Alfresco Content Services 5.2

Developer Guide

Contents

Developer Guide

The Developer Guide includes extensive guidance and reference materials to aid the developer in creating applications and extensions for Alfresco.

This Developer Guide PDF does not include the full developer documentation and API references. To access the complete set of documentation, see the online Alfresco documentation.

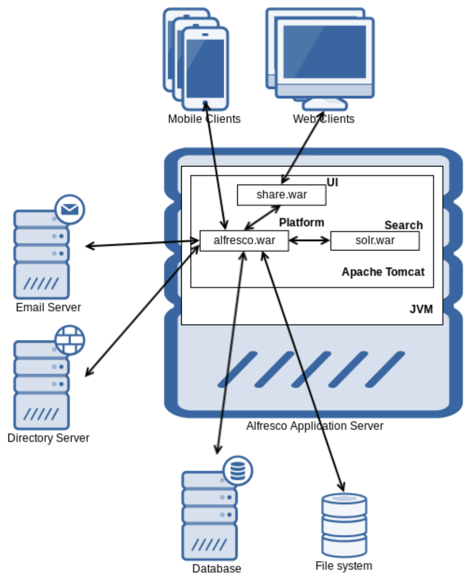
There are a number of approaches to developing for Alfresco depending on what you want to do. For example, if you are writing a client application, perhaps in Ruby or Python to connect to Alfresco either on-premise, or Alfresco in the Cloud, then you would most likely use the Alfresco REST API. If on the other hand you wanted to write a server-side extension in Java, you would use the Public Java API, or perhaps write a web script using Java, JavaScript and FreeMarker. Generally if you are creating extensions to Alfresco you would use the Alfresco SDK. This allows you to work in your IDE of choice, using technologies you already know, such as Java and Maven.

This Developer guide attempts to lay out the various options available to you, so you can use the right approach, depending on what you want to achieve.

Alfresco Content Services architecture

This gives a view of the architecture of Alfresco Content Services from the developer's perspective. At its core is a repository that provides a store for content, and a wide range of services that can be used by content applications to manipulate the content.

The following diagram illustrates the idea that can be thought of as consisting of three main components, **Platform**, User Interface (**UI**), and **Search**. These components are implemented as separate web applications:



The main component is called the **Platform** and is implemented in the `alfresco.war` web application. It provides the repository where content is stored plus all the associated content services. Alfresco Share provides a web client interface (that is a User Interface, UI) for the repository and is implemented as the `share.war` web application. Share makes it easy for users to manage their sites, documents, users and so on. The search functionality is implemented on top of Apache **Solr 4** and provides the indexing of all content, which enables powerful search functionality. Besides the web clients accessing the Repository via Share there are also mobile clients that will access the content via REST APIs provided by the platform.

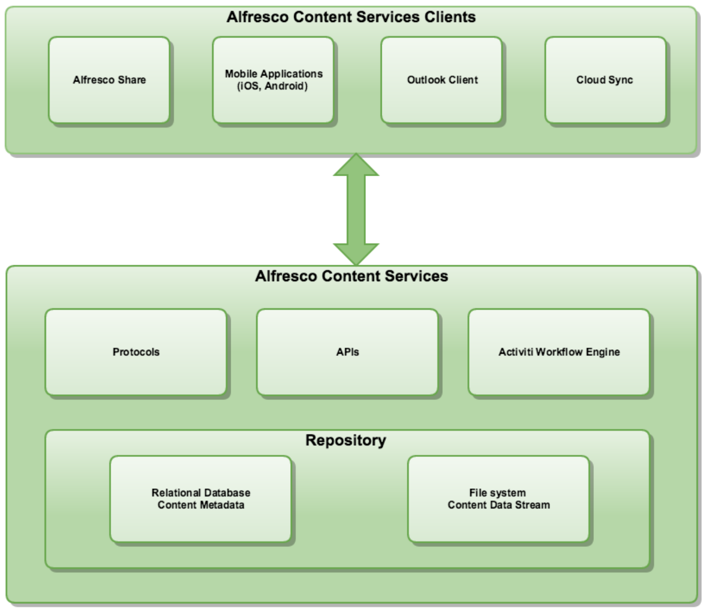
If we dive deeper into the platform (packaged in `alfresco.war`) we will see that it also supports workflow in the form of the embedded Activiti Workflow Engine. The platform is usually also integrated with a Directory Server (LDAP) to be able to sync users and groups with Alfresco Content Services. And most installations also integrates with an SMTP server so the Platform can send emails, such as site invitations.

For more information about the internals of the Platform, and specifically the content repository, see the concepts section.

Besides Share there are also many other clients that can connect to the repository, including any CMIS-compatible client, and via the Microsoft SharePoint protocol any SharePoint client. Enterprise to Cloud Sync can synchronize content between an on-premise installation of and in the Cloud, under user control.

The Platform also contains numerous APIs, Services, and Protocols.

The following diagram illustrates this extended architecture:


Alfresco Content Services Clients
Alfresco Share
Mobile Applications (iOS, Android)
Outlook Client
Cloud Sync
Alfresco Content Services
Protocols
APIs
Activiti Workflow Engine
Repository
Relational Database Content Metadata
File system Content Data Stream

Note that content metadata is stored in a relational database system such as PostgreSQL, MySQL, Oracle, and so on. The content itself is stored on the file system (or other storage system such as Amazon S3).

Alfresco provides a number of extension points to allow you to customize it. These extensions points have various formats, but include:

- Platform extension points and detailed architecture
- Share extension points and detailed architecture
- Share integration points and detailed architecture
- APIs
- Protocols
- Services

The links in the list above provide further information on each of these topics.

Alfresco Content Services architecture overview

At the core of the Alfresco Content Services system is a repository supported by a server that persists content, metadata, associations, and full text indexes. Programming interfaces support multiple languages and protocols upon which developers can create custom applications and solutions. Out-of-the-box applications provide standard solutions such as document management and records management.

As a Java application, the system runs on virtually any system that can run Java Enterprise Edition. At the core is the Spring platform, providing the ability to modularize functionality, such as versioning, security, and rules. Alfresco Content Services uses scripting to simplify adding new functionality and developing new programming interfaces. This portion of the architecture is known as *web scripts* and can be used for both data and presentation services. The lightweight architecture is easy to download, install, and deploy.

There are many ways to deploy, however most deployments follow a general pattern. Ultimately, Alfresco Content Services is used to implement ECM solutions, such as document management and records management. There can also be elements of collaboration and search across these solutions.

The solutions are typically split between clients and server, where clients offer users a user interface to the solution and the server provides content management services and storage. Solutions commonly offer multiple clients against a shared server, where each client is tailored for the environment in which it is used.

Clients

Alfresco Content Services offers a web-based client called Alfresco Share, built entirely with the web script technology. Share provides content management capabilities with simple user interfaces, tools to search and browse the repository, content such as thumbnails and associated metadata, previews, and a set of collaboration tools such as wikis and discussions. Share is organized as a set of sites that can be used as a meeting place for collaboration. It's a web-based application that can be run on a different server to the server that runs the repository, providing opportunities to increase scale and performance.

Share can be deployed to its own tier separate from the content application server. It focuses on the collaboration aspects of content management and streamlining the user experience. It's implemented using Surf and can be customized without JSF knowledge.

Clients also exist for portals (by using JSR-168 portlets), mobile platforms, Microsoft Office, and the desktop. In addition, using the folder drive of the operating system, users can share documents through a network drive. Using JLAN technology, Alfresco can look and act just like a folder drive. JLAN is the only Java server-side implementation of the CIFS protocol, letting users interact with Alfresco Content Services as they do any other normal file drive except the content is now stored and managed in the content application server.

Server

The content application server comprises a content repository and value-added services for building solutions.

The content application server provides the following categories of services built upon the content repository:

- Content services (transformation, tagging, metadata extraction)
- Control services (workflow, records management, change sets)
- Collaboration services (social graph, activities, wiki)

Clients communicate with the content application server and its services through numerous supported protocols. HTTP and SOAP offer programmatic access while CIFS, FTP, WebDAV, IMAP, and Microsoft SharePoint protocols offer application access. The Alfresco Content

Services installer provides an out-of-the-box prepackaged deployment where the content application server and Share are deployed as distinct web applications inside Apache Tomcat.

Guiding design principles

The Alfresco Content Services architecture supports the requirements of Enterprise Content Management (ECM) applications, such as Document Management (DM), Web Content Management (WCM), Records Management (RM), Digital Asset Management (DAM), and Search.

Support ECM requirements

Each of these disciplines has unique and overlapping characteristics so that the design of each capability is not done in isolation but in the context of the whole system.

Simple, simple, simple

Alfresco Content Services aims to be as simple as possible to develop against, customize, deploy, and use. The simplest and probably most widely deployed ECM solution is the shared document drive: the architecture is driven by the aim to be as simple as a shared drive.

Scaling to the enterprise

Every service and feature is designed up front to scale in terms of size of data set, processing power, and number of users.

Modular approach

Alfresco Content Services architecture takes a modular approach in which capabilities are bundled into modules whose implementation can be replaced if required, or not included at all. Aspect-Oriented Programming (AOP) techniques allow for fine-tuning and optimization of an ECM solution.

Incorporating best-of-breed libraries

Where possible, Alfresco Content Services incorporates best-of-breed third-party libraries. The open source nature lends itself to integrating with the wealth of available open source libraries. This is done whenever it is more profitable to integrate than build or whenever expertise is better provided in another project rather than in-house.

Environment independence

Alfresco Content Services does not dictate the environment upon which it depends, allowing choice in the operating system, database, application server, browser, and authentication system to use when deploying. ECM is less about the application and more about the services embedded within an application. You can choose how to package Alfresco Content Services — for example, as a web application, an embedded library, or portlet.

Solid core

The heart of Alfresco Content Services is implemented in Java. This decision was driven by the wealth of available Java libraries, monitoring tools, and enterprise integrations. Java is also a trusted runtime for many enterprises wishing to deploy applications in their data centers. Each capability is implemented as a black-box Java service tested independently and tuned appropriately.

Scriptable extensions

Extensions will always need to be created for custom solutions and there are many custom solutions versus the single Alfresco Content Services core. Therefore, extension points are developed using JVM-based scripting languages, allowing a much wider pool of developers to build extensions versus those that can contribute to the core. Extensions are packaged entities, allowing for the growth of a library of third-party reusable extensions.

Standards-based approach

The architecture always complies with standards where applicable and advantageous. Primary concerns are to reduce lock-in, improve integration possibilities, and hook into the ecosystems built around the chosen standards.

Architecture of participation

The architecture promotes a system designed for community contribution. In particular, the architecture principles of a solid core, modularity, standards compliance, simplicity of development, and scriptable extensions encourage contribution of plug-ins and custom ECM solutions. Participation complements the open source approach to the development of Alfresco Content Services and fosters growth of the Alfresco community. As the community grows, the quality of self service improves, as well as the quality of feedback. This, in turn, enhances Alfresco Content Services and creates the ultimate feedback loop.

Web tier and Surf

Alfresco Content Services provides ECM capabilities as data services, user interfaces, and user applications. The user interface capabilities are provided by applications and application components using Alfresco Content Services web tier, Surf, originally developed as a faster way to develop content applications using scripting and REST architecture.

Development of web scripts allows for the creation of a REST-based API. Web scripts can be executed without compilation, and provide a quick and easy way to extend and enhance Alfresco Content Services standard services.

The web script infrastructure accommodates Java beans as easily as JavaScript. Web scripts add little overhead but provide a great deal of flexibility and development productivity. Web scripts in the web tier let you quickly build user interface components with Surf or simple HTML and deploy them as Alfresco Share components, portlets, or other web platforms such as Google Gadgets.

Alfresco Share client application

The Alfresco Content Services client application provide a means of accessing the repository. Alfresco Content Services provides Alfresco Share which is a web-based client application, providing an interface that allows the user to create, upload, and manage content.

The user interface is built entirely with the Alfresco web script technology, which can be used to extend the application. Share provides content management capabilities with simple user interfaces, tools to search and browse the repository, content such as thumbnails and associated metadata, renditions of content, and a set of collaboration tools such as wikis, discussions, and blogs. Alfresco Share is organized as a set of sites that can be used as a meeting place for collaboration. Alfresco Share is a web-based application that can be run on a different server to the server that runs the repository, providing opportunities to increase scale and performance.

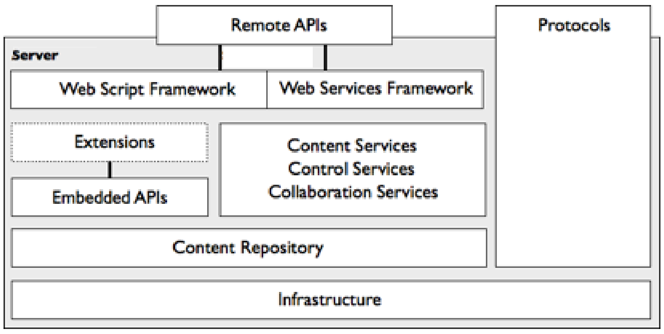
Application server

At the heart of Alfresco Content Services is the application server, which manages and maintains the repository. The server's primary responsibility is to provide services for use in building ECM solutions. All the applications of the Alfresco Content Services suite are built upon and executed by the application server.

The application server exposes a set of remote public interfaces for allowing a client to communicate with it. The remote public interfaces are the only part of the server visible to the client. There are two types:

- **Remote APIs** - for interacting with services of the server programmatically

- **Protocol bindings** - for mapping services for use by a protocol-compliant client



Internally, the server comprises several layers. The foundation includes infrastructure concerns, such as configuration, authentication, permissions, and transactions that cut across all capabilities. Infrastructure also shields the server from being tied to any specific environmental implementation, such as transaction managers or caching mechanisms.

The repository is built on this infrastructure, which itself is the building block for content, control, and collaboration services. Each capability of the repository and content services is individually bundled as a module with its own in-process interface and implementation. Modules are bound together by the infrastructure through their interfaces.

You can deploy extensions to the content application server to extend or override its capabilities. Their implementation might use the in-process interfaces offered by the repository and content services.

Repository

The repository is comparable to a database, except that it holds more than data. The binary streams of content are stored in the repository and the associated full-text indexes are maintained by SOLR indexes.

The actual binary streams of the content are stored in files managed in the repository, although these files are for internal use only and do not reflect what you might see through the shared drive interfaces. The repository also holds the associations among content items, classifications, and the folder/file structure. The folder/file structure is maintained in the database and is not reflected in the internal file storage structure.

The repository implements services including:

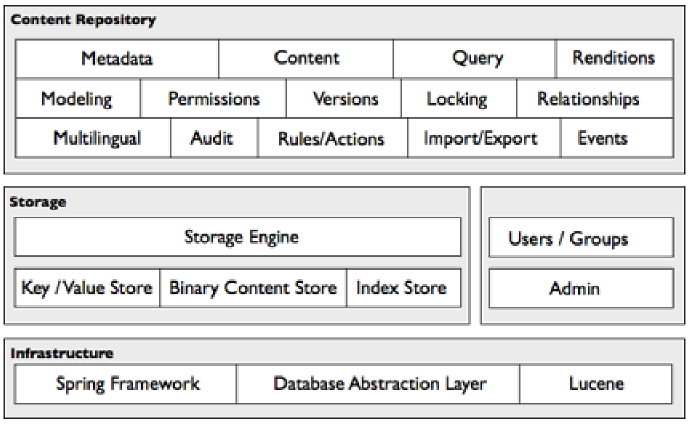
- Definition of content structure (modeling)
- Creation, modification, and deletion of content, associated metadata, and relationships
- Query of content
- Access control on content (permissions)
- Versioning of content
- Content renditions
- Locking
- Events
- Audits
- Import/Export
- Multilingual

- Rules/Actions

The repository implements and exposes these services through an API, CMIS protocol bindings, and the JSR-170 Java API. The storage engine of the repository stores and retrieves content, metadata, and relationships, and operates on the following constructs:

- **Nodes** - provide metadata and structure to content. A node can support properties, such as author, and relate to other nodes such as folder hierarchy and annotations. Parent to child relationships are treated specially.
- **Content** - the content to record, such as a Microsoft Word document or an XML fragment.

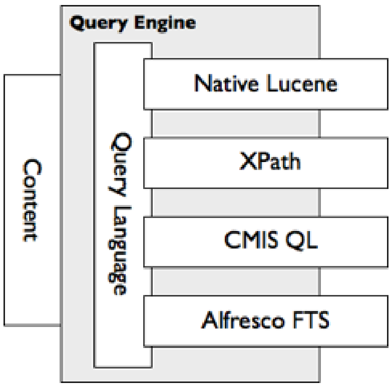
Content models are registered with the repository to constrain the structure of nodes and the relationships between them, and to constrain property values.



The storage engine also exposes query capabilities provided by a custom query engine built on Apache Lucene that supports the following search constructs:

- Metadata filtering
- Path matching
- Full text search
- Any combination of these search constructs

The query engine and storage engines are hooked into the transaction and permission support of the infrastructure, offering consistent views and permission access. Several query languages are exposed, including native Lucene, XPath, Alfresco FTS (Full Text Search), and CMIS Query Language (with embedded Alfresco FTS).



By default nodes are stored in an RDBMS while content is stored in the file system. Using a database provides transaction support, scaling, and administration capabilities. A database abstraction layer is used for interacting with the database, which isolates the storage engine from variations in SQL dialect. This eases the database porting effort, allowing certification against all the prominent RDBMS implementations. The file system stores content to allow for very large content, random access, streaming, and options for different storage devices. Updates to content are always translated to append operations in the file system, allowing for transaction consistency between database and file system.

You can bundle and deploy the repository independently or as part of a greater bundle, such as the application server.

Content services

Services address the core use cases for content management applications including the logical organization of content, file management, version control, and security. Services also support the control of content through workflow and process management, and social and collaborative applications.

