that is not a supported file type, an error displays. For example, when you upload a .png file, the following warning displays:

**Upload Forms**

New In Progress (12)

Please select the ACORD forms you would like to upload. File types supported are .pdf, .tif, .tiff and .zip.

Please fix errors before uploading.

Drag file here

⚠ Screen Shot 2015-09-04 at 10.05.32 AM.png 552 KB Delete

Add Files Start Upload

Click the Delete button to remove the file.

5. Click the Start Upload button. The files are set to process and are added to the queue.
  - Files in process display as Queued.
  - Files with errors display as Error. Click Error to view the error message.
  - Files processed and ready to commit display as Processed. Proceed to step 6.

**Upload Forms**

New In Progress (4)

File	Date/Time	Status	Actions
101.pdf	2014-07-15 12:20:28.0	QUEUED	X
#4A.pdf	2014-07-15 12:18:44.0	PROCESSED	X
12-13 WC ACORD.pdf	2014-07-15 12:18:42.0	ERROR	X
#47.pdf	2014-07-15 12:17:25.0	PROCESSED	X

To remove an uploaded file, click the X next to the file.

6. Select the processed file. A modal displays with the ACORD form(s) included and not included in the upload, and the line of business (LOB) to associate with the form(s). You must select at least one LOB.

#### Note

The LOB types that display on this modal are based on the LOBs supported in AgencyPortal implementation; they do not display based on the type of form uploaded.

**Upload Forms**

What lines of business do you want to quote?

<b>Forms Processed</b> ACORD_127_1994_08	<b>Lines of Business</b> <ul style="list-style-type: none"> <li><input type="checkbox"/> Business Owners Application</li> <li><input type="checkbox"/> Commercial Auto</li> <li><input type="checkbox"/> Homeowners</li> <li><input type="checkbox"/> Personal Auto</li> <li><input type="checkbox"/> Testing Application</li> <li><input type="checkbox"/> Testing Application - AUTOB</li> <li><input type="checkbox"/> Workers Comp</li> </ul>
<b>▼ Forms Not Processed: 0</b> None	
<b>▼ Pages Not Recognized: 2</b>	

**Continue**

**Back**

Forms not processed display in the Forms Not Processed section and forms not recognized display in the Forms Not Recognized section. Click the drop-down arrow to display what was not processed.

- Click the Continue button. A modal displays to indicate how you want to commit the work item.

**Upload Forms**

What account is this for?

**No Account**   **New Account**   **Search for an Account**

- No Account** – Select this option to create a work item without associating the work item with an account. A work item confirmation modal displays. You must click the Open button to display the My Work page, which may require you to enter any missing information.

#### Note

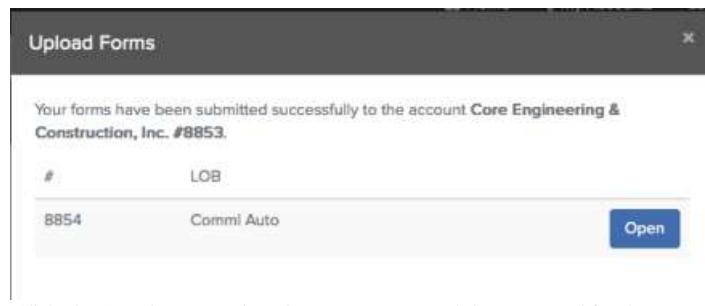
If there was an error creating a work item from the LOB(s) selected, a caution icon displays with an error message. Click the Error button to view the error returned from Connect5.

**Upload Forms**

Your forms have been submitted. Some LOBs have failed please review below.

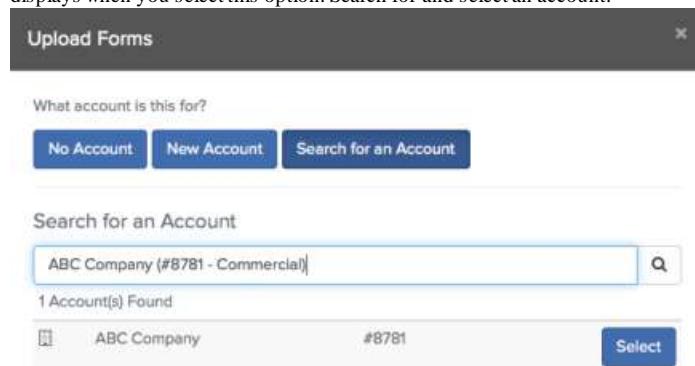
#	LOB	
2983	Commercial Auto	<b>Open</b>
	Workers Comp	<b>Error</b>

- New Account** – Select this option to create a new account and associate it with the work item. A confirmation modal displays when you select this option.