Alfresco Content Services exposes services at various levels including:

- Java
- Scripting
- REST
- Web services
- Client interfaces, such as Alfresco Share

Some services are considered internal; others are public. For example, the Java level services are internal. The majority of these are accessible through other public interfaces including the public APIs, client applications, and CMIS.

Services are divided into two main categories; application services and repository services.

Programming models

A number of programming models are available for building an application using the content application server.

- The simplest model for non-programmers is to use out-of-the-box components of the Alfresco Share application and the Rules and Actions model, a set of conditions and actions to take on content based on those conditions. You can define rules and actions using a wizard and perform actions such as converting content, moving content, or executing a simple JavaScript snippet.
- Web scripts let you perform more sophisticated processing without complex programming. The Content Management Interoperability Services (CMIS) implementation was built using web scripts. By using JavaScript to build these data services, it is easy to create new services. To build new user interfaces or extensions to Share, you can also use web scripts by using a web templating language like FreeMarker. Most of Sharewas built using web scripts.
- To use Java to build applications or extend Share, you can use the many tools associated with Java that were used to build the system. Surf, the web runtime framework, lets you extend Share and build web applications. Because Share was built using Surf, you can build your own extensions as a combination of Java programming and web scripts, or with Java alone. You can also use Java to access or even replace whole pieces of Alfresco Content Services, content application server, or Share by using the Spring platform. You can use the source code as an example for rewriting pieces and using Spring beans and configuration to extend or replace functionality in Alfresco Content Services.

- To write applications that use Alfresco Content Services but are portable to other ECM systems, you can use Content Management Interoperability Services (CMIS), the OASIS standard for accessing content repositories.

APIs

To access and extend out-of-the-box services, the content application server exposes two flavors of API, each designed for a specific type of client.

The two main categories of API are embedded and remote APIs.

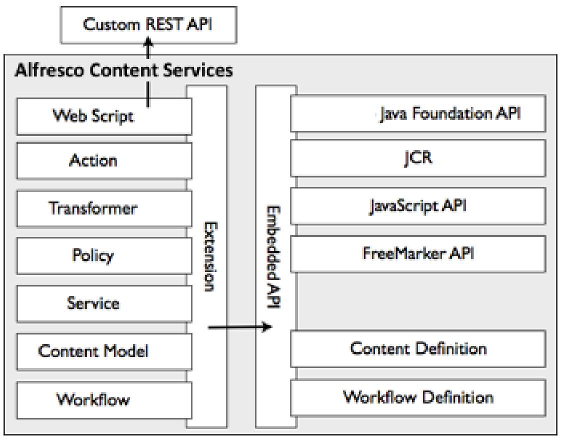
Embedded APIs

The Embedded API is used for developing extensions to the application server. Extensions deployed into the server often depend on existing services provided by the server. Therefore, developers of extensions use the Embedded API to gain access to those services.

The Embedded API comes in several forms, where each form is structured for a particular need or kind of extension:

- Alfresco Public Java API - a set of public Java interfaces exposed by services built into the content application server
- JavaScript API - an object-oriented view of the Java Foundation API specifically tailored for use in JavaScript. There is a JavaScript API for the repository tier and a JavaScript API for the Share tier.
- FreeMarker API - an object-oriented view of the Java Foundation API specifically tailored for use in FreeMarker templates
- Content Definition - an API for creating and editing content models
- Workflow Definition - an API for defining business processes

The JavaScript and Template APIs are the key building blocks for web scripts to develop the RESTful APIs.

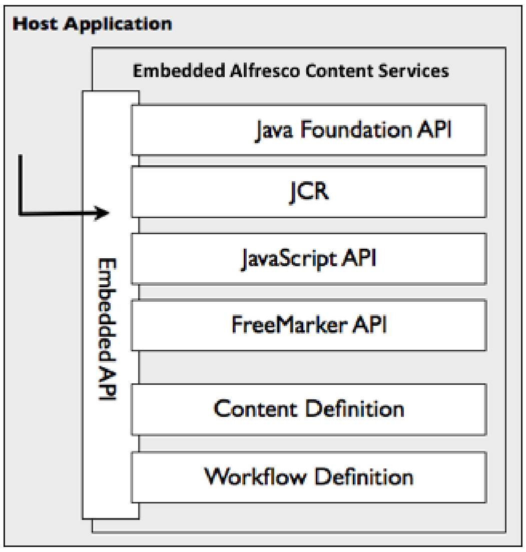


Web scripts are a popular extension for the content application server. They allow you to define your own Remote API for clients to interact with the content application server. A web script implementation can use any of the Embedded APIs, such as the Public Java API, JavaScript, and FreeMarker, for its implementation. Developing your own Remote API is very useful for the following scenarios:

- Exposing new extension services deployed into the application server to remote clients
- Providing alternate batching or transaction demarcation of existing services

- Creating a facade for integration with a third-party tool, such as a Forms engine

There is another use case for the Embedded API. An application or client can also directly embed the content application server to inherit its suite of content services.



The infrastructure of the server means it can be deployed into a number of environments, not just as a web application. Essentially, the content application server is treated as a library, where any of its services, including the content repository, can be chosen independently or mixed to provide a custom solution. The server can scale down as well as up.

Remote APIs

The Remote API is primarily used to build ECM solutions against the content application server.

There are three main remote APIs:

1. Alfresco Content Services API
2. CMIS API
3. Repository REST API (Deprecated)

The Alfresco Content Services API was introduced with version 4.x, and is also present in Alfresco Content Services in the Cloud. It provides the main remote API, and is the recommended API for developing remote client applications to work across cloud, on-premise and hybrid deployments. It comprises two sub-APIs, the Alfresco Content Services REST API for gaining access to Alfresco Content Services-specific functionality such as sites, and a standard CMIS API for repository manipulation and management. SDKs such as the Mobile SDK for Android and the Mobile SDK for iOS both use the services of the Alfresco Content Services API.

CMIS provides a standardized set of common services for working with content repositories. CMIS is not language-specific, it does not dictate how a repository works, and it does not seek to incorporate every feature of every repository. Alfresco Content Services provides an implementation of CMIS Web service and RESTful bindings, as well as a CMIS client API for use in Surf and other environments.

The Repository REST API provides access to the core repository functionality using a RESTful approach. This is useful where the developer does not want to, or have a need to, write custom web scripts, and is developing a client-side application. This API can be thought of as a ready-built collection of web scripts that can be called from any client capable of making REST requests and receiving the associated responses.

For more information about the APIs and their support status see the API overview page.

Content modeling

Content modeling is a fundamental building block of the repository that provides a foundation for structuring and working with content.

For more information about working with custom metadata models (aspects, types and forms), flexible content organization and actions in the Model Manager, see Content modeling.

Content modeling specifies how nodes stored in the repository are constrained, imposing a formal structure on nodes that an application can understand and enforce. Nodes can represent anything stored in the repository, such as folders, documents, XML fragments, renditions, collaboration sites, and people. Each node has a unique ID and is a container for any number of named properties, where property values can be of any data type, single or multi-valued.

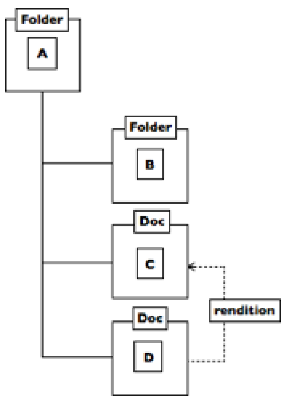
Nodes are related to each other through relationships. A parent/child relationship represents a hierarchy of nodes where child nodes cannot outlive their parent. You can also create arbitrary relationships between nodes and define different types of nodes and relationships.

A content model defines how a node in the repository is constrained. Each model defines one or more types, where a type enumerates the properties and relationships that a node of that type can support. Often, concepts that cross multiple types of node must be modeled, which the repository supports through aspects. Although a node can only be of a single type, you can apply any number of aspects to a node. An aspect can encapsulate both data and process, providing a flexible tool for modeling content.

Content modeling puts the following constraints on the data structure:

- A node must be of a given kind.
- A node must carry an enumerated set of properties.
- A property must be of a given data type.
- A value must be within a defined set of values.
- A node must be related to other nodes in a particular way.

These constraints allow the definition (or modeling) of entities within the domain. For example, many applications are built around the notion of folders and documents. It is content modeling that adds meaning to the node data structure.



The repository provides services for reading, querying, and maintaining nodes. Events are fired on changes, allowing for processes to be triggered. In particular, the repository provides the following capabilities based on events:

- Policies: event handlers registered for specific kinds of node events for either all nodes or nodes of a specific type

- Rules: declarative definition of processes based on addition, update, or removal of nodes (for example, the equivalent of email rules)

Models also define kinds of relationships, property data types, and value constraints. A special data type called `content` allows a property to hold arbitrary length binary data. Alfresco Content Services comes prepackaged with several content models. You can define new models for specific use cases from scratch or by inheriting definitions from existing models.

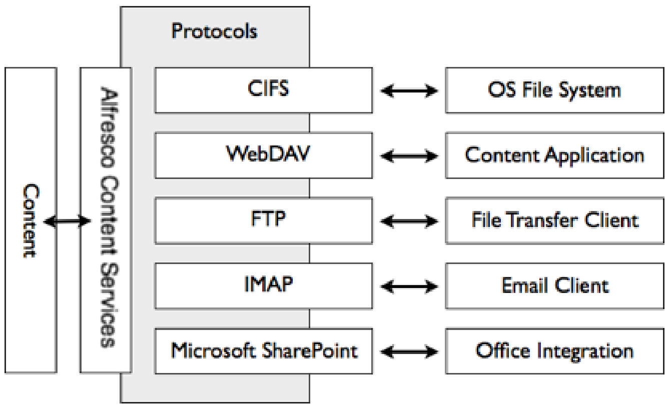
Protocols

The content application server supports many folder and document-based protocols to access and manage content held within the repository using familiar client tools.

All the protocol bindings expose folders and documents held in the repository. This means a client tool accessing the repository using the protocol can navigate through folders, examine properties, and read content. Most protocols also permit updates, allowing a client tool to modify the folder structure, create and update documents, and write content. Some protocols also allow interaction with capabilities such as version histories, search, and tasks.

Internally, the protocol bindings interact with the repository services, which encapsulate the behavior of working with folders and files. This ensures a consistent view and update approach across all client tools interacting with the content application server.

A subsystem for file servers allows configuration and lifecycle management for each of the protocols either through property files or JMX.



Supported protocols include:

CIFS (Common Internet File System)

CIFS allows the projection of Alfresco Content Services as a native shared file drive. Any client that can read and write to file drives can read and write to Alfresco Content Services, allowing the commonly used shared file drive to be replaced with an ECM system without users even knowing.

WebDAV (Web-based Distributed Authoring and Versioning)

WebDAV provides a set of extensions to HTTP for managing files collaboratively on web servers. It has strong support for authoring scenarios such as locking, metadata, and versioning. Many content production tools, such as the Microsoft Office suite, support WebDAV. Additionally, there are tools for mounting a WebDAV server as a network drive.

FTP (File Transfer Protocol)

FTP is a standard network protocol for exchanging and manipulating files over a network. This protocol is particularly useful for bulk loading folders and files into the repository.

IMAP (Internet Message Access Protocol)

IMAP is a prevalent standard for allowing email access on a remote mail server. Alfresco presents itself as a mail server, allowing clients such as Microsoft Outlook, AppleMail, and Thunderbird to connect to and interact with folders and files held within the repository. IMAP supports three modes of operation:

1. Archive: allows email storage in the repository by using drag/drop and copy/paste from the IMAP client
2. Virtual: folders and files held in the repository are exposed as emails within the IMAP client with the ability to view metadata and trigger actions using links embedded in the email body
3. Mixed: a combination of both archive and virtual

Microsoft SharePoint Protocols

Alfresco Office Services support Microsoft SharePoint Protocols. This allows Alfresco Content Services to act as a SharePoint server, creating tight integration with the Microsoft Office suite. A user who is familiar with the Microsoft task pane can view and act upon documents held within the repository. Collaborative features of Microsoft SharePoint are mapped to Alfresco Share site capabilities.

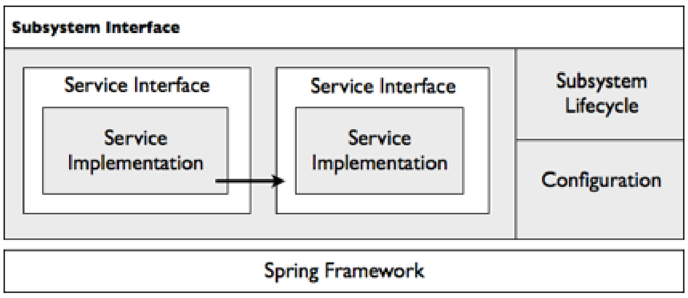
Modularity

The Alfresco Content Services system is modular. Every moving part is encapsulated as a service, where each service provides an external face in a formally defined interface and has one or more black-box implementations.

The system is designed this way to allow for:

- Pick and mix of services for building an ECM solution
- Reimplementation of individual services
- Multiple implementations of a service, where the appropriate implementation is chosen based on the context within which the solution is executed
- A pattern for extending Alfresco Content Services (at design and runtime)
- Easier testing of services

To support this approach, Alfresco Content Services used the Spring framework for its factory, Dependency Injection, and Aspect-Oriented Programming (AOP) capabilities. Services are bound together through their interfaces and configured using Spring's declarative Dependency Injection.



A service interface is defined as a Java interface. For services that form the internal embedded API for extensions, cross-cutting concerns such as transaction demarcation, access control, auditing, logging, and multi-tenancy are plugged in through Spring AOP behind the service interface. This means that service implementations are not polluted with these concerns. It also means the cross-cutting concerns can be configured independently or even switched off across the server if, for example, performance is the top-most requirement and the feature is not necessary.

Multiple services are aggregated into an Alfresco Content Services subsystem where a subsystem represents a complete coherent capability of the Alfresco Content Services server, such as authentication, transformation, and protocols. As a unit, subsystems have their own lifecycle where they can be shut down and restarted while the server is running. This is useful to disable aspects of the server, or reconfigure parts of it, such as how LDAP synchronization is mapped. Each subsystem supports its own administration interface that is accessible through property files or JMX.

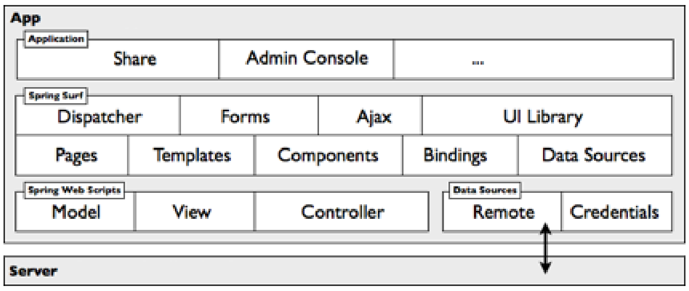
Web application framework

Alfresco Share and all new web applications are built on Surf. This web application framework provides the typical features of this kind of framework and supports web content management needs.

At the heart of Surf is a site assembly framework that bundles a full site construction object model and toolkit for building websites and applications.

Its features include:

- A Site Dispatcher to create pages easily, link them to the overall navigation of a website, and build pages in a way that promotes reusability.
- Templates for defining a page layout once and then reusing it across a large set of pages. You can develop pages using FreeMarker, JSP, HTML, or Java.
- A UI Library containing reusable UI components comprising back-end application logic and front-end presentation code that can be bound into regions (or slots) within a page or template.
- Pages that you can render in multiple formats, such as print, PDF, or mobile device.
- AJAX support for integration with the Yahoo! User Interface (YUI) library.
- Forms in a rich forms engine for rendering and collecting data.



Surf embeds Spring web scripts, allowing developers to use the same techniques that were used when building content application server RESTful APIs. Often, a Surf website requires access to and management of content held within the application content server, such as to support user-generated content, dynamic site artifacts, personalized presentation, and tagging. To support this, Surf provides the following integration services:

- Remote: encapsulates any number of data sources with out-of-the-box support for the content application server
- Credentials: manages user authentication with out-of-the-box support for the content application server

With the CMIS client API, Surf provides an open stack for implementing web-based, content-enabled applications.

Alfresco Content Services 5.2 includes the new UI framework built on Surf, Aikau. Aikau provides a modern, higher-level approach to developing custom UI applications, and features a simplified method for creating pages and widgets. New pages with standard widgets can be created through JSON code, and then extended as required using JavaScript.

Deployment options

You can deploy Alfresco Content Services in many different forms and topologies. Because its infrastructure foundation protects Alfresco Content Services from the environment within which it executes, you can choose components such as operating system, database, application server, web browser, and authentication system. It's designed to scale down as well as up.

Embedded Alfresco Content Services

An embedded Alfresco Content Services is contained directly within a host where the host communicates with Alfresco Content Services through its embedded API, meaning the host and Alfresco Content Services reside in the same process. Typical hosts include content-rich client applications that require content-oriented storage, retrieval, and services, but can also include hosts such as test harnesses and samples. A client can choose to embed the web application framework or content application server, or both, treating Alfresco Content Services as a third-party library. In any case, the client can pick and mix the services to embed, allowing very small-footprint versions. The host is responsible for the start up and shutdown of Alfresco Content Services.

Content application server

An content application server is a stand-alone server capable of servicing requests over remote protocols. A single server can support any number of different applications and clients where new applications can be arbitrarily added. Clients communicate through its Remote API and protocol bindings, although you can configure a server to omit or prohibit specific access points. This type of deployment takes advantage of an application server where Alfresco Content Services is bundled as a web application. Application server features, such as transaction management and resource pooling, are injected into the infrastructure foundation, allowing Alfresco Content Services to take advantage of them.

For example, you can embed the content application server inside Apache Tomcat for the lightest-weight deployment, as well as inside Java Enterprise Edition compliant application servers from JBoss, Oracle, or IBM to take advantage of advanced capabilities such as distributed transactions.

Clustered

To support large-scale systems, you can cluster Alfresco Content Services. This lets you set up multiple servers to work with each other, allowing client requests to be fulfilled across a number of processors. You can cluster both the web application framework and content application server, allowing each tier to scale out independently. Each node of a clustered content application server shares the same content repository store, although the store itself can be replicated across the nodes, if required. Caches and search indexes are also distributed, meaning that a clustered content application server looks and acts like a single content application server.

Typically, a load balancer is placed in front of the clustered content application server to distribute requests across the nodes. This setup also supports Cloud deployments. Alfresco Content Services provides images and tools for easily deploying a clustered content application server across multiple Amazon EC2 virtual nodes.

Backup server

This is a special case of the clustered deployment where, in case of failure, an application can switch to a backup version of the deployed stack. Depending upon configuration, the backup version might be available immediately on failure (known as *hot backup*) or shortly after failure, following some configuration changes (known as *warm backup*). One of the nodes in the cluster is designated the master, which supports the live application, while the other node is designated the slave, which keeps itself replicated with the master. The slave remains read-only until the point of switchover.

Multi-tenancy

Multi-tenancy allows a single content application server (clustered or not) to support multiple tenants, where a tenant such as a customer, company, or organization believes they are the only user of the server as they connect to a logical partition. Physically, all tenants share the same infrastructure, such as deployed nodes in a cluster and content, repository storage. However, data maintained by one tenant cannot be read or manipulated by another tenant. A deployment of this type eases administration and reduces the cost associated with maintaining many different applications and user bases, in particular when upgrading core services or performing backups, as this only needs to be done once for all tenants.

Alfresco Content Services provides administration tools for managing tenants, including the creation of tenants at runtime. In conjunction with clustering, multi-tenancy provides an ideal deployment option for the Cloud.

Access protocols

Alfresco Content Services supports a number of different protocols for accessing the content repository. Their availability extends the options available to developers, when building their own applications and extensions.

Protocols provide developers with another possible avenue for building their own applications and extensions. For example, if you are building a client application to connect with multiple repositories from multiple vendors, including Alfresco Content Services, then CMIS is a consideration. If you are building a client to connect via the SharePoint Protocol, then use the Alfresco Office Services (AOS). Protocols provide a resource for developers, in addition to the numerous other extension points and APIs built into Alfresco.

When any of these protocols are used to access or upload content to the repository, access control is always enforced based on configured permissions, regardless of what protocol that is used.

The following table list some of the main protocols supported by Alfresco Content Services and links to more detailed documentation.

Protocol	Description	Support Status
HTTP	The main protocol used to access the repository via for example the REST APIs.	Standard in Alfresco Content Services and Community Edition.
WebDAV	Web-based Distributed Authoring and Versioning is a set of HTTP extensions that lets you manage files collaboratively on web servers.	Standard in Alfresco Content Servicesand Community Edition.
FTP	File Transfer Protocol - standard network protocol for file upload, download and manipulation. Useful for bulk uploads and downloads.	Standard in Alfresco Content Services and Community.

Protocol	Description	Support Status
CIFS	Common Internet File System - allows the projection of Alfresco Content Services as a native shared drive. Any client that can read or write to file drives can read and write to Alfresco Content Services, allowing the commonly used shared file drive to be replaced with an ECM system, without users knowing.	Standard in Alfresco Content Servicesand Community Edition.
SPP	Enables Alfresco Content Services to act as a Microsoft SharePoint Server. Allows Microsoft Office users to access documents within the Alfresco repository.	Supported as part of Alfresco Office Services (AOS). Community versions have support for the older SharePoint Protocol Support.
Alfresco Office Services	Alfresco Office Services (AOS) allow you to access Alfresco Content Services directly from all your Microsoft Office applications.	Standard in Alfresco Content Services and Community Edition.
CMIS	Alfresco fully implements both the CMIS 1.0 and 1.1 standards to allow your application to manage content and metadata in an on-premise repository or Alfresco Content Services in the cloud.	Standard in Alfresco Content Services and Community Edition.
IMAP	Internet Message Access Protocol - allows access to email on a remote server. Alfresco Content Services can present itself as an email server, allowing clients such as Microsoft Outlook, Thunderbird, Apple Mail and other email clients to access the content repository, and manipulate folders and files contained there.	Standard in Alfresco Content Services and Community Edition.
SMTP	It is possible to email content into the repository (InboundSMTP). A folder can be dedicated as an email target.	Standard in Alfresco Content Services and Community Edition.

Repository concepts

It is important as a developer to have a good understanding of the fundamental concepts of Alfresco Content Services when implementing extensions. Important concepts covered include repository, nodes, stores, types, aspects and so on.

Key Concepts

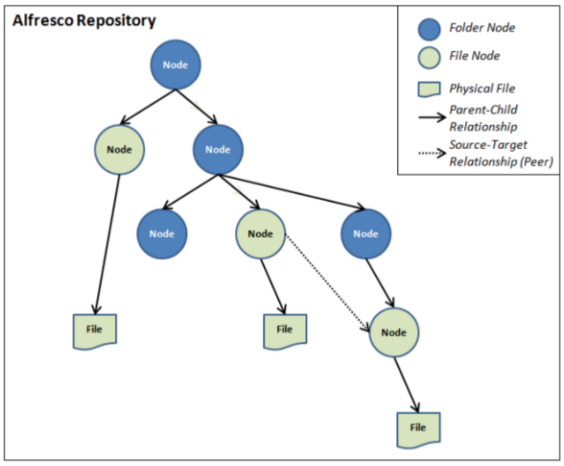
All files that are stored in Alfresco Content Services are stored in what is referred to as the **repository**. The repository is a logical entity that consists of three important parts:

1. The physical content files that are uploaded
2. The index files created when indexing the uploaded file so it is searchable
3. The metadata/properties for the file, which are stored in a relational database management system (RDBMS).

When a file is uploaded to the repository it is stored on disk in a special directory structure that is based on the date and time of the upload. The file name is also changed to the UUID (Universally Unique Identifier) assigned to the file when it is added to the repository. The file's metadata is stored in an RDBMS such as PostgreSQL. Indexes are also stored on the disk. When the file is added to the repository it is stored in a folder, and the folder has domain specific metadata, rules, and fine grained permissions. The top folder in the repository is called **Company Home**, although it will be referred to with the name **repository** in the Alfresco Share user interface.

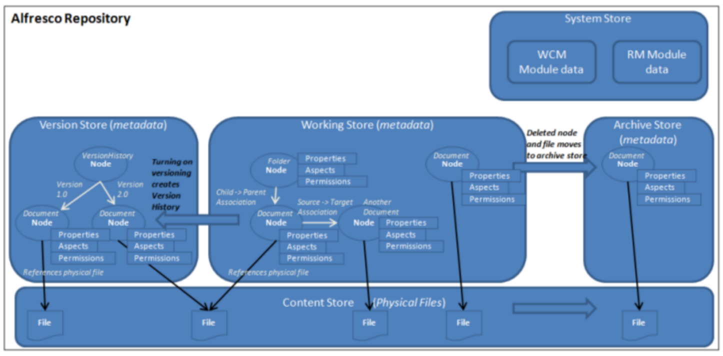
Logical Structure

All the files and folders that are uploaded and created in the repository are referred to as **nodes**. Some nodes, such as folders and rules, can contain other nodes (and are therefore known as container nodes). Nodes can also be associated with other nodes in a peer to peer relationship, in a similar fashion to how an HTML file can reference an image file. All nodes live in a **Store**. Each store has a root node at the top, and nodes can reference specific files, as shown in the following diagram:



Stores Overview

The Repository contains multiple logical stores. However, a node lives only in one store. Most of the stores are implemented as data in the connected RDBMS, only the **Content Store** is implemented so as to store items on disk:



The main stores

The **Working Store** (`workspace://SpacesStore`) contains the metadata for all active/live nodes in the Repository. This store is implemented using a database (RDBMS).

The **Content Store** contains the physical files uploaded to the Repository and is located in the `{Alfresco install dir}/alf_data/contentstore` directory on the filesystem by default, but can also be configured to use other storage systems, for example, Amazon S3. It is also possible to define content store policies for storing files on different storage systems, effectively defining more than one physical content store.

Whenever a node is deleted, the metadata for the node is moved to the **Archive Store** (`archive://SpacesStore`), which uses the configured database. The physical file for a deleted node is moved (by default after 14 days) to the `{Alfresco install dir}/alf_data/contentstore.deleted` directory, where it stays indefinitely. However, a clean-up job can be configured to remove the file at a certain point in time (referred to as eager clean-up).

When the `versionable` aspect is applied to a node, a version history is created in the **Version Store** (`workspace://lightWeightVersionStore`). Versioned node metadata is stored in the database and files remain in the `{Alfresco install dir}/alf_data/contentstore` directory. Versioning is not applicable to folder nodes.

The **System Store** is used to save information about installed Alfresco Content Services extension modules.

Content Store Implementation

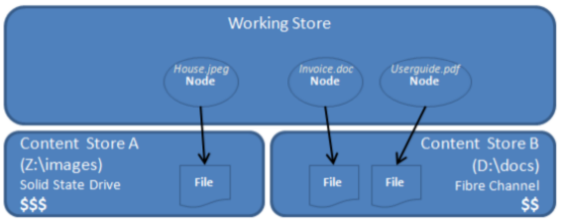
When considering file storage, it should be noted that files added to Alfresco Content Services can be of almost any type, and include images, photos, binary document files (Word, PPT, Excel), as well as text files (HTML, XML, plain text). Some binary files such as videos and music files can be relatively large. Content store files are located on the disk, rather than in the database as BLOBs. There are several reasons for this:

1. It removes incompatibility issues across database vendors.
2. The random file access support (as required by CIFS and other applications) cannot be provided by database persistence without first copying files to the file system.
3. Possibility of real-time streaming (for direct streaming of files to browser).
4. Standard database access would be difficult when using BLOBs as the most efficient BLOB APIs are vendor-specific.
5. Faster access.

Content Store Selectors

The *content store selector* provides a mechanism to control the physical location on disk for a content file associated with a particular content node. This allows storage polices to be implemented to control which underlying physical storage is used, based on your applications needs or business policies.

For example, it is possible to use a very fast tier-1 Solid State Drive (SSD) for our most important content files. Then, based on business policies that have been decided, gradually move the data, as it becomes less important, to cheaper tier-2 drives such as Fiber Channel (FC) drives or Serial ATA drives. In this way, it is possible to manage the storage of content more cost effectively:



Store Reference

When working with the APIs a store is accessed via its **store reference**, for example `workspace://SpacesStore`. The store reference consists of two parts: the protocol and the identifier. The first part (for example `workspace`) is called the protocol and indicates the content you are interested in, such as live content (`workspace://SpacesStore`) or archived content (`archive://SpacesStore`). The second part is the identifier (the type of store) for the store, such as `SpacesStore`, which contains folder nodes (previously called spaces) and file nodes data, or for example `lightWeightVersionStore` that contains version history data.

The reason some things are referred to as spaces (for example SpacesStore) is that in previous versions of Alfresco Content Services a folder used to be called a space. The Share user interface has generally been changed to use the name folder instead of the name space. However, there is functionality, such as Space Templates, that still uses the term "space". A space can simply be thought of as a folder.

Node Information

A node usually represents a folder or a file. Each store also contains a special root node at the top level with the type `sys:store_root`. The root node can have one or more child nodes, such as the Company Home folder node. Each node has a primary path to a parent node and the following metadata:

- **Type**: a node is of one type, such as Folder, File, Marketing Document, Rule, Calendar Event, Discussion, Data List and so on.
- **Aspects**: a node can have many aspects applied, such as Versioned, Emailed, Transformed, Classified, Geographic and so on.
- **Properties**: both types and aspects define properties. If it is a file node then one of the properties points to the physical file in the content store.
- **Permissions**: access control settings for the node.
- **Associations**: relationships to other nodes (peer or child).

Node Reference

A node is uniquely identified in the Repository via its **node reference**, also commonly referred to as *NodeRef*. A node reference points to a store and a node in that store. A node reference has the following format: `{store protocol://store identifier}/UUID` such as for example `workspace://SpacesStore/986570b5-4a1b-11dd-823c-f5095e006c11`. The first part is the store reference and the second part is a Universally Unique Identifier (UUID) for that store. Node references are used a lot in the available APIs so it is good to have an idea of how they are constructed.

Node Properties

The node properties, also referred to as the node's **metadata**, contains the majority of the information for a node. The `sys:store-protocol`, `sys:store-identifier`, and `sys:node-uuid` properties contains all the data needed to construct the NodeRef, uniquely identifying the node. The special property called `cm:content` points to where the physical content file is stored on disk (unless it is a folder or other contentless node type). All properties are either contained in a type or in an aspect defined in a content model. When a node is created some properties are automatically set by the system and cannot be easily changed, they are called **audit properties** (from the `cm:auditable` aspect) and are Created Date, Creator, Modified Date, Modifier, and Accessed. Defining new domain specific node properties, together with the types and aspects that contain them, is the primary way of classifying a node so it can be easily found via searches.

Metadata/Property Extractors

Some of the properties of a file node are set automatically when it is uploaded to the Repository, such as *author*. This is handled by **metadata extractors**. A metadata extractor is set up to extract properties from a specific file MIME type. There are numerous metadata extractors available out-of-the-box covering common MIME types such as MS Office document types, PDFs, Emails, JPEGs, HTML files, DWG files and more. The metadata extractors are implemented via the Tika library, although custom metadata extractors are available. Each metadata extractor implementation has a mapping between the properties it can extract from the content file, and what content model properties that should be set as node metadata.

Node Associations

There are two types of associations:

- **Parent** to **Child** associations - these are for example folder to file associations where deleting the folder will cascade delete its children.
- **Peer** to **Peer** - an example could be article to image associations where deleting the article does not affect the related image node(s). These associations are also referred to as source to target associations.

QName

All properties are defined within a specific content model, which also defines a unique **namespace**. For example, a property called `description` can be part of many namespaces (content models). To uniquely identify what `description` property is being referenced, a fully qualified name, or a `QName`, is used. A QName has the following format: `{namespace URL}property local name`, for example:

```
{http://www.alfresco.org/model/content/1.0}description
```

The first part in curly braces is the namespace identifier defining the content model that the property is part of. The second part is the local name of the property (that is *description* in this case).

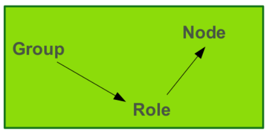
A QName is used for types, aspects, properties, associations, constraints and so on. The QName for the generic folder type that is part of the out-of-the-box document content model is `cm:folder`. Note the use of `cm` to denote the namespace. Each content model defines a prefix for each namespace that is used in the content model. Each type, aspect, property and so on in the content model definition uses the namespace prefix instead of the full namespace URL. You will also use the prefix when referring to entities such as types, aspects, properties, in the different APIs.

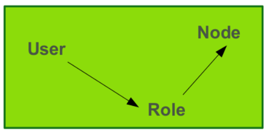
Permissions

Permissions are set up per node and a node can inherit permissions from its parent node. A Role (Group) Based Access Control configuration is the preferred way to set up permissions in the repository. However, permissions can also be set for an individual user. Groups and users can be synchronized with an external directory such as LDAP or MS Active Directory. Some groups are created automatically during installation:

- **EVERYONE** – all users in the system
- **ALFRESCO_ADMINISTRATORS** – administrators with full access to everything in the Repository.
- **ALFRESCO_SEARCH_ADMINISTRATORS** – can access the Search Manager tool and set up search filters (facets).
- **SITE_ADMINISTRATORS** – can access the Site Manager tool and change visibility of sites, delete sites, and perform site related operations.
- **E-MAIL_CONTRIBUTORS** – users that can send email with content into Alfresco Content Services.

Permission settings involve three entities:





There are a number of out-of-the-box roles:

1. Consumer
2. Contributor
3. Editor
4. Collaborator
5. Coordinator

Whenever a Share site is created there are also four associated groups created that are used to set up permissions within the site. In the repository, groups are prefixed with `GROUP_` and roles with `ROLE_`, this is important when referring to a group or role when using one of the APIs.

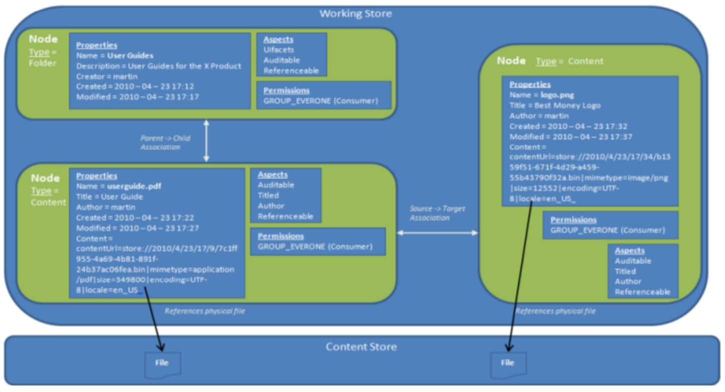
A **Site** is a collaboration area in Alfresco Share where a team of people can collaborate on content.

Owner

The Repository contains a special authority called owner. Whoever creates a node in the repository is called the owner of the node. Owner status overrides any other permission setting. As owner you can do any operation on the node (basically the same as being coordinator/admin). Anyone with Coordinator or Admin status can take ownership of a node (`cm:ownable` is then applied).