Click the Open button to view the account; any work items created for the account display on the account page.

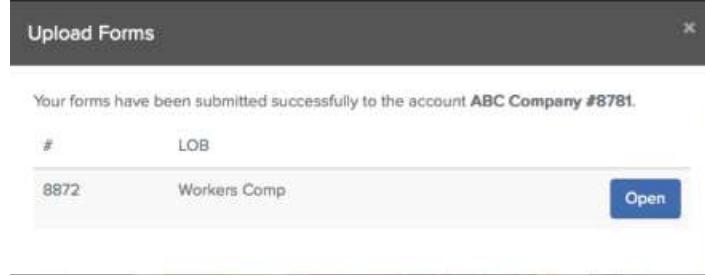
- **Search for an Account** – Select this option to search for an existing account and associate it with the work item. A search box displays when you select this option. Search for and select an account.



## Note

Refer to the [Search and Filter](#) topic for more information on this functionality.

Click the Open button to view the account; any work item(s) created for the account display on the account page.



## Note

The following modal message displays when your upload was successful, but there are items you must correct:



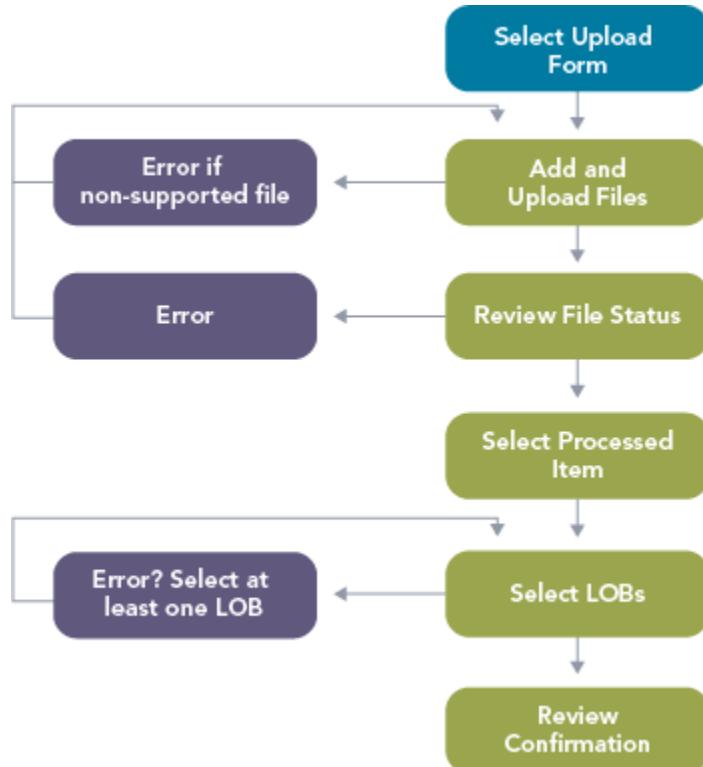
- [Jump to the first problem](#) [Double-check from the beginning](#)

- Click the "things that need to be fixed" hyperlink to view a list of errors by page and field.
- Click the "Apply your defaults" hyperlink to fill in any blank fields that correlate to saved default values.

- Click the Automatic Corrections hyperlink to view a list of items that were changed with the automatic corrections feature.
- Select the Jump to First Problem button to navigate to the first field with a validation error.
- Select the Double-check from the beginning button to navigate to the first page.

#### Pre-filling From an ACORD Form From an Account

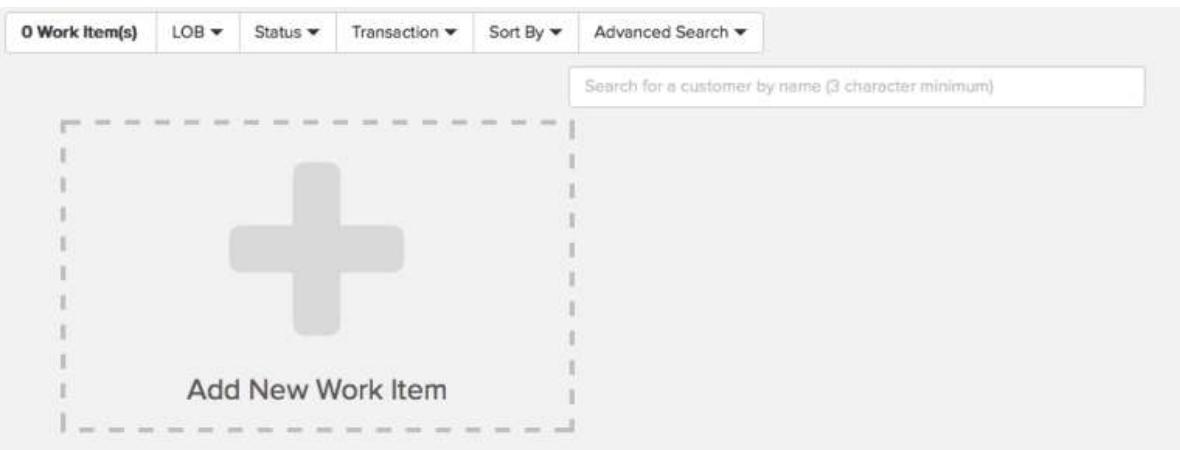
The Upload Form functionality at the account level allows you to upload ACORD forms for a selected account.