Folder Node and File Node Overview

The diagram illustrates a typical folder node with a child file node when it has been classified with the out-of-the-box default document content model:



Properties
Name = **User Guides**
Description = User Guides for the X Product
Creator = martin
Created = 2010 – 04 – 23 17:12
Modified = 2010 – 04 – 23 17:17

Aspects
Uifacets
Auditable
Referenceable

Permissions
GROUP_EVERONE (Consumer)

Properties
Name = **userguide.pdf**
Title = User Guide
Author = martin
Created = 2010 – 04 – 23 17:22
Modified = 2010 – 04 – 23 17:27
Content = contentUrl=store://2010/4/23/17/9/7c1ff955-4a69-4b81-891f-24b37ac06fea.bin|mimetype=application/pdf|size=349800|encoding=UTF-8|locale=en_US_

Aspects
Auditable
Titled
Author
Referenceable

Permissions
GROUP_EVERONE (Consumer)

Properties
Name = **logo.png**
Title = Best Money Logo
Author = martin
Created = 2010 – 04 – 23 17:32
Modified = 2010 – 04 – 23 17:37
Content = contentUrl=store://2010/4/23/17/34/b1359f51-671f-4d29-a459-55b43790f32a.bin|mimetype=image/png|size=12552|encoding=UTF-8|locale=en_US_

Permissions
GROUP_EVERONE (Consumer)

Aspects
Auditable
Titled
Author
Referenceable

Mini glossary

Terms and concepts used when developing for Alfresco Content Services.

Term	Description
Actions	Actions typically work in conjunction with Rules. When creating a rule you specify the action to be carried out when the rule is activated. There are standard actions, but you can also create custom actions. Custom actions are implemented in Java as Spring beans.

Term	Description
Aspects	While nodes must be of a single Type, they can have multiple Aspects applied to them. Dublin Core, Taggable, EXIF, Geographic, Emailed are all examples of aspects. Also a single aspect can be applied to multiple types.
Associations	Relationships between Types are modeled with Associations. There are two types: Child Associations and Peer Associations.
Attributes	Attributes provide a global means of storing key-value data. Whereas properties are attached to a node, attributes are system-wide, and not stored per-node. They can be quickly searched for without the need for an index and are cluster-safe.
Auditing	Auditing allows you to track events that occur in Alfresco Content Services. The events that you audit are completely configurable.
Configuration	Platform provides many points at which the configuration of the system can be changed. For example, changes may be made to `alfresco-global.properties` or many of the other configuration files. In addition, Share is highly configurable.
Content	This is the piece of content to be stored in the repository. It could be a Word document, a PDF, a PNG image file, an audio file and so on. Note that the content itself will be stored on the file system, while its corresponding node, containing metadata, will be stored in an RDBMS.
Content Model	The content model describes the nodes and the hierarchical relationship between them, as well as any constraints that may exist. For example, nodes that are of container type can contain other nodes.
Content Renditions	Renditions are manipulations of content that typically involves some content transformation, followed by some other operation such as crop or resize. For example, you might have a PDF document, which you might convert the first page of to a PNG, and then crop and resize that image to create a thumbnail view of the document. The key service is the Rendition Service.
Content Store	The repository has multiple content stores. Typically this would include a main content store, an encrypted content store, an archive content store (for deleted items), and a version store (to hold previous versions of documents). It is also possible for developers to create custom content stores for specific purposes.

Term	Description
Content Transformation	Content transformation transforms one format of content into another. There are numerous applications of this, such as converting content into plain text for indexing and generating PDF versions of Word files. Transformations can be chained together, for example DOC to PDF using LibreOffice. Key service is the ContentTransformation Service.
Extension	Extensions can be thoughts of as server-side additions to Alfresco Content Services. There are two main types of extension: Platform and Share. Each of these extension types are fully described in this documentation, along with all officially supported extension points.
Events	Data structures created on various changes within the repository, such as name change of a piece of content. There are three types of Event: 1. Inbound event - content arriving into a folder 2. Outbound event - content leaving a folder 3. Update event - content being modified
Indexing	As content is added to Alfresco Content Services it is indexed by an indexer such as Solr. Solr indexes both meta data and the plain text content of files added. The content model defines the metadata (aspects, properties, types, associations) that are to be indexed via the `<tokenise>` element. The indexes can be queried using a variety of query languages, including: • fts-alfresco • storeroot • noderef • xpath • lucene • cmis-strict • cmis-alfresco • db-afts • db-cmis Queries can be executed from JavaScript and Java code, and also in the Node Browser (available under Admin Tools in Share).

Term	**Description**
Nodes	Each piece of content in the repository has a node associated with it. The node contains information about the content, such as its metadata and location within the content store. The node is stored in a RDBMS such as PostgreSQL, the content itself is stored on the file system.
Predefined Content Model	There are pre-defined content models provided out-of-the-box, these include Folder/Document hierarchy, Dublin Core, blogs, Wiki, Sites.
Policies	These are event handlers triggered by certain node events for either all nodes or just nodes of a specific type.
Metadata	Most content has metadata associated with it. For example, photographs have EXIF metadata. Word documents would have Author, Creation Date, and so on. The metadata provides very useful information for document discovery, without the overhead of having to extract and process the full content of a document.
Metadata extraction	Content type (mimetype) can automatically be identified for the standard types by Tika. This metadata can be extracted from the content and copied into the content's associated node (properties). For custom content types it is possible to create Custom metadata extractors. Key service is the MetadataExtractor Service. You can also create custom metadata extractors.
Mimetypes	The mimetype essentially identifies the type of content. Alfresco Content Services can automatically identify content types and establish mimetype (using Tika). It is also possible to create custom content identification through custom mimetypes.
Module	A module is the format in which an *extension* is packaged.
Property	Properties are named items of metadata associated with a Type. Each Property is of a specific data type (such as Text or Date). Constraints can be applied to restrict the value of a Property.
Repository	This is where content is stored, and can be thought of as the content stores and all the related services. It consists of the filesystem or storage service where the content is stored and a database containing metadata. See Repository Concepts for an overview.

Term	Description
Rules	Declarative definition of processes based on addition, update, or removal of nodes with respect to folders (think email rules for content). These are set up for a folder in Share. See documentation and videos on applying rules to folders. Note that Rules can be filtered based on conditions/criteria: • All items (no filter) • Items with a specific mimetype (for example `.doc`, `.pdf`) • Contained in a category • Specific content type applied to a specific aspect file name pattern (for example `*-context.xml`) Boolean NOT can be used (for example not `.pdf`). There are no limits to the number of conditions that can be applied to each Rule.
Type	A node is always of a single Type. A Type is similar to a class in Object-Oriented Programming, Types can be inherited from a parent Type in the content model.

Alfresco SDK 2.2.0

This documentation covers version 2.2.0 of the Alfresco SDK. The Alfresco SDK is a Maven based development kit that provides an easy to use approach to developing applications and extensions for Alfresco.

With this SDK you can develop, package, test, run, document and release your Alfresco extension project. It is important to note that while previous versions of the Alfresco SDK were based around Ant, the latest versions of the SDK are based on Maven.

Links to documentation for previous versions are available on this page.

This is the PDF version of this documentation and is not complete. Please refer to Alfresco Documentation online for access to the complete set of documentation.

The Alfresco SDK 2.2 includes a number of Maven archetypes. These archetypes aim to provide a standardized approach to development, release, and deployment of Alfresco extensions.

Project documentation, website, and support

From version 2.0 and up, releases for the Alfresco SDK are available in Maven Central.

The Alfresco SDK source code is hosted on GitHub.

The Alfresco SDK issue tracking is hosted on GitHub.

The Alfresco SDK forum for support is hosted on Alfresco Forums.

What's new?

New features in this version of the Alfresco SDK.

What's new in Alfresco SDK 2.2.0

- Support for Alfresco 5.1
- Alfresco REST API Explorer is now available with the `run` profile (http://localhost:8080/api-explorer)
- All-In-One AMP module names and ID now includes AIO name to make them unique
- Aligned Module version with Artifact version - 5.0 and 5.1 supports SNAPSHOT versions in `module.properties`
- Numerous bug fixes. See the release notes for details.
- Issues: you will have to run without Spring Loaded with AIO and Repo AMP, only works with Share AMP at the moment.

What's new in Alfresco SDK 2.1.1

- Reintroduce support for Java 7
- Numerous bug fixes. See the release notes for details.

What's new in Alfresco SDK 2.1

- Support for Solr 4
- Regression testing Share Webapp with Alfresco Share Page Object (PO) tests
- Functional testing of Custom Pages with Alfresco Share PO
- Rapid Application Development (RAD) improvements:
  - New `alfresco:refresh-repo` goal for alfresco-maven-plugin for automatic refresh of repository (`alfresco.war`)
  - New `alfresco:refresh-share` goal for alfresco-maven-plugin for automatic refresh of Share (`share.war`)
- Automatic JavaScript compression
- Custom package name can be used when generating projects
- Records Management profile 'rm' removed, now included as other AMPs in the repository and Share WAR POMs
- Run script for Windows now available (`run.bat`)
- Upgraded to Alfresco Community Edition 5.0.d and Enterprise 5.0.1 (JDK8)
- Fixed DB initialization error (dbObject cannot be null)
- Fixed blank Admin Console bug

What's new in Alfresco SDK 2.0

- Support for Alfresco 5.0
- Name changed from Maven Alfresco SDK to Alfresco SDK.
- New Share AMP archetype.
- RAD in Eclipse and IntelliJ.
- Now requires Maven 3.2.2
- RAD with Spring Loaded.
- Support for JRebel has been deprecated.
- Introduction of enterprise profile option -Penterprise.

What was new in Alfresco SDK 1.1.1

- Bug fixes.

What was new in Alfresco SDK 1.1.0

- Runs in Tomcat7 (replaces Jetty)
- Remote running of JUnit
- JRebel integration
- Improved IDE integration

What was in Alfresco SDK 1.0.x

- Use of Alfresco POMs
- Relied on a number of components:
    - The SDK parent POM providing full Alfresco project lifecycle feature, to be added as a `<parent>` in your projects
    - Archetypes like the AMP or all-in-one providing sample project to kickstart your Alfresco development and boost it with best practices
    - The Alfresco Maven Plugin to manage AMP packaging and dependencies
    - Alfresco Platform Distribution POM can (optionally) be used to provide centralized <dependencyManagement> on a particular Alfresco version / edition (Community Edition / Enterprise)
    - The Alfresco Artifacts Repository provides backing for this SDK. Check the Alfresco Wiki for Community Edition / Alfresco One access information
- Embedded Jetty server and H2 database

What was in Maven Alfresco Lifecycle

- First implementation of Alfresco Maven SDK (**Now deprecated**)
- No use of Alfresco POMs
- Available archetypes and plugins:
    - `maven-alfresco-extension-archetype` to create WAR packaged webapps that can provide all Maven lifecycle and features
    - `maven-alfresco-share-archetype` to create and manage Alfresco Share customization webapps
    - `maven-alfresco-share-module-archetype` to create and manage Alfresco Share custom dashlets, pages as JARs
    - `maven-alfresco-amp-archetype` to create `maven-amp-plugin` managed web apps, which can provide all Maven lifecycle and features to Alfresco modules. The `maven-amp-plugin` is also used as a replacement to MMT to unpack AMPs into WARs builds, using the Maven dependency mechanism provided by the `maven-amp-plugin`
- Embedded Jetty server and H2 database to run Alfresco or Share
- Possible to use Maven standard dependency management to pull in AMPs in your build
- Further information

Introduction to the Alfresco SDK

You will learn about the different Maven Archetypes that can be used to generate Alfresco extension projects.

You will learn which SDK version is compatible with which Alfresco version, and you will find links to some useful community resources.

Introduction to Maven archetypes

There are three Maven archetypes that can be used to generate Alfresco extension projects.

The following project types, and archetypes, are available:

- *Alfresco Repository AMP*: this archetype is used to create extensions for the Alfresco Repository Web Application (`alfresco.war`) in the form of Alfresco Module Packages (AMP).
- *Alfresco Share AMP*: this archetype is used to create extensions for the Alfresco Share Web Application (`share.war`) in the form of AMPs.
- *Alfresco all-in-one (AIO)*: this archetype is a multi-module project that leverages the Alfresco SDK's powerful capabilities to customize and run the full Alfresco platform embedded with all its components. The archetype does not require additional downloads and provides a perfect starting point for full-blown Alfresco projects.

You can view these archetypes when you obtain a list of archetypes from Maven Central:

```
mvn archetype:generate -Dfilter=org.alfresco:
```

Note the use of a filter to display only archetypes in the namespace `org.alfresco`.

Repository AMP archetype

The Alfresco Repository (Repo) AMP Archetype generates a sample project for managing Alfresco Repository extensions/customizations. These extensions are packaged as Alfresco Module Packages (AMP).

This archetype should be used to extend the Alfresco Repository web application (`alfresco.war`).

The following are typical use-cases for when this archetype should be used:

- You work in a bigger team and want to develop, tag, and release a Repo module separately from the main Alfresco Extension project (All-in-One) that it is included in.
- You want to add, and contain, an extra Repo module in an All-In-One Project (useful when you don't have a Nexus artifacts repo to which you can release individual repo AMPs).
- You intend to build an Add-On, Component, Module etc that should be distributed independently.

If you intend to build an extension for the Alfresco Share web application, use the Share AMP archetype instead.

The main features of this archetype are:

- AMP packaging - the supported packing type for Alfresco extensions.
- AMP dependency management in Maven.
- Installation of AMPs into an Alfresco WAR.
- Sample repository web script demonstrating how to implement a custom REST-based API.
- Content Model Skeleton XML file ready to be filled in with your domain specific content model.
- AMP Unit Testing support. Just run the standard `mvn test` and see your `src/test/java` Alfresco unit tests run. An sample Unit Test is provided in this archetype.
- Run embedded in Tomcat with an embedded H2 database for demo purposes (-Pamp-to-war), rapid application development and integration testing.

This is not a supported stack, so it should only be used for development purposes.

- Support for (remote) Junit and integration testing and Rapid Application Development. This uses spring-loaded. Project can easily be launched for this scenario using `run.sh`.
- Easy to integrate with an IDE environment such as Eclipse and IntelliJ IDEA.

Share AMP archetype

The Alfresco Share AMP Archetype generates a sample project for managing Alfresco Share extensions/customizations. These extensions are packaged as Alfresco Module Packages (AMP).

This archetype should be used to extend the Alfresco Share web application (`share.war`).

The following are typical use-cases for when this archetype should be used:

- You work in a bigger team and want to develop, tag, and release a Share UI module separately from the main Alfresco Extension project (All-in-One) that it is included in.
- You want to add, and contain, an extra Share UI module in an All-In-One Project (useful when you don't have a Nexus artifacts repo to which you can release individual Share UI modules).
- You intend to build an Add-On, Component, Module etc that should be distributed independently.

If you intend to build an extension for the Alfresco Repository web application, use the Repository AMP archetype instead.

The main features of this archetype are:

- AMP packaging - the supported packing type for Alfresco extensions.
- AMP dependency management in Maven.
- Installation of AMPs into an Share WAR.
- Sample Aikau page and widget to demonstrate how to develop new pages for the Alfresco Share UI.
- AMP Unit Testing support. Just run the standard `mvn test` and see your `src/test/java` Alfresco unit tests run. An sample Unit Test is provided in this archetype.
- Run embedded in Tomcat for demo purposes (-Pamp-to-war), rapid application development and integration testing.

  Requires a running Alfresco Repository on localhost:8080.
- Easy to integrate with an IDE environment such as Eclipse and IntelliJ IDEA.

All-in-One archetype

The Alfresco All-in-One (AIO) Archetype is a multi-module project, leveraging Alfresco SDK's powerful capabilities to customize and run the full Alfresco platform embedded with all its components. The archetype does not require additional downloads, such as an Alfresco installer, and provides a perfect starting point for full-blown Alfresco projects where the final artifacts should be the customized `alfresco.war` and `share.war`.

The following are typical use-cases for when this archetype should be used:

- You are going to start on a project for a client and need an Alfresco extension project that can produce the final customized Alfresco WAR and Share WAR artifacts.
- Your project needs access to the full regression testing suite for the Alfresco Share UI.

- Your project needs access to the functional testing based on the Alfresco Share Page Object (PO) library.
- When testing with the RAD features you need Solr to be running.

Note that if you are going to develop an addOn, reusable component, module, and so on, that should be distributed independently, then have a look at the AMP projects instead. For Alfresco repository extensions see Repository AMP and for Alfresco Share extensions see Share AMP.

The main features of the AIO archetype are:

- AMP packaging for repository and share extensions - the supported packing type for Alfresco extensions.
- AMP dependency management in Maven.
- Automatic installation of AMPs into Alfresco WAR and Share WAR.
- Easy to include extra AMPs and have them included in the WARs.
- Out-of-the-box Alfresco extensions such Records Management (RM), SharePoint Protocol (SPP), Media Management etc easily included in the same way as custom AMPs for consistency.
- AMP Unit Testing support. Just run the standard `mvn test` and see your `src/test/java` Alfresco unit tests run. An sample Unit Test is provided in this archetype.
- Alfresco Share Regression Testing - you don't have to write tests to protect against regression in out-of-the-box Share UI functionality, just use the -Prun,regression-testing profiles
- Custom Functional Testing - Utilize the Alfresco Share Page Objects (PO) to write your own custom web page testing (example test included), use the -Prun,functional-testing profiles
- Run a full Alfresco stack (that is, `alfresco.war`, `share.war`, `solr4.war`) embedded in Tomcat using the H2 database for demo purposes (-Prun), rapid application development and integration testing.

  This is not a supported stack, so it should only be used for development purposes.
- Support for (remote) Junit and integration testing and Rapid Application Development. This uses spring-loaded. Projects can easily be launched for this scenario using `run.sh`.
- Seamless IDE integration with Eclipse and IntelliJ IDEA.

Compatibility matrix

Alfresco SDK has several versions and compatibility with Alfresco versions varies.

It is recommended you use the latest version of the Alfresco SDK where possible.

The following table shows compatibility between Alfresco SDK and versions of Alfresco.

Alfresco version	**Maven Alfresco Lifecycle (deprecated**	**Maven SDK 1.0.x (deprecated**	**Maven SDK 1.1.x**	**Alfresco SDK 2.0.x**	**Alfresco SDK 2.1.x**	**Alfresco SDK 2.2.x**
3.2.2 - 4.1.1.x	Compatible (but not supported)	Not available	Not available	Not available	Not available	Not available

Alfresco version	Maven Alfresco Lifecycle (deprecated	Maven SDK 1.0.x (deprecated	Maven SDK 1.1.x	Alfresco SDK 2.0.x	Alfresco SDK 2.1.x	Alfresco SDK 2.2.x
4.1.x (x >= 2)	Not available	Compatible (but not supported)	Not available (SDK 1.1.0 does not work with Alfresco 4.1.2-4.1.5 using Solr Search Subsystem. It is possible to use Alfresco 4.1.6 and greater, or use Lucene Search Subsystem)	Not available	Not available	Not available
4.2.x	Not available	Not available	Compatible and supported	Not available	Not available	Not available
5.0 and 5.0.c	Not available	Not available	Not available	Compatible and supported	Not available	Not available
5.0.1+ and 5.0.d+	Not available	Not available	Not available	Compatible and supported	Compatible and supported	Not available
5.1+ and 5.1.d+	Not available	Not available	Not available	Not available	Not available	Compatible and supported

Note that Alfresco 4.1.x requires Java 6, Alfresco 4.2.x and Alfresco 5.0 require Java 7. Alfresco 5.0.1 and 5.0.d requires Java 7 or 8. Alfresco One 5.2 and 5.1.d requires Java 7 or 8. Note also that Alfresco SDK works only on Linux, Windows or Mac.

Community resources

Use the community resources that are available to help you master the Alfresco SDK.

Some community resources that are worth trying out:

Link	Description
Alfresco SDK on GitHub	Where you can clone the code, submit issues and read community documentation.
Order of the Bee	Quoting from the site: We are an independent organization of the Alfresco community. We are here to promote Alfresco Community Edition and aggregate the best from the community for you.
Mind the Gab	News, thoughts and tutorials from one of the main developers behind the Alfresco SDK.
Getting Started with Alfresco SDK 2	Great article by Jeff Potts about how to get going with SDK 2.
Alfresco SDK 2.0 Deep Dive	Article that takes a look at the "behind the scenes" stuff used by the Alfresco SDK 2.0.

Link	Description
Ole Hejlskov's Alfresco SDK 2.0 Beta screencast	Screencast showing use of the three main archetypes of the SDK in Eclipse. Also demonstrates some important features of the SDK such as hot reloading of code.

Getting Started with the Alfresco SDK

This information gets you started with the Alfresco SDK. The Alfresco SDK itself does not need to be installed (as it is based around Maven), but there are some prerequisites that will need to be made available.

This information is split into three areas:

- Installing software that is a prerequisite for running the SDK successfully
- Getting the SDK working from the command line
- Getting the SDK working from IDEs

Before you begin

There are some points you need to be aware of before you start using the Alfresco SDK.

Some things to do and note before you start:

1. Check the What's New page to orientate yourself to the latest features of the SDK.
2. Check the compatibility matrix to make sure that you have the right version of the SDK for your version of Alfresco.
3. Make sure that you understand the different types of Maven archetypes that are available. Check the Maven Archetypes summary to make sure that you use the right one for your extension.

Installing and configuring software

Use this information to install and configure the software on which the SDK depends.

You'll need to download and install the following tools and libraries, if you don't already have them:

1. The Oracle Java Software Development Kit (JDK) version 8
2. Apache Maven 3.2.5+
3. Spring Loaded (**Note**. only works with the Share AMP archetype at the moment).

Install Spring Loaded

The Alfresco SDK's Rapid Application Development (RAD) features uses Spring Loaded.

There are no pre-requisites for this installation. (**Note**. ONLY works with the Share AMP archetype at the moment).

Spring Loaded is a Java agent (represented by a JAR file) that enables class reloading in a running JVM. It will enable you to update a Java file in your Alfresco extension project and then see the effect of the change directly in a running Alfresco-Tomcat-JVM instance without having to re-build JARs, AMPs, and WARs and re-deploying them, saving you loads of time.

1. Download the Spring Loaded JAR from here.
2. Copy JAR to some directory.

   There is no specific installation needed, just copy the JAR to a permanent place where you can refer to it.

You now have the Spring Loaded JAR readily available in a directory.

Install JDK

The Alfresco SDK is based on Maven, which requires the JDK to be installed. This topic steps you through installing the JDK and verifying its installation.

There are no pre-requisites for this installation.

To use the Alfresco SDK most effectively, and to align with what JDK is used by the default Alfresco versions in the SDK, you need to have Oracle JDK 1.8 installed. Maven requires that the JDK be installed - the Java run-time alone is not sufficient.

Checking for the availability of the JDK.

1. Check if you have the JDK already installed. Go to your command line and type the following command:

   ```
   javac -version
   ```

   You will see a message such as the following, if you have the JDK installed:

   ```
   javac 1.8.0_45
   ```

   Make sure you use `javac` when you test if JDK is installed and not `java`, which tests if JRE is installed.

   If you get "command not found" you need to install the JDK. Also if you have a version of the JDK prior to 1.8 you will need to install 1.8 or above. It is possible to have multiple versions of the JDK installed (and multiple Java run-times). You will later see how you can configure your `JAVA_HOME` variable so that Maven uses the correct version of the JDK.

Downloading the JDK.

2. Download the JDK from the Oracle JDK site.

Installing the JDK.

3. Carefully review the Oracle JDK 8 installation guide as appropriate for your system.
4. Install the JDK, following the Oracle instructions.

Verifying the JDK is successfully installed.

5. Go to your command line and type the following command:

   ```
   javac -version
   ```

   This will display information such as the following:

   ```
   javac 1.8.0_45
   ```

   To be extra sure you should also check your Java run-time by entering the following command:

   ```
   java -version
   ```

   This will display information such as the following:

   ```
   java -version
   java version "1.8.0_45"
   Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
   Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)
   ```

Double check that the version of Java installed is correct (1.8).

You now have JDK 1.8 installed and you have verified that you are running the correct version of Java.

Setting JAVA_HOME

Before using the Alfresco SDK, you need to set your `JAVA_HOME` environment variable to a suitable value, using the correct mechanism for your operating system.

Setting the `JAVA_HOME` environment variable ensures that the correct JDK is accessed. This is especially important where you have multiple JDKs installed on your system.

1. On Mac OS X you can edit your `.bash_profile` file and add something similar to the following (the exact version you are using may vary):

   ```
   export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/
   Contents/Home
   ```

   Restart the terminal session or run `source .bash_profile` to activate the environment variable.

   Note that the actual value you specify here will depend on which JDK you have installed, and the resultant directory name.

2. On Linux you can edit your `.bashrc` file and add something similar to the following:

   ```
   export JAVA_HOME=/usr/lib/jvm/java-8-oracle
   ```

   Restart the terminal session or run `source .bashrc` to activate the environment variable.

   Note that the actual value you specify here will depend on which JDK you have installed, and the resultant directory name.

3. On Windows, the exact procedure for setting environment variables varies depending on the version of Windows you are running. For example, the procedure for Windows XP can be found in the Microsoft Knowledgebase.

   Note that the actual value you specify here will depend on which JDK you have installed, and the resultant directory name.

4. Ensure that the `JAVA_HOME` environment variable is set correctly, using a method suitable for your system. For example, on Mac OS X and Linux you can enter the following command:

   ```
   $ env |grep JAVA_HOME
   JAVA_HOME=/usr/lib/jvm/java-8-oracle
   ```

   You will see the value that `JAVA_HOME` has been set to.

   Ensure that the result matches the value you specified in your shell configuration file (such as `.bashrc`).

   If you are on Windows you can use a command such as `SET J` to display environment variables starting with 'J'.

Your `JAVA_HOME` environment variable is now set, and you have verified it is reflected in your environment.

Install Maven

The Alfresco SDK is now based around Maven (formerly it used Ant). To use the Alfresco SDK you need to have Maven installed.

To be able to use Maven you need to have a suitable JDK installed. For this version of the SDK you should have JDK 1.8 installed.

To use the Alfresco SDK you need to have Maven installed. *The version required is 3.2.5 and above*.

Check for the availability of Maven.

1. First, check to see if you already have the correct version of Maven installed. On your command line enter the following command:

   ```
   mvn --version
   ```

   If you get "command not found", or you have a version of Maven less than 3.2.5, you will need to install Maven or upgrade to 3.2.5 or above. In this case it is recommended you download the latest version of Maven (3.2.5+) from the official Maven website.

Downloading Maven.

2. Download Maven from the Apache Maven project web site.

Installing Maven.

3. Carefully review the platform-specific installation instructions in the Installing Maven Sonatype documentation.
4. Install Maven using the platform-specific instructions provided in the Maven documentation.

Verifying Maven is correctly installed.

5. Run the following command to verify Maven is correctly installed:

   ```
   mvn --version
   ```

   This will display information such as the following:

   ```
   Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06;
    2015-04-22T12:57:37+01:00)
   Maven home: /home/martin/apps/apache-maven-3.3.3
   Java version: 1.8.0_45, vendor: Oracle Corporation
   Java home: /usr/lib/jvm/java-8-oracle/jre
   Default locale: en_GB, platform encoding: UTF-8
   OS name: "linux", version: "3.13.0-58-generic", arch: "amd64", family:
    "unix"
   ```

   Check that the correct versions of Maven and the JDK are being used. If Maven is *not* using the correct version of the JDK, make sure you have set your `JAVA_HOME` environment variable, as described in the previous tutorial.

You have now installed Maven and verified that it is the correct version and is using the correct version of the JDK.

Setting MAVEN_OPTS & M2_HOME

Before using the Alfresco SDK, you need to set your `MAVEN_OPTS` and `M2_HOME` environment variables to suitable values using the correct mechanism for your operating system.

Setting `M2_HOME` specifies the home of Maven and is used by the script `mvn` (or `mvn.bat` on Windows). `MAVEN_OPTS` is used to configure a bit of extra memory for Maven as it will run an embedded Apache Tomcat application server with Alfresco Repo, Share, and Solr web applications deployed. It also sets the Spring Loaded Java Agent so it is available during Rapid Application Development (RAD).

Setting Variables.

1. On **Mac** OS X you can edit your `.bash_profile` file and add the following:

   ```
   export M2_HOME=/home/martin/apps/apache-maven-3.3.3
   export MAVEN_OPTS="-Xms1024m -Xmx1G -XX:PermSize=1024m -noverify"
   ```

   Spring Loaded ONLY works with the Share AMP archetype at the moment. If you are configuring `MAVEN_OPTS` to run a Share AMP project set `MAVEN_OPTS="-Xms1024m -Xmx1G -XX:PermSize=1024m -javaagent:/home/martin/libs/springloaded-1.2.5.RELEASE.jar -noverify"`

   Remove `-XX:PermSize=1024m` if you are using JDK 8, it is only needed for JDK 7.

   Refer to previous installation sections for in what directory Maven was installed and in what directory Spring Loaded was installed.

   Restart the terminal session or run `source .bash_profile` to activate the environment variables.
2. On **Linux** you can edit your `.bashrc` file and add the following:

   See step 1 for Mac OS, do the same thing for Linux.

   Restart the terminal session or run `source .bashrc` to activate the environment variable.
3. On **Windows**, the exact procedure for setting environment variables varies depending on the version of Windows you are running. For example, the procedure for Windows XP can be found in the Microsoft Knowledgebase.

   ```
   set M2_HOME=C:\Tools\apache-maven-3.3.1
   set MAVEN_OPTS=-Xms256m -Xmx1G -XX:PermSize=1024m -noverify
   ```

   Spring Loaded ONLY works with the Share AMP archetype at the moment. If you are configuring `MAVEN_OPTS` to run a Share AMP project set `MAVEN_OPTS=-Xms256m -Xmx1G -XX:PermSize=1024m -javaagent:C:\Tools\spring-loaded\springloaded-1.2.5.RELEASE.jar -noverify`

   Remove `-XX:PermSize=1024m`.

   If the path to the Spring Loaded JAR contains spaces, then you might need to double quote it like `-javaagent:"C:\My Tools\spring-loaded\springloaded-1.2.5.RELEASE.jar"`. Refer to previous installation sections for in what directory Maven was installed and in what directory Spring Loaded was installed.

   Restart the Windows terminal/console session.

Verifying Variables.

4. Ensure that the `MAVEN_OPTS` and `M2_HOME` environment variables are set correctly, using a method suitable for your system. For example, on Mac OS X and Linux you can enter the following command:

   ```
   $ env|egrep "M2|MAV"
   MAVEN_OPTS=-Xms256m -Xmx1G -XX:PermSize=1024m -noverify
   M2_HOME=/home/martin/apps/apache-maven-3.3.3
   ```

   Ensure that the result matches the value you specified in your shell configuration file (such as `.bashrc`).

   If you are on Windows you can use a command such as `set M` to display environment variables starting with 'M'.

   ```
   C:\Users\mbergljung>set M
   ```

```
M2_HOME=C:\Tools\apache-maven-3.3.1
MAVEN_OPTS=-Xms256m -Xmx1G -XX:PermSize=1024m -noverify
```

Your `MAVEN_OPTS` and `M2_HOME` environment variables are now set. Feel free to increase the specified memory settings if required, for example, if you get "out of memory" errors when running your projects.

Using Alfresco One (Enterprise) (Optional)

By default the Alfresco SDK will use Alfresco Community Edition artifacts but it can be configured to use Alfresco One (Enterprise) artifacts. This requires access credentials for the Alfresco Private Repository, and modification of several Maven configuration files.

To obtain access to the Alfresco One repository located here, refer to this knowledge base article. If you do not have access to this portal then contact your Alfresco technical representative within your company, or Alfresco directly.

Accessing the Alfresco Private Repository

The first matter to consider is to ensure that you have credentials for the Alfresco Private Repository, where the Alfresco One artifacts are stored. In fact the private repository also includes all public artifacts too. Once you have suitable credentials you need to add support for Alfresco private repository to your configuration. This would typically be done by adding your access credentials to the `settings.xml` contained in your `~/.m2` directory (for Linux and OS X). On Windows 7 and Vista this resolves to `<root>\Users\<username>` and on XP it is `<root>\Documents and Settings\<username>\.m2`.

This procedure is explained in detail in the tutorial Configuring access to the Alfresco Private Repository.

Configuring access to Alfresco Private Repository

In order to be able to utilize Enterprise artifacts, it is necessary to allow Maven access to the Alfresco Private Artifacts Repository, where the Enterprise artifacts are maintained.

You need to have permission to access the Alfresco private repository. Enterprise customers can obtain access credentials from Alfresco.

In order to allow Maven access to the Alfresco Private Repository, you must add your credentials to the Maven configuration. This is usually done by adding an entry to the `settings.xml` file, located in your `.m2` directory.

1. Obtain access credentials for the Alfresco Private Repository from Alfresco. This is only available for Enterprise-level customers.
2. Change into your Maven configuration directory. For Linux and Mac OS X that will most likely be `~/.m2` for a configuration on a per-user basis, or for global configuration in `<maven_install>/conf/`. On Windows this would be located in `%USER_HOME%/.m2/` for a per-user configuration, and `%M2_HOME%/conf` for a global configuration.
3. Load `settings.xml` into your editor. Add the following new server configuration in the `<servers>` section:

   ```
   <server>
      <id>alfresco-private-repository</id>
      <username>username</username>
      <password>password</password>
    </server>
   ```

   You will need to replace the placeholder text with your real username and password as allocated by Alfresco. The `id` value should not be changed as it is used in the Alfresco SDK project build files to specify the Enterprise artifacts Maven repository.

It is possible to use encrypted passwords here. See the official Maven documentation for details on how to do this.

At this point you have configured Maven to have access to the Alfresco Private Repository.

Verify install

Before proceeding to use the Alfresco SDK, you should do one final check of your system to ensure you have the prerequisites correctly installed.

Check you have the JDK and Maven correctly installed, and the correct versions of both, and that Maven is configured to use the correct version of the JDK.

Check your configuration by running the command `mvn --version` and listing Maven environment. This will display information similar to the following:

```
$ mvn --version
Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0dfe41d4a06;
 2015-04-22T12:57:37+01:00)
Maven home: /home/martin/apps/apache-maven-3.3.3
Java version: 1.8.0_45, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-8-oracle/jre
Default locale: en_GB, platform encoding: UTF-8
OS name: "linux", version: "3.13.0-58-generic", arch: "amd64", family:
 "unix"

$ env|egrep "M2|MAV"
MAVEN_OPTS=-Xms256m -Xmx1G -noverify
M2_HOME=/home/martin/apps/apache-maven-3.3.3
```

Make sure that the correct version of Maven is installed (3.2.5+) and that the correct version of the JDK is installed (1.8+). If you have multiple JDKs installed double check that Maven is using the correct version of the JDK. If you do not see this kind of output, and your operating system cannot find the `mvn` command, make sure that your `PATH` environment variable and `M2_HOME` environment variable have been properly set.

You are now ready to start using the Alfresco SDK.

Creating a project

The following tutorials take you step by step through how to create and run projects using the standard archetypes provided by the Alfresco SDK.

The tutorials show creation of projects using:

1. Alfresco Repository AMP archetype - is this artifact for your project, read more here.
2. Alfresco Share AMP archetype - is this artifact for your project, read more here.
3. All-in-One archetype - is this artifact for your project, read more here.

Creating a Repository extension project (AMP)

The Alfresco Repository AMP maven archetype can be used to create a new Alfresco Module extension project for the `alfresco.war`.For more information about this project type see Repository AMP Archetype

This task assumes that you have completed all instructions in Installing and configuring software.