1. Click the My Accounts tab on the navigation menu. The My Account page displays.
2. Search for and select an account.

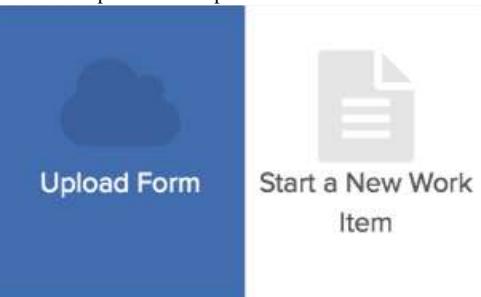


3. Click the Open button. The account information page displays.
4. Scroll down and click the Add New Work Item card.

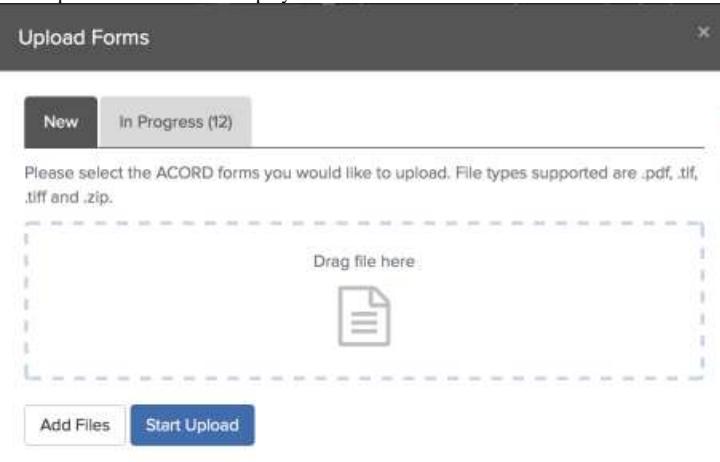


Two new options display: Upload Form and Start a New Item.

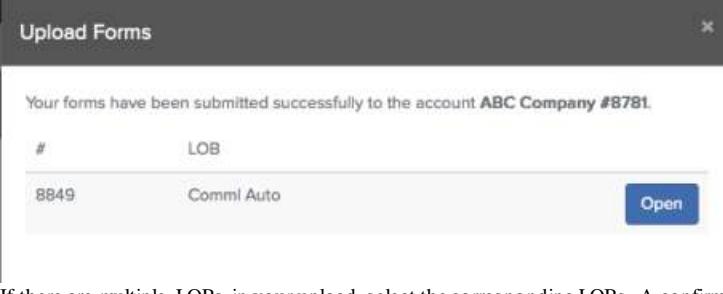
5. Click the Upload Form option.



The Upload Forms modal displays.



6. Upload the file(s). Refer to steps 3 to 6 [above](#).
7. After you upload your files, click the Continue button. A confirmation modal displays.



If there are multiple LOBs in your upload, select the corresponding LOBs. A confirmation displays with multiple work items. Click one of the Open buttons to view the item. Any other item(s) are added to the WIP page and are available to select and view.

8. Click the Open button to view the account; any work items created for the account display on the account page.

## Note

The following modal message displays when your upload was successful, but there are items you must correct:



- Click the "things that need to be fixed" hyperlink to view a list of errors by page and field.
- Click the "Apply your defaults" hyperlink to fill in any blank fields that correlate to saved default values.
- Click the Automatic Corrections hyperlink to view a list of items that were changed with the automatic corrections feature.
- Select the Jump to First Problem button to navigate to the first field with a validation error.
- Select the Double-check from the beginning button to navigate to the first page.

Otherwise, the following modal displays to let you know your upload was copied over and no corrections were needed.

## Upload Help

An Upload drop-down displays at the top of the page for work items uploaded via Turnstile or an agency management system. This menu provides options to view the source information for the uploaded work item and any relevant help information.