This task shows how you can use the Repo AMP archetype of the Alfresco SDK to generate a repository extension module containing a simple example web script.

1. Create a suitable directory in which to store all your Maven projects (if you have not already done so), such as `alfresco-extensions`.
2. Change into your `alfresco-extensions` directory.

3. Run the following command:

   ```
   mvn archetype:generate -Dfilter=org.alfresco:
   ```

   As the archetypes are available via Maven Central you do not need to specify a catalog.

   You will be prompted to choose an archetype:

   ```
   Choose archetype:
   1: remote -> org.alfresco.maven.archetype:alfresco-allinone-archetype
    (Sample multi-module project for All-in-One development on the
    Alfresco plaftorm. Includes modules for: Repository WAR overlay,
    Repository AMP, Share WAR overlay, Solr configuration, and embedded
    Tomcat runner)
   2: remote -> org.alfresco.maven.archetype:alfresco-amp-archetype
    (Sample project with full support for lifecycle and rapid development
    of Repository AMPs (Alfresco Module Packages))
   3: remote -> org.alfresco.maven.archetype:share-amp-archetype (Share
    project with full support for lifecycle and rapid development of AMPs
    (Alfresco Module Packages))
   Choose a number or apply filter (format: [groupId:]artifactId, case
    sensitive contains): :
   ```

4. Enter `2` to have Maven generate an Alfresco Repository Module Package (AMP) project.
5. You will be prompted to choose an archetype version:

   ```
   Choose org.alfresco.maven.archetype:alfresco-amp-archetype version:
   1: 2.0.0-beta-1
   2: 2.0.0-beta-2
   3: 2.0.0-beta-3
   4: 2.0.0-beta-4
   5: 2.0.0
   6: 2.1.0
   7: 2.1.1
   8: 2.2.0
   Choose a number: 8:
   ```

   Press Enter to select the default (the most recent version).
6. You will then be prompted to enter a value for the property `groupId`:

   ```
   Define value for property 'groupId': : com.acme
   ```

   Here we have specified `com.acme` representing the domain for a fictional company `acme.com`. Specify a `groupId` matching your company domain.
7. You will then be prompted to enter a value for the property `artifactId`:

   ```
   Define value for property 'artifactId': : componentX-repo
   ```

   Here we have specified `componentX-repo` representing an X component with a specific extension for the Alfresco Repository. Try and name the Repository extensions in a way so it is easy to see what kind of extension it is for the `alfresco.war` application. Here are some example names for repo extensions so you get the idea: `zip-and-download-action-repo`, `digital-signature-repo`, `business-reporting-repo`, these repository extensions would typically have corresponding Share extensions if they also include user interface functionality. It is good practice to use the following naming convention for repository extensions: `{name}-repo`, where `-repo` indicates that this is an Alfresco Repository extension. Note, hyphens are typically used in artifact IDs.

8. You will then be prompted to enter a value for the property `package`:

   ```
   Define value for property 'package':  com.acme: : com.acme.componentX
   ```

   Here we have specified `com.acme.componentX` representing an X component Java package. This package will be used for any example Java code generated by the archetype. It is good practice to keep all Java code you write under this package so it does not clash with other components/extensions. Any Spring beans generated by this archetype will use this package in the ID.

   Java packages cannot have hyphens in them.

9. You will then be prompted to enter `Y` to accept the values you have entered, or `n` to reject and change. Press Enter to accept the values.

   A new project directory containing a number of sub-directories and support files for the AMP will be created in the directory `componentX-repo`.

10. Change into the freshly created `componentX-repo` directory and browse the various files and directories to see what has been created.

    The following directory structure has been created for you:

    ```
    componentX-repo/
    ### pom.xml (Maven project file)
    ### run.sh  (Mac/Linux script to have this AMP applied to the Alfresco
     WAR and run in Tomcat)
    ### run.bat (Windows script to have this AMP applied to the Alfresco
     WAR and run in Tomcat)
    ### src
    #   ### main
    #   #   ### amp (For more information about the AMP structure see:
     http://docs.alfresco.com/community/concepts/dev-extensions-modules-
    intro.html)
    #   #   #   ### config
    #   #   #   #   ### alfresco
    #   #   #   #       ### extension
    #   #   #   #       #   ### templates
    #   #   #   #       #       ### webscripts (Your Web Scripts should go
     under this directory)
    #   #   #   #       #           ### webscript.get.desc.xml   (Sample
     Web Script that you can try out)
    #   #   #   #       #           ### webscript.get.html.ftl
    #   #   #   #       #           ### webscript.get.js
    #   #   #   #       ### module
    #   #   #   #           ### componentX-repo (AMP Module ID)
    #   #   #   #               ### alfresco-global.properties (Put default
     values for properties specific to this extension here)
    #   #   #   #               ### context
    #   #   #   #               #   ### bootstrap-context.xml
     (Bootstrapping of content models, content, i18n files etc)
    #   #   #   #               #   ### service-context.xml (Your service
     beans go here)
    #   #   #   #               #   ### webscript-context.xml (Your Web
     Script Java controller beans)
    #   #   #   #               ### log4j.properties
    #   #   #   #               ### model
    #   #   #   #               #   ### content-model.xml (Content model
     for your files)
    #   #   #   #               #   ### workflow-model.xml (Content model
     for workflow implementations)
    ```

```
#   #   #   #               ### module-context.xml (Spring context file
 that is picked up by Alfresco)
#   #   #   ### module.properties (AMP module ID, Version etc)
#   #   #   ### web (If your AMP has some UI the files would go here,
 unlikely now when the Alfresco Explorer UI is gone)
#   #   #       ### css
#   #   #       #   ### demoamp.css
#   #   #       ### jsp
#   #   #       #   ### demoamp.jsp
#   #   #       ### licenses
#   #   #       #   ### README-licenses.txt
#   #   #       ### scripts
#   #   #           ### demoamp.js
#   #   ### java (Your Java classes go here, this is where most of the
 module extension implementation code would go, you can remove the demo
 component)
#   #       ### com
#   #           ### acme
#   #               ### componentX
#   #                   ### demoamp  (Demo module component and demo
 web script controller, can be removed)
#   #                       ### DemoComponent.java
#   #                       ### Demo.java
#   #                       ### HelloWorldWebScript.java
#   ### test
#       ### java
#       #   ### com
#       #       ### acme
#       #           ### componentX
#       #               ### demoamp
#       #                   ### test  (Example test of the demo
 component, can be removed)
#       #                       ### DemoComponentTest.java
#       ### properties
#       #   ### local
#       #       ### alfresco-global.properties (environment specific
 configuration, the local env is active by default)
#       ### resources
#           ### alfresco
#           #   ### extension
#           #       ### disable-webscript-caching-context.xml (file to
 disable server side JavaScript compilation to Java code)
#           ### log4j.properties
### tomcat
    ### context.xml  (Virtual Webapp context for RAD development)
```

11. At this point, before you have made any changes, you can build the project by typing:

    ```
    mvn clean install
    ```

    Maven will ensure that all requirements are downloaded. This may take some time.

    The project will return with the message `BUILD SUCCESS`. You should see the AMP artifact installed in your local repository `.m2/repository/com/acme/componentX-repo/1.0-SNAPSHOT/componentX-repo-1.0-SNAPSHOT.amp`

12. Run and Test the sample Web Script

    To test the Web Script you will need to start an embedded Tomcat and deploy the Alfresco WAR with the componentX-repo AMP applied. This can be done in two ways:

    1. With `mvn clean install -Pamp-to-war`
    2. Via the `run.sh` script (or run.bat on Windows), which does the same thing, plus making sure Spring Loaded library is available.

This will only run the customized Alfresco Repository application (alfresco.war), Alfresco Share UI (share.war) and Search (solr4.war) is not available. If you need those too then use the All-in-One project instead.

Let's start Tomcat via the script as follows (use run.bat on Windows):

```
./run.sh
...
INFO: WSSERVLET12: JAX-WS context listener initializing
Apr 30, 2015 10:04:39 AM
 com.sun.xml.ws.transport.http.servlet.WSServletDelegate <init>
INFO: WSSERVLET14: JAX-WS servlet initializing
2015-04-30 10:04:39,152  WARN  [shared_impl.util.LocaleUtils]
 [localhost-startStop-1] Locale name in faces-config.xml null or empty,
 setting locale to default locale : en_GB
 Apr 30, 2015 10:04:39 AM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8080"]
2015-04-30 10:05:24,545  INFO
 [management.subsystems.ChildApplicationContextFactory]
 [SearchScheduler_Worker-1] Starting 'Transformers' subsystem, ID:
 [Transformers, default]
 2015-04-30 10:05:24,741  INFO
 [management.subsystems.ChildApplicationContextFactory]
 [SearchScheduler_Worker-1] Startup of 'Transformers' subsystem, ID:
 [Transformers, default] complete
```

You may first need to make the shell script executable using the command `chmod +x run.sh`.

13. Once the web application has started (it may take a little while), point your browser at `http://localhost:8080/alfresco/service/sample/helloworld` to test the web script. The web script should return the following text: *Message: Hello World from JS! HelloFromJava*
14. If you are prompted to login, then use username `admin` and a password `admin`.
15. Configure Enterprise Edition **(Optional)**

    So far we have used the project with its default configuration, which is using Alfresco Community Edition 5.1.e. If you are going to work with the Enterprise edition then you need to do the following:

    1. Decide what Enterprise version you will be using, if you are going to use version 5.1.0 then you are set as that is the default Enterprise version. If you want to use another version then open up the `componentX-repo/pom.xml` file and uncomment the `<alfresco.version>5.1.e</alfresco.version>` property and set the new version number you want to use.
    2. Update the `run.sh` and `run.bat` scripts to use the Enterprise edition, this is done by using the `enterprise` profile in the maven command line: `mvn clean install -Pamp-to-war,enterprise`. Note. this automatically sets the version to 5.1.0 if you have not explicitly set it.
    3. And as you might have guessed, if you are not using the run script, you will have to remember to always activate the Enterprise profile when running: `mvn clean install -Pamp-to-war,enterprise`
    4. Install an enterprise license, otherwise the server will be in read-only mode, it goes into the following directory:

    ```
    componentX-repo/
    ### src
    ```

```
#   ### main
#   #   ### amp
#   #   #   ### config
#   #   #   #   ### alfresco
#   #   #   #       ### extension
#   #   #   #       #   ### license
#   #   #   #       #       ### acme-5.1.0-enterprise.lic
```

16. Stopping the Embedded Tomcat Server

    To stop the Alfresco Tomcat server that was started with the `mvn clean install -Pamp-to-war` command (either directly or indirectly via the run script), do **Ctrl+C** in the terminal that was used.

    The Maven embedded tomcat process will terminate after a short wait.

In this tutorial you have learned how to generate, deploy, and run a project using the Repository AMP archetype.

Creating a Share extension project (AMP)

The Alfresco Share AMP maven archetype can be used to create a new Alfresco Module extension project for the `share.war`.For more information about this project type see Share AMP Archetype

This task assumes you have completed all instructions in Installing and configuring software.

This task shows how you can use the Share AMP archetype of the Alfresco SDK to generate a Share webapp extension module containing a simple Aikau Page and Widget (Aikau is the new Alfresco UI Framework used to build web pages and Dashlets).

1. Create a suitable directory in which to store all your Maven projects (if you have not already done so), such as `alfresco-extensions`.
2. Change into your `alfresco-extensions` directory.
3. Run the following command:

   ```
   mvn archetype:generate -Dfilter=org.alfresco:
   ```

   As the archetypes are available via Maven Central you do not need to specify a catalog.

   You will be prompted to choose an archetype:

   ```
   Choose archetype:
   1: remote -> org.alfresco.maven.archetype:alfresco-allinone-archetype
    (Sample multi-module project for All-in-One development on the
    Alfresco plaftorm. Includes modules for: Repository WAR overlay,
    Repository AMP, Share WAR overlay, Solr configuration, and embedded
    Tomcat runner)
   2: remote -> org.alfresco.maven.archetype:alfresco-amp-archetype
    (Sample project with full support for lifecycle and rapid development
    of Repository AMPs (Alfresco Module Packages))
   3: remote -> org.alfresco.maven.archetype:share-amp-archetype (Share
    project with full support for lifecycle and rapid development of AMPs
    (Alfresco Module Packages))
   Choose a number or apply filter (format: [groupId:]artifactId, case
    sensitive contains): :
   ```

4. Enter `3` to have Maven generate an Alfresco Share Module Package (AMP) project.
5. You will be prompted to choose an archetype version:

   ```
   Choose org.alfresco.maven.archetype:alfresco-amp-archetype version:
   ```

```
1: 2.0.0-beta-1
2: 2.0.0-beta-2
3: 2.0.0-beta-3
4: 2.0.0-beta-4
5: 2.0.0
6: 2.1.0
7: 2.1.1
8: 2.2.0
Choose a number: 8:
```

Press Enter to select the default (which is the most recent version).

6. You will then be prompted to enter a value for the property `groupId`:

   ```
   Define value for property 'groupId': : com.acme
   ```

   Here we have specified `com.acme` representing the domain for a fictional company `acme.com`. Specify a `groupId` matching your company domain.

7. You will then be prompted to enter a value for the property `artifactId`:

   ```
   Define value for property 'artifactId': : componentX-share
   ```

   Here we have specified `componentX-share` representing an X component with a specific extension for the Alfersco Share UI. Try and name the Share extensions in a way so it is easy to see what kind of extension it is for the `share.war` application. Here are some example names for share extensions so you get the idea: zip-and-download-action-share, digital-signature-share, business-reporting-share, these share extensions would typically have corresponding Repo extensions if they also include server side business logic as part of the implementation. It is good practice to use the following naming convention for share extensions: `{name}-share`, where -share indicates that this is an Alfresco Share extension. Note, hyphens are typically used in artifact IDs.

8. You will then be prompted to enter a value for the property `package`:

   ```
   Define value for property 'package':  com.acme: : com.acme.componentX
   ```

   Here we have specified `com.acme.componentX` representing an X component Java package. This package will be used for any example Java code generated by the archetype. It is good practice to keep all Java code you write under this package so it does not clash with other components/extensions. Any Spring beans generated by this archetype will use this package in the ID.

   Java packages cannot have hyphens in them.

9. You will then be prompted to enter `Y` to accept the values you have entered, or `n` to reject and change. Press Enter to accept the values.

   A new project directory containing a number of sub-directories and support files for the AMP will be created in the directory `componentX-share`.

10. Change into the freshly created `componentX-share` directory and browse the various files and directories to see what has been created.

    The following directory structure has been created for you:

    ```
    componentX-share/
    ### pom.xml (Maven project file)
    ### run.sh  (Mac/Linux script to have this AMP applied to the Share WAR
     and run in Tomcat)
    ```

```
### run.bat (Windows script to have this AMP applied to the Share WAR
 and run in Tomcat)
### src
#   ### main
#   #   ### amp (For more information about the AMP structure see:
 https://wiki.alfresco.com/wiki/AMP_Files)
#   #   #   ### config
#   #   #   #   ### alfresco
#   #   #   #       ### web-extension
#   #   #   #           ### componentX-share-slingshot-application-
context.xml  (Loads componentX-share.properties)
#   #   #   #           ### messages
#   #   #   #           #   ### componentX-share.properties  (Custom
 share UI labels, messages etc)
#   #   #   #           ### site-data
#   #   #   #           #   ### extensions
#   #   #   #           #       ### componentX-share-example-
widgets.xml  (Dojo package definitions for the Aikau framework, Share
 config)
#   #   #   #           ### site-webscripts
#   #   #   #               ### com
#   #   #   #               #   ### example
#   #   #   #               #           ### pages
#   #   #   #               #           ### simple-page.get.desc.xml
 (Simple Aikau page for demonstration purpose)
#   #   #   #               #           ### simple-page.get.html.ftl
#   #   #   #               #           ### simple-page.get.js
#   #   #   #               ### org
#   #   #   #                   ### alfresco
#   #   #   #                       ### README.md
#   #   #   ### file-mapping.properties
#   #   #   ### module.properties
#   #   #   ### web
#   #   #       ### js
#   #   #           ### example
#   #   #               ### widgets (Simple Aikau widget for
 demonstration purpose)
#   #   #                   ### css
#   #   #                   #   ### TemplateWidget.css
#   #   #                   ### i18n
#   #   #                   #   ### TemplateWidget.properties
#   #   #                   ### templates
#   #   #                   #   ### TemplateWidget.html
#   #   #                   ### TemplateWidget.js
#   #   ### java
#   #   #   ### com
#   #   #       ### acme
#   #   #           ### componentX
#   #   ### resources
#   #       ### META-INF
#   #           ### resources
#   #           #   ### test.html
#   #           ### share-config-custom.xml.sample (Remove .sample to
 use and keep extension specific stuff in this config)
#   ### test
#       ### java
#       #   ### com
#       #       ### acme
#       #           ### componentX
#       ### properties
#       ### resources
#           ### alfresco
#           #   ### web-extension
#           #       ### share-config-custom.xml (Configures where the
 Repository is running)
#           ### log4j.properties
### tomcat
```

```
    ### context.xml (Virtual Webapp context for RAD development)
```

11. At this point, before you have made any changes, you can build the project by typing:

    ```
    mvn clean install
    ```

    Maven will ensure that all requirements are downloaded. This make take some time.

    The project will return with the message `BUILD SUCCESS`. You should see the AMP artifact installed in your local repository `.m2/repository/com/acme/componentX-share/1.0-SNAPSHOT/componentX-share-1.0-SNAPSHOT.amp`

12. Run and Test the sample Aikau Page

    For this to work you will need to have the Alfresco WAR running in another Tomcat (8080). You can quite easily achieve this by generating a repo-amp project and running it, see Repository AMP Project.