This drop-down displays the following options:

- Source System Info - This option displays information on the source of the upload, such as the system used, version and vendor.



- Help - This option displays help text for the upload feature, including descriptions of the field level upload messaging and icons that display when uploaded items need to be reviewed.

## Help with Upload

X

As you are working, look out for:

**Labels in Bold/Red Text** - These require your attention. These fields might be required, but not present in the upload or may just need to be adjusted or reviewed before they can be submitted.

 There are multiple scenarios for which this icon may appear.

Upload Not Available - It is possible for values to be uploaded (e.g., limits) that we do not offer. Click on the icon to see the value that was originally uploaded. It is up to you to make another selection with one of the available fields.

Changed/Invalid Values - The system changed a value that did not conform to the field format (e.g., max length exceeded) or the value was uploaded as is, but did not conform to the field format and requires correction. You can click on this icon to see the value that was originally uploaded.

 Automatic Correction Applied - The system was able to make a determination as to what value best matches the uploaded value or the system applied formatting to a value that remained otherwise unchanged.

 Change-Undo Arrows - As you change the value of an uploaded field on the screen, this icon will appear. Click on it to see the value that was originally uploaded. If you'd like, you can use the undo arrow to restore the original value by clicking on the Revert button.

# Field Level Upload Messaging

When a form is uploaded from Turnstile (or an agency management system), AgencyPortal applies logic at the field level to maximize the accuracy of the uploaded data. When an issue or a change from the original value is detected, specific field level messaging displays to grab your attention. Icons display in red and a pop over message can display next to the field to you know there is an issue or that you should review the value for accuracy.

There are several scenarios that cause these type of field level messages to display. The following describes each of these scenarios and a sample of the messaging that can display out of the box. Refer to the [Field Level Messaging](#) topic in the Developer Guide section for information on how to customize this feature.

## Mismatch

A mismatch occurs when a value that is uploaded into a select or filter list does not match any of the available values.

On the initial display of the field, a  icon and the following message display to let you know a perfect match was not found between the uploaded value and the one in AgencyPortal: "There was not a perfect match between what was uploaded from your agency management system (XXXX) and the options in the list. Please make another selection".

If the field is required, the field, icon and message display in red.

### Example



The screenshot shows a form with three fields: 'Started' (text input), '\* SIC Code' (select dropdown with 'Select One' placeholder), and 'NAICS Code' (text input). The 'SIC Code' field has a red border and a red error icon (a red circle with a white slash). A red callout box points to the field with the message: 'There was not a perfect match between what was uploaded from your agency management system (22) and the options in the list. Please make another selection.'

If the field is not required, the field, icon and message display in grey.

### Example



The screenshot shows a form with three fields: 'Mail Address' (text input), 'Legal Entity' (select dropdown with 'Select One' placeholder), and 'Area Name' (select dropdown with 'Select One' placeholder). The 'Legal Entity' and 'Area Name' fields have grey borders and grey error icons. A grey callout box points to the 'Legal Entity' field with the message: 'There was not a perfect match between what was uploaded from your agency management system (SeeRemarks) and the options in the list. Please make another selection.'

Since a value did not match, the list defaults to the Select One option. After you select a value, the icon, field and message no longer display in red if it is a required field; however, they will always display next to the field to let you know there was a change from the original uploaded value.

### Example



The screenshot shows a form with a single field: '\* SIC Code' (select dropdown). The dropdown contains the value '800 Forestry'. To the right of the dropdown is a grey error icon (a grey circle with a white slash).

## Automatic Corrections

An automatic correction occurs when the system determines how an uploaded value should be mapped when there is no exact match. Autocorrect is engaged when

- The system finds a value in a select list that is a 'best match' for the uploaded value (aka, bumping and slotting).

- The system applies formatting to the uploaded value (e.g., dashes are added to a tax ID number or format is applied to a phone number), but does not actually change the value.

You will have the option to review the system-selected value and modify it if necessary. On initial display, a  icon displays next to the field and the following message displays to let you know an automatic correction was applied: "The value of this field has been automatically corrected since it was uploaded. The original value was XXXX".

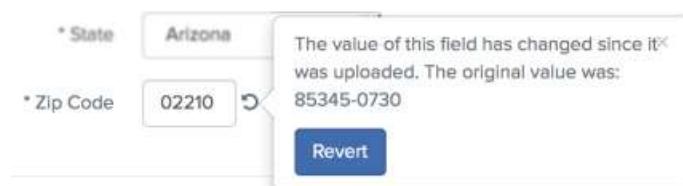
#### Example



A screenshot of a form with three fields: Name, Office Phone, and Mobile Phone. The Name field contains "Mary LaRue Walker". The Office Phone field contains "(480) 783-4620" with a  icon next to it. A callout bubble points to the icon with the text: "The value of this field has been automatically corrected since it was uploaded. The original value was +1-480-7834620."

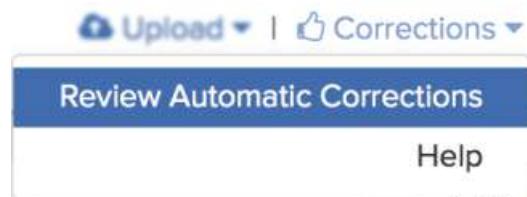
After you review the value, the message no longer displays; however, the icon will always display next to the field to let you know there was a change from the original uploaded value. Click the icon to display the original value.

If you decide to change the original value, a  icon also displays next to the field. This allows you to change the value back to the original autocorrected value.



A screenshot of a form with two fields: State and Zip Code. The State field contains "Arizona" and the Zip Code field contains "02210" with a  icon next to it. A callout bubble points to the icon with the text: "The value of this field has changed since it was uploaded. The original value was: 85345-0730". Below the fields is a "Revert" button.

A list of all automatic corrections is also available from the Corrections drop-down menu.



This gives you a quick way to review of all of the fields within the work item that had automatic corrections applied.

**Automatic Corrections**

Made on 2015-11-05

Field: Producer Phone

Was: +1-602-9562250  
Is Now: (602) 956-2250

Error: Agency Information.Producer Phone is not a recognized phone format

Notes: The system adjusted the value to comply with validation requirements

Field: Producer Fax

Was: +1-602-9562258  
Is Now: (602) 956-2258

Error: Agency Information.Producer Fax is not a recognized phone format

Notes: The system adjusted the value to comply with validation requirements

## Validations

A validation occurs when an uploaded value does not conform to the field format and must be corrected. On the initial display of the field, a  icon displays next to the field and the following message displays to let you know the value uploaded was invalid or does not conform to the format of the field in AgencyPortal: "The value of this field does not conform to the field format and requires correction. The original value was XXXX".



## Manual Update

A manual update occurs when you change an uploaded value that passes field validation and has not been changed by the system. After the field is manually changed, a  icon displays next to the field. Click the icon to view the following message: "The value of this field has changed since it was uploaded. The original value was: XXXX". A Revert button also displays, which allows you to revert the field back to its originally uploaded value.

### Example



Roster entries display a little differently. If an originally uploaded roster item is deleted, an Undo button displays to revert the deletion.

Example

For Location	Class Code	Full Time Empl	Part Time Empl	Est. Annual Exposure	
3252 S. Auto Way Gilbert, AZ 85296	0006			254507	<b>Undo</b>

# PDF Spotlight

AgencyPortal allows users to quickly view the original PDF as a reference to determine whether the data upload correctly if it comes from a poor quality scanned form.

This feature is configured based on application properties. Refer to the [Turnstile Integration](#) topic in the Developer Guide section for information on how to set up and configure this feature.

The screenshot shows the AgencyPortal interface with a sidebar menu on the left and a main form area. A red arrow points to a field in the main form where the placeholder text "Please enter first named insured and all other named insureds. Use additional space if needed." is visible. To the right, a larger window displays the original uploaded PDF, specifically a "COMMERCIAL INSURANCE APPLICANT INFORMATION" form. The "CONTACT" section of the PDF is highlighted with a red border, showing fields like Name, Address, and Email. The "AGENCY CUSTOMER ID" field is also highlighted.

While working in the transaction and you tab onto or select a field, the corresponding field data is highlighted in the original uploaded PDF.

The screenshot shows the "Applicant Information" section of the transaction. On the left is the AgencyPortal form with fields for "Applicant Name" (David Turnstile) and "Tax Id Type" (FEIN). On the right is the original uploaded PDF, which includes sections for "POLICY INFORMATION", "APPLICANT INFORMATION", and "NAME (First Named Insured) AND MAILING ADDRESS (including ZIP+4)". The "NAME (First Named Insured) AND MAILING ADDRESS" section is highlighted with a yellow background.

This allows you to review the field data in both the PDF and the AgencyPortal field and make any necessary changes before continuing.

A toggle button displays in the upper right corner of the work item to display or not display the original PDF while working the transaction.



# External References

This section provides links and additional information on external references. This includes:

- [JavaDoc](#)
- [JSDoc](#)

# JavaDoc

Click [here](#) to view the AgencyPortal SDK JavaDoc.

# JSDoc

Click [here](#) to view the AgencyPortal SDK JSDoc.