    To test the custom Share page you will need to start an embedded Tomcat (8081) and deploy the Share WAR with the componentX-share AMP applied. This can be done in two ways:

    1. With `mvn clean install -Pamp-to-war`
    2. Via the `run.sh` script (or run.bat on Windows), which does the same thing, plus making sure Spring Loaded library is available.

    Let's start Tomcat via the script as follows (use run.bat on Windows):

    ```
    ./run.sh
    ...
    Apr 30, 2015 11:40:42 AM org.apache.catalina.core.ApplicationContext
     log
    INFO: org.tuckey.web.filters.urlrewrite.UrlRewriteFilter INFO: loaded
     (conf ok)
    Apr 30, 2015 11:40:42 AM org.apache.catalina.core.ApplicationContext
     log
    INFO: Initializing Spring FrameworkServlet 'Spring Surf Dispatcher
     Servlet'
    Apr 30, 2015 11:40:42 AM org.apache.coyote.AbstractProtocol start
    INFO: Starting ProtocolHandler ["http-bio-8081"]
    ```

    You may first need to make the shell script executable using the command `chmod +x run.sh`.

13. Once the web application has started (it may take a little while), point your browser at `http://localhost:8081/share/page/hdp/ws/simple-page` to test the sample Page. The page should return the following Title: *This is a simple page* and a Hello World widget with the text: *Hello from i18n!*
14. If you are prompted to login, then use username `admin` and a password `admin`.
15. Configure Enterprise Edition **(Optional)**

    So far we have used the project with its default configuration, which is using Alfresco Community Edition 5.1.e. If you are going to work with the Enterprise edition then you need to do the following:

    - Decide what Enterprise version you will be using, if you are going to use version 5.1.0 then you are set as that is the default Enterprise version. If you want to use another version then open up the `componentX-share/pom.xml` file and uncomment the `<alfresco.version>5.1.e</alfresco.version>` property and set the new version number you want to use.

- Update the `run.sh` and `run.bat` scripts to use the Enterprise edition, this is done by using the `enterprise` profile in the maven command line: `mvn clean install -Pamp-to-war,enterprise`. Note. this automatically sets the version to 5.1.0 if you have not explicitly set it.
- And as you might have guessed, if you are not using the run script, you will have to remember to always activate the Enterprise profile when running: `mvn clean install -Pamp-to-war,enterprise`

16. Stopping the Embedded Tomcat Server

    To stop the Alfresco Tomcat server that was started with the `mvn clean install -Pamp-to-war` command (either directly or indirectly via the run script), do **Ctrl+C** in the terminal that was used.

    The Maven embedded tomcat process will terminate after a short wait.

In this tutorial you have learned how to generate, deploy, and run a project using the Share AMP archetype.

Creating an All-in-One (AIO) project (WARs)

The All-in-One maven archetype can be used to create a new multi-module project that will produce customized `alfresco.war` and `share.war` artifacts. For more information about this project type see All-in-One Archetype

This task assumes that you have completed all instructions in Installing and configuring.

This task shows how you can use the All-in-One archetype of the Alfresco SDK to generate a multi module extension project containing repo-amp, share-amp, and WAR projects.

1. Create a suitable directory in which to store all your Maven projects (if you have not already done so), such as `alfresco-extensions`.
2. Change into your `alfresco-extensions` directory.
3. Run the following command:

   ```
   mvn archetype:generate -Dfilter=org.alfresco:
   ```

   As the archetypes are available via Maven Central you do not need to specify a catalog.

   You will be prompted to choose an archetype:

   ```
   Choose archetype:
   1: remote -> org.alfresco.maven.archetype:alfresco-allinone-archetype
    (Sample multi-module project for All-in-One development on the
    Alfresco plaftorm. Includes modules for: Repository WAR overlay,
    Repository AMP, Share WAR overlay, Solr configuration, and embedded
    Tomcat runner)
   2: remote -> org.alfresco.maven.archetype:alfresco-amp-archetype
    (Sample project with full support for lifecycle and rapid development
    of Repository AMPs (Alfresco Module Packages))
   3: remote -> org.alfresco.maven.archetype:share-amp-archetype (Share
    project with full support for lifecycle and rapid development of AMPs
    (Alfresco Module Packages))
   Choose a number or apply filter (format: [groupId:]artifactId, case
    sensitive contains): :
   ```

4. Enter `1` to have Maven generate an Alfresco All-in-One project.
5. You will be prompted to choose an archetype version:

```
Choose org.alfresco.maven.archetype:alfresco-amp-archetype version:
1: 2.0.0-beta-1
2: 2.0.0-beta-2
3: 2.0.0-beta-3
4: 2.0.0-beta-4
5: 2.0.0
6: 2.1.0
7: 2.1.1
8: 2.2.0
Choose a number: 8:
```

Press Enter to select the default (which is the most recent version).

6. You will then be prompted to enter a value for the property `groupId`:

   ```
   Define value for property 'groupId': : com.acme
   ```

   Here we have specified `com.acme` representing the domain for a fictional company `acme.com`. Specify a `groupId` matching your company domain.

7. You will then be prompted to enter a value for the property `artifactId`:

   ```
   Define value for property 'artifactId': : acme-cms-poc
   ```

   Here we have specified `acme-cms-poc` representing a Proof-of-Concept (PoC) Content Management System (CMS) project to validate Alfresco as a perfect fit. Try and name the All-in-One projects so it is easy to know what they contain/represent. The naming should represent complete projects in contrast to specific AMP extensions that just implements a specific functionality in a larger solution. Note, hyphens are typically used in artifact IDs.

8. You will then be prompted to enter a value for the property `package`:

   ```
   Define value for property 'package':  com.acme: : com.acme.cmspoc
   ```

   Here we have specified `com.acme.cmspoc` representing the top Java package for this project. This package will be used for any example Java code generated by the archetype. It is good practice to keep all Java code you write under this package so it does not clash with other projects. Any Spring beans generated by this archetype will use this package in the ID.

   Java packages cannot have hyphens in them.

9. You will then be prompted to enter `Y` to accept the values you have entered, or `n` to reject and change. Press Enter to accept the values.

   A new project directory containing a number of sub-directories and support files for the project will be created in the directory `acme-cms-poc`.

10. Change into the freshly created `acme-cms-poc` directory and browse the various files and directories to see what has been created.

    The following directory structure has been created for you:

    ```
    acme-cms-poc/
    ### pom.xml (Maven project file)
    ### run.sh  (Mac/Linux script to run customized Alfresc WAR and Share
     WAR together with Solr 4 in Tomcat)
    ### run.bat (Windows script to run customized Alfresc WAR and Share WAR
     together with Solr 4 in Tomcat)
    ### acme-cms-poc-repo-amp  (This is a standard Repository AMP extions
     project - produces a Repository AMP)
    ```

```
#   ### pom.xml (Maven project file for AMP, put dependencies here that
 are only needed by this AMP)
#   ### src
#       ### main
#       #   ### amp  (For more information about the AMP structure see:
 https://wiki.alfresco.com/wiki/AMP_Files)
#       #   #   ### config
#       #   #   #   ### alfresco
#       #   #   #       ### extension
#       #   #   #       #   ### templates
#       #   #   #       #       ### webscripts  (Your Web Scripts
 should go under this directory)
#       #   #   #       #           ### webscript.get.desc.xml  (Sample
 Web Script that you can try out)
#       #   #   #       #           ### webscript.get.html.ftl
#       #   #   #       #           ### webscript.get.js
#       #   #   #       ### module
#       #   #   #           ### acme-cms-poc-repo-amp  (AMP Module ID)
#       #   #   #               ### alfresco-global.properties (Put
 default values for properties specific to this extension here)
#       #   #   #               ### context
#       #   #   #               #   ### bootstrap-context.xml
 (Bootstrapping of content models, content, i18n files etc)
#       #   #   #               #   ### service-context.xml  (Your
 service beans go here)
#       #   #   #               #   ### webscript-context.xml (Web
 Script Java controller beans)
#       #   #   #               ### model
#       #   #   #               #   ### content-model.xml   (Content
 model for your files)
#       #   #   #               #   ### workflow-model.xml  (Content
 model for workflow implementations)
#       #   #   #               ### module-context.xml  (Spring context
 file that is picked up by Alfresco)
#       #   #   ### module.properties  (AMP module ID, Version etc)
#       #   #   ### web  (If your AMP has some UI the files would go
 here, unlikely now when the Alfresco Explorer UI is gone)
#       #   #       ### css
#       #   #       #   ### demoamp.css
#       #   #       ### jsp
#       #   #       #   ### demoamp.jsp
#       #   #       ### licenses
#       #   #       #   ### README-licenses.txt
#       #   #       ### scripts
#       #   #           ### demoamp.js
#       #   ### java  (Your Java classes go here, this is where most of
 the module extension implementation code would go, you can remove the
 demo component)
#       #       ### com
#       #           ### acme
#       #               ### cmspoc
#       #                   ### demoamp  (Demo module component and Web
 Script Java controller, can be removed)
#       #                       ### DemoComponent.java
#       #                       ### Demo.java
#       #                       ### HelloWorldWebScript.java
#       ### test
#           ### java
#           #   ### com
#           #       ### acme
#           #           ### cmspoc
#           #               ### demoamp
#           #                   ### test  (Example test of the demo
 component, can be removed)
#           #                       ### DemoComponentTest.java
#           ### properties
#           #   ### local
#           #       ### alfresco-global.properties  (environment
 specific configuration, the local env is active by default)
#           ### resources
#               ### alfresco
#               #   ### extension
#               #       ### disable-webscript-caching-context.xml
 (file to disable server side JavaScript compilation to Java code)
#               ### test-log4j.properties
### acme-cms-poc-share-amp  (This is a standard Share AMP extions
 project - produces a Share AMP)
#   ### pom.xml (Maven project file for AMP, put dependencies here that
 are only needed by this AMP)
#   ### src
#       ### main
#       #   ### amp  (For more information about the AMP structure see:
 https://wiki.alfresco.com/wiki/AMP_Files)
#       #   #   ### config
#       #   #   #   ### alfresco
#       #   #   #       ### web-extension
#       #   #   #           ### acme-cms-poc-share-amp-slingshot-
application-context.xml  (Loads the acme-cms-poc-share-amp.properties
 file)
#       #   #   #           ### messages
#       #   #   #           #   ### acme-cms-poc-share-amp.properties
 (Custom share UI labels, messages etc)
#       #   #   #           ### site-data
#       #   #   #           #   ### extensions
#       #   #   #           #       ### acme-cms-poc-share-amp-example-
widgets.xml  (Dojo package definitions for the Aikau framework, Share
 config)
#       #   #   #           ### site-webscripts
#       #   #   #               ### com
#       #   #   #               #   ### example
#       #   #   #               #       ### pages
#       #   #   #               #           ### simple-
page.get.desc.xml  (Simple Aikau page for demonstration purpose)
#       #   #   #               #           ### simple-
page.get.html.ftl
#       #   #   #               #           ### simple-page.get.js
#       #   #   #               ### org
#       #   #   #                   ### alfresco
#       #   #   #                       ### README.md
#       #   #   ### file-mapping.properties
#       #   #   ### module.properties
#       #   #   ### web
#       #   #       ### js
#       #   #           ### example
#       #   #               ### widgets   (Simple Aikau widget for
 demonstration purpose)
#       #   #                   ### css
#       #   #                   #   ### TemplateWidget.css
#       #   #                   ### i18n
#       #   #                   #   ### TemplateWidget.properties
#       #   #                   ### templates
#       #   #                   #   ### TemplateWidget.html
#       #   #                   ### TemplateWidget.js
#       #   ### java
#       #   #   ### com
#       #   #       ### acme
#       #   #           ### cmspoc
#       #   ### resources
#       #       ### META-INF
#       #           ### share-config-custom.xml.sample
#       ### test
#           ### java
#           #   ### com
#           #       ### acme
#           #           ### cmspoc
#           #               ### demoamp  (Example of how to use
 Alfresco Share Page Objects (PO) to create functional tests for your
 UI customizations)
#           #                   ### DemoPageTestIT.java
#           #                   ### po
#           #                       ### DemoPage.java
#           ### resources
#               ### testng.xml
### repo  (This is the Alfresco WAR project - produces a customized
 Alfresco.WAR by applying the AMP produced by the /acme-cms-poc-repo-
amp project)
#   ### pom.xml (Maven project file for Repository WAR (alfresco.war),
 add AMP and JAR dependencies and overlay config here)
#   ### src
#       ### main
#           ### properties
#           #   ### local
#           #       ### alfresco-global.properties
#           ### resources
#               ### alfresco
#                   ### extension
#                       ### dev-log4j.properties
### runner (Tomcat Runner that deploys the WARs produced by the /repo
 and /share projects, the Solr 4 webapp is deployed directly from maven
 repo)
#   ### pom.xml
#   ### src
#   #   ### main
#   #       ### webapp
#   #           ### index.html
#   ### tomcat   (Virtual Webapp contexts for RAD development)
#       ### context-repo.xml
#       ### context-share.xml
#       ### context-solr.xml
### share (This is the Share WAR project - produces a customized
 Share.WAR by applying the AMP produced by the /acme-cms-poc-share-amp
 project)
#   ### pom.xml (Maven project file for Share WAR (share.war), add AMP
 and JAR dependencies and overlay config here)
#   ### src
#       ### main
#       #   ### resources
#       #       ### alfresco
#       #       #   ### web-extension
#       #       #       ### custom-slingshot-application-
context.xml.sample
#       #       #       ### share-config-custom.xml.sample
#       #       ### log4j.properties
#       ### test
#           ### resources
#               ### alfresco
#               #   ### web-extension
#               #       ### share-config-custom.xml
#               ### log4j.properties
### solr-config  (Loads the configuration files necessary for running
 Apache Solr 4)
   ### pom.xml
```

11. At this point, before you have made any changes, you can build the project by typing:

```
mvn clean install
```

Maven will ensure that all requirements are downloaded. This make take some time.

As the build continues you will see the following artifacts built and installed in your local repository:

- `.m2/repository/com/acme/acme-cms-poc-repo-amp/1.0-SNAPSHOT/acme-cms-poc-repo-amp-1.0-SNAPSHOT.amp`
- `.m2/repository/com/acme/acme-cms-poc-share-amp/1.0-SNAPSHOT/acme-cms-poc-share-amp-1.0-SNAPSHOT.amp`
- `.m2/repository/com/acme/repo/1.0-SNAPSHOT/repo-1.0-SNAPSHOT.war`, contains the acme-cms-poc-repo-amp-1.0-SNAPSHOT.amp
- `.m2/repository/com/acme/share/1.0-SNAPSHOT/share-1.0-SNAPSHOT.war`, contains the acme-cms-poc-share-amp-1.0-SNAPSHOT.amp

The project will return with the message `BUILD SUCCESS`.

12. You can build, load RAD requirements, and run your project by typing:

```
./run.sh
```

You may first need to make the shell script executable using the command `chmod +x run.sh`.

13. Direct your web browser to:

```
http://localhost:8080/share
```

You can log in using a user name of `admin` and a password of `admin`.

14. Using Alfresco Community version > 5.1.e **(Optional)**

It is likely that the latest Alfresco Community version is newer than what is default in the SDK (i.e. 5.1.e). See this article for what do when setting newer version than 5.1.e. It requires a few more steps than just setting the version number.

15. Configure Enterprise Edition **(Optional)**

So far we have used the project with its default configuration, which is using Alfresco Community Edition 5.1.e. If you are going to work with the Enterprise edition, then you need to do the following:

1. Decide what Enterprise version you will be using, if you are going to use version 5.1.0 then you are set as that is the default Enterprise version. If you want to use another version then open up the `acme-cms-poc/pom.xml` file and uncomment the `<alfresco.version>5.1.e</alfresco.version>` property and set the new version number you want to use. See this article for what do when setting newer version than 5.1.0. It requires a few more steps than just setting the version number.
2. Update the `run.sh` and `run.bat` scripts to use the Enterprise edition, this is done by using the `enterprise` profile in the maven command line: `mvn clean install -Prun,enterprise`. Note. this automatically sets the version to 5.1.0 if you have not explicitly set it.
3. And as you might have guessed, if you are not using the run script, you will have to remember to always activate the Enterprise profile when running: `mvn clean install -Prun,enterprise`
4. Install an enterprise license, otherwise the server will be in read-only mode. It goes into the following directory:

```
acme-cms-poc/
```

```
### repo
#   ### pom.xml
#   ### src
#       ### main
#           ### properties
#           ### resources
#               ### alfresco
#                   ### extension
#   #   #   #       #   ### license
#   #   #   #       #       ### acme-5.1.0-enterprise.lic
```

If the license is properly installed you should see logs as follows when the server starts:

```
...
2015-05-08 09:52:21,359  INFO
 [enterprise.license.AlfrescoLicenseManager] [localhost-startStop-1]
 Successfully installed license from file [/home/martin/src/alfresco-
extensions/acme-cms-poc/runner/target/tomcat/webapps/repo/WEB-INF/
classes/alfresco/extension/license/Enterprise-5.0.lic]
...
2015-05-08 09:52:23,614  INFO  [service.descriptor.DescriptorService]
 [localhost-startStop-1] Alfresco started (Enterprise). Current
 version: 5.0.1 (r100823-b68) schema 8,022. Originally installed
 version: 5.0.0 (d r99759-b2) schema 8,022.
...
```

16. Stopping the Embedded Tomcat Server

    To stop the Alfresco Tomcat server that was started with the `mvn clean install -Prun` command (either directly or indirectly via the run script), do **Ctrl+C** in the terminal that was used.

    The Maven embedded tomcat process will terminate after a short wait.

In this tutorial you have learned how to generate, deploy, and run a project using the All-in-One (AIO) archetype.

Maven Archetypes - Command Reference

There are three Maven archetypes on which the Alfresco SDK can base the generation of projects. This information provides the Maven command reference for these projects.

For more information about the archetypes see:

- Repository AMP archetype
- All-In-One (AIO) archetype
- Share AMP archetype

Repository AMP archetype command reference

This describes the scripts and Maven commands that can be used on an Alfresco Repository extension project based on the Repository AMP archetype.

Scripts and commands:

Command	Description
`./run.sh and run.bat`	**Linux/Mac and Windows** scripts for running an embedded Tomcat with the customized `alfresco.war` (that is, with the Repo AMP applied) and the flat file database H2. Access to Alfresco UI is via `http://localhost:8080/alfresco`. The username/password is admin/admin. This script will also configure JVM memory (it basically sets up `MAVEN_OPTS` for you). See inside the script for further details. **Note. Spring loaded is no longer used.** This script assumes that you are developing for Alfresco Community Edition. If you use Alfresco One you need to update the maven command in this script so it uses the `enterprise` profile: `mvn integration-test -Pamp-to-war,enterprise`.
`mvn compile alfresco:refresh-repo`	Compiles the source code and puts the class files and resources under `/target`. Then makes a POST call to the Alfresco Repository web application (`alfresco.war`) to refresh the web script container. So any changes that were made to web scripts should be visible after a page refresh. This command is typically used together with the `run.sh/bat script` for Rapid Application Development (RAD). The RAD process can be described like this: 1. Start Tomcat with current `alfresco.war` customization (`run.sh/bat`) in console window one. 2. From an editor, change some files (classes, web scripts, and so on). 3. Execute this cmd (`mvn compile alfresco:refresh-repo`) from console window two. 4. Refresh the page/web script that you are working on. 5. Done? No -> Go back to step 2 and start over. 6. Finished with implementation.
mvn package	Runs unit tests and packages AMP in `${project.build.directory}/${project.build.finalName}.amp`.
mvn install	Like `mvn package` but also installs AMP in local Maven repository to be depended upon.
mvn test	Runs unit tests.
mvn install -DskipTests=true	Like `mvn install` but skips unit tests.

Command	Description
mvn install -Pamp-to-war	Like `run.sh` or `run.bat` but does not configure JVM memory if you have not configured it in `MAVEN_OPTS`. See set up MAVEN_OPTS. If you use Alfresco One see the next command.
mvn install -Pamp-to-war,enterprise	Like `mvn install -Pamp-to-war` but uses Alfresco One (Enterprise) artifacts. Note you need to have set up access to the private repository containing the Alfresco One artifacts.
mvn clean -Ppurge	Removes H2 database (with metadata), alf_data (with content files and index files), and log files. Useful to purge the development repo (by default self contained in `${project.basedir}/alf_data_dev`). This is an important command to use if you change significant settings in your project. For example, if you change Alfresco Community Edition to Alfresco One. It is important to purge databases and other data that might otherwise be persisted. The `purge` profile cannot be used together with the `amp-to-war` profile.

Share AMP archetype command reference

This describes the scripts and Maven commands that can be used on an Alfresco Share extension project based on the Share AMP archetype.

Scripts and commands:

Command	Description
`./run.sh and run.bat`	**Linux/Mac and Windows** scripts for running an embedded Tomcat with the customized `share.war` (that is, with the Share AMP applied). Access to Alfresco Share UI is via `http://localhost:8081/share`. The username/password is admin/admin. This script will also configure JVM memory and automatically set up Spring Loaded for hot reloading of classes (it basically sets up `MAVEN_OPTS` for you). See inside script for further details. This script assumes that you are developing for the Alfresco Community Edition. If you use Alfresco One you need to update the maven command in this script so it uses the `enterprise` profile: `mvn integration-test -Pamp-to-war,enterprise`. This script also assumes that another Tomcat is running locally on port 8080 with the Alfresco Repository (`alfresco.war`) web application deployed.

Command	Description
`mvn compile alfresco:refresh-share`	Compiles the source code and puts the class files and resources under `/target`. Then makes POST calls to the Alfresco Share web application (`share.war`) to refresh the Spring Surf web script container and clear dependency caches. So any changes that was made to web scripts, Aikau pages, Aikau Widgets, Dashlets, and so on, should be visible after a page refresh. This command is typically used together with the run.sh/bat script for Rapid Application Development (RAD). The RAD process can be described like this: 1. Start Tomcat with current `share.war` customization (`run.sh/bat`) in console window one. 2. From an editor change some files (classes, pages, widgets, and so on) 3. Execute this cmd (`mvn compile alfresco:refresh-share`) from console window two. 4. Refresh the page/web script you are working on. 5. Done? No -> Go back to step 2 and start over. 6. Finished with implementation.
mvn package	Runs unit tests and packages AMP in `${project.build.directory}/${project.build.finalName}.amp`.
mvn install	Like `mvn package` but also installs AMP in local Maven repository to be depended upon.
mvn test	Runs unit tests.
mvn install -DskipTests=true	Like mvn install but skips unit tests.
mvn install -Pamp-to-war	Like `run.sh` or `run.bat` but does not configure JVM memory and Spring Loaded if you have not configured it in `MAVEN_OPTS`. See set up MAVEN_OPTS. If you use the Alfresco One see next command.
mvn install -Pamp-to-war,enterprise	Like `mvn install -Pamp-to-war` but uses Alfresco One artifacts. Note you need to have set up access to the private repository containing the Alfresco One artifacts.

All-in-One (AIO) archetype command reference

This describes the scripts and Maven commands that can be used on an Alfresco All-in-One (AIO) extension project based on the AIO archetype.

The All-in-One Alfresco project contains the following modules:

- `repo-amp`: A Repository AMP project, demonstrating sample project structure and demo component loading.
- `repo`: An alfresco.war aggregator project, overlaying the standard Alfresco WAR with the repo-amp and any other AMPs and JARs that have been included as dependencies and configured in the overlay
- `share-amp`: A Share AMP project, demonstrating sample project structure and demo Aikau page
- `share`: A `share.war` aggregator project, overlaying the standard Share WAR with the share-amp and any other AMPs and JARs that have been included as dependencies and configured in the overlay
- `solr-config`: Brings in the Apache Solr 4 configuration files
- `runner`: A Tomcat + H2 runner, capable of running the custom `alfresco.war`, custom `share.war`, and `solr4.war` in embedded mode for demo/integration-testing purposes

Scripts and commands:

Command	Description
`./run.sh and run.bat`	**Linux/Mac and Windows** scripts for running an embedded Tomcat with the customized `alfresco.war` (repo-amp applied), custom share.war (share-amp applied), and solr4.war. Access to Alfresco Share UI is via `http://localhost:8080/share`. Username/pwd is admin/admin. This script will also configure JVM memory (it basically sets up `MAVEN_OPTS` for you). See inside script for further details. **Note. Spring loaded is no longer used.** This script assumes that you are developing for the Alfresco Community Edition. If you use Alfresco One, you need to update the maven command in this script so it uses the 'enterprise' profile: `mvn install -Prun,enterprise`.

Command	Description
`repo-amp/mvn compile alfresco:refresh-repo`	Compiles the source code for the **Repository AMP** and puts the class files and resources under repo-amp/target. Then makes a POST call to the Alfresco Repository web application (alfresco.war) to refresh the web script container. So any changes that was made to Web scripts should be visible after a page refresh. This command is typically used together with the run.sh/bat script for Rapid Application Development (RAD). The RAD process can be described like this: 1. Start Tomcat with current `alfresco.war` customization (that is, `run.sh`/`bat`) in console window one. 2. From an editor change some files (classes, web scripts, and so on) for the Repository AMP. 3. Execute this cmd (that is, `mvn repo-amp/compile alfresco:refresh-repo`) from console window two. 4. Refresh the page / web script you are working on. 5. Done? No -> Go back to step 2 and start over. 6. Finished with implementation.

Command	Description
`share-amp/mvn compile alfresco:refresh-share`	Compiles the source code for the **Share AMP** and puts the class files and resources under `share-amp/target`. Then makes POST calls to the Alfresco Share web application (`share.war`) to refresh the Spring Surf web script container and clear dependency caches. So any changes that was made to web scripts, Aikau pages, Aikau widgets, dashlets, and so on, should be visible after a page refresh. This command is typically used together with the `run.sh/bat` script for Rapid Application Development (RAD). The RAD process can be described like this: 1. Start Tomcat with current `share.war` customization (that is, `run.sh/bat`) in console window one. 2. From an editor change some files (classes, pages, widgets, and so on) for the Share AMP. 3. Execute this cmd (that is, `share-amp/mvn compile alfresco:refresh-share`) from console window two. 4. Refresh the page / web script you are working on. 5. Done? No -> Go back to step 2 and start over. 6. Finished with implementation.
mvn package	Runs unit tests and packages modules in their respective target directories, for example: • `all-in-one/repo-amp/target/1.0-SNAPSHOT/repo-amp-1.0-SNAPSHOT.amp` • `all-in-one/share-amp/target/1.0-SNAPSHOT/share-amp-1.0-SNAPSHOT.amp` • `all-in-one/repo/target/1.0-SNAPSHOT/repo-1.0-SNAPSHOT.war`, contains repo-amp-1.0-SNAPSHOT.amp from local maven repo (not the just packed version) • `all-in-one/share/target/1.0-SNAPSHOT/share-1.0-SNAPSHOT.war`, contains share-amp-1.0-SNAPSHOT.amp from local maven repo (not the just packed version) This does not apply these newly packaged AMPs to their respective WARs, use `mvn install` for that.

Command	Description
mvn install	Like `mvn package` but also installs artifacts in local Maven repository, for example: • `.m2/repository/com/acme/repo-amp/1.0-SNAPSHOT/repo-amp-1.0-SNAPSHOT.amp` • `.m2/repository/com/acme/share-amp/1.0-SNAPSHOT/share-amp-1.0-SNAPSHOT.amp` • `.m2/repository/com/acme/repo/1.0-SNAPSHOT/repo-1.0-SNAPSHOT.war`, contains the repo-amp-1.0-SNAPSHOT.amp • `.m2/repository/com/acme/share/1.0-SNAPSHOT/share-1.0-SNAPSHOT.war`, contains the share-amp-1.0-SNAPSHOT.amp Where these artifacts can be accessed by other local projects that depend on them.
mvn install -DskipTests=true	Like `mvn install` but skips unit tests.
mvn install -Prun	Like `run.sh or run.bat` but does not configure JVM memory and Spring Loaded if you have not configured it in `MAVEN_OPTS`, see set up MAVEN_OPTS. If you use Alfresco One, see the next command.
mvn install -Prun,enterprise	Like `mvn install -Prun` but uses Alfresco One (Enterprise) artifacts. Note you need to have set up access to the private repository containing the Alfresco One artifacts.

Command	Description
mvn clean install -Prun,regression-testing	Runs regression testing of the Alfresco Share UI (`share.war`). Uses the Alfresco internal Selenium Page Object (PO) based tests. This is very useful when you have developed a lot of customizations for the Share UI and you want to make sure you have not broken any standard Share UI functionality. Typically run this from a CI server (or better a Selenium-Grid) to test for regression of the standard Alfresco Share UI. The Selenium WebDriver is configured to use FireFox (FF) by default, so you need to have FF installed for the regression tests to be able to run. Use version 35 or newer. It is also highly recommended that the workstation that is running the regression tests is not being worked on at the same time as the tests are running, as that can affect the outcome of the tests. This command assumes that you are developing for the Alfresco Community Edition. If you use Alfresco One, you need to update the maven command so it uses the 'enterprise' profile and the Alfresco One `share-po` library: `mvn clean install -Prun,enterprise,regression-testing`. Also, make sure that you have installed an enterprise license in the `repo` project, otherwise the system will be in read-only mode and loads of tests will not pass.

Command	Description
mvn clean install -Prun,functional-testing	Runs functional testing of the Alfresco Share UI customizations that you have developed, such as pages and Dashlets. For information about how to write these tests, see the example test called `share-amp/src/test/java/{package-path}/demoamp/DemoPageTestIT` and its Page Object class called `share-amp/src/test/java/{package-path}/demoamp/po/DemoPage.java`. The Selenium WebDriver is configured to use FireFox (FF) by default, so you need to have FF installed for the functional tests to be able to run. Use version 35 or newer. It is also highly recommended that the workstation that is running the functional tests is not being worked on at the same time as the tests are running, as that can affect the outcome of the tests. This command assumes that you are developing for Alfresco Community Edition. If you use Alfresco One, you need to update the maven command so it uses the 'enterprise' profile and the Alfresco One `share-po` library: `mvn clean install -Prun,enterprise,functional-testing`.
mvn clean -Ppurge	Removes H2 database (with metadata), alf_data (with content files and index files), and log files. Useful to purge the development repo (by default self contained in `${project.basedir}/alf_data_dev`. This is an important command to use if you change significant settings in your project. For example, if you change Alfresco from Community Edition to Alfresco One. It is important to purge databases and other data that might otherwise be persisted. The `purge` profile cannot be used together with the `run` profile.

Rapid Application Development (RAD)

These tutorials cover how to employ the RAD features of the Alfresco SDK.

Rapid Application Development (RAD) and Test Driven Development (TDD) are big goals for the Alfresco SDK. The SDK is designed to support the hot reloading of code (via Spring Loaded) so that you can modify JavaScript, FreeMarker and Java code, and have the changes take effect without having to click the **Refresh Web Scripts** button, restart Alfresco Tomcat, or restart anything else.

For example, in your SDK project, you can change test code, re-run your test, and the results will be displayed immediately. This allows for Test Driven Development (TDD).

The hot reloading above all saves you time as a developer. No more waiting around for Alfresco Tomcat restarts to see your code changes take effect.

It is assumed that you will work through the tutorials in the order in which they are presented.

Importing SDK projects into Eclipse

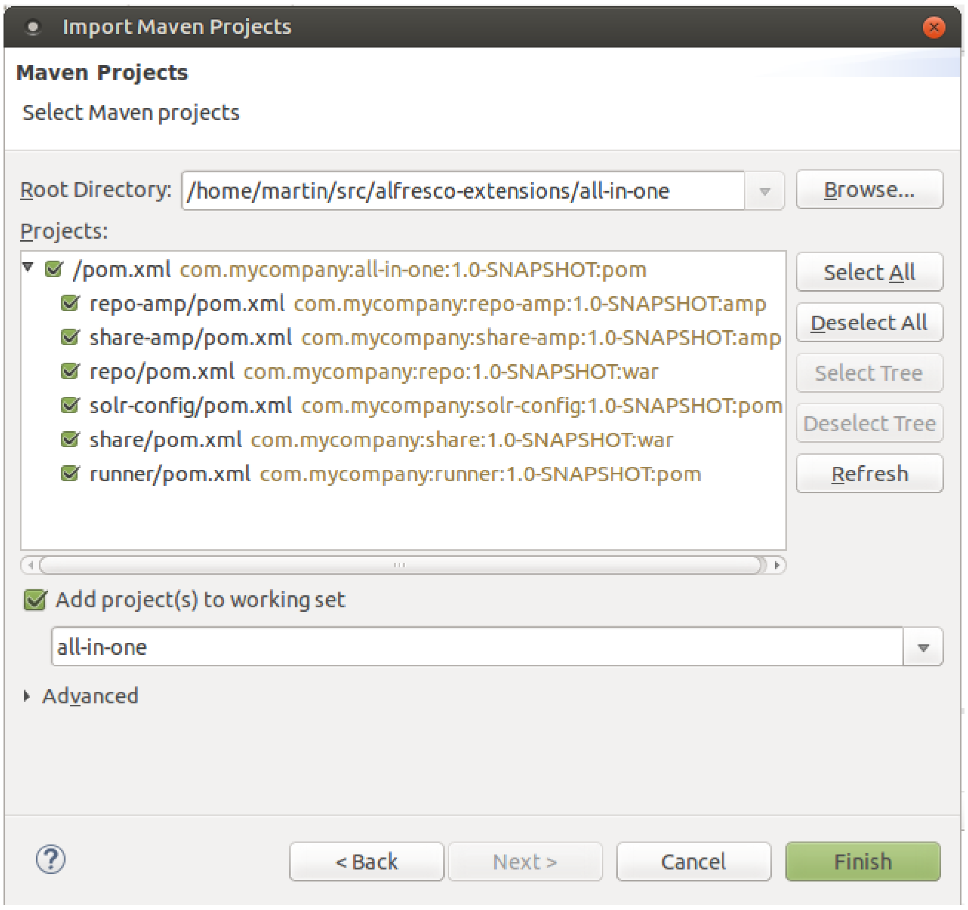
The Alfresco SDK is designed to work well with Eclipse. This support includes the ability to import existing SDK projects (created via the command line) into Eclipse.

You should have completed Installing and Configuring software and generated a project.

You will learn how to import an existing Maven project into Eclipse.

1. In Eclipse, from the main menu select **File** > **Import...** > **Maven** > **Existing Maven Projects**.
2. Click **Next >**
3. Click **Browse...>**
4. Navigate to the directory where your Maven project is located. For example `alfresco-extensions/all-in-one`.

   You should see a dialog looking something like this when importing an All-in-One (AIO) project:



Import Maven Projects

Maven Projects

Select Maven projects

Root Directory: /home/martin/src/alfresco-extensions/all-in-one

Projects:

- ✓ /pom.xml com.mycompany:all-in-one:1.0-SNAPSHOT:pom
  - ✓ repo-amp/pom.xml com.mycompany:repo-amp:1.0-SNAPSHOT:amp
  - ✓ share-amp/pom.xml com.mycompany:share-amp:1.0-SNAPSHOT:amp
  - ✓ repo/pom.xml com.mycompany:repo:1.0-SNAPSHOT:war
  - ✓ solr-config/pom.xml com.mycompany:solr-config:1.0-SNAPSHOT:pom
  - ✓ share/pom.xml com.mycompany:share:1.0-SNAPSHOT:war
  - ✓ runner/pom.xml com.mycompany:runner:1.0-SNAPSHOT:pom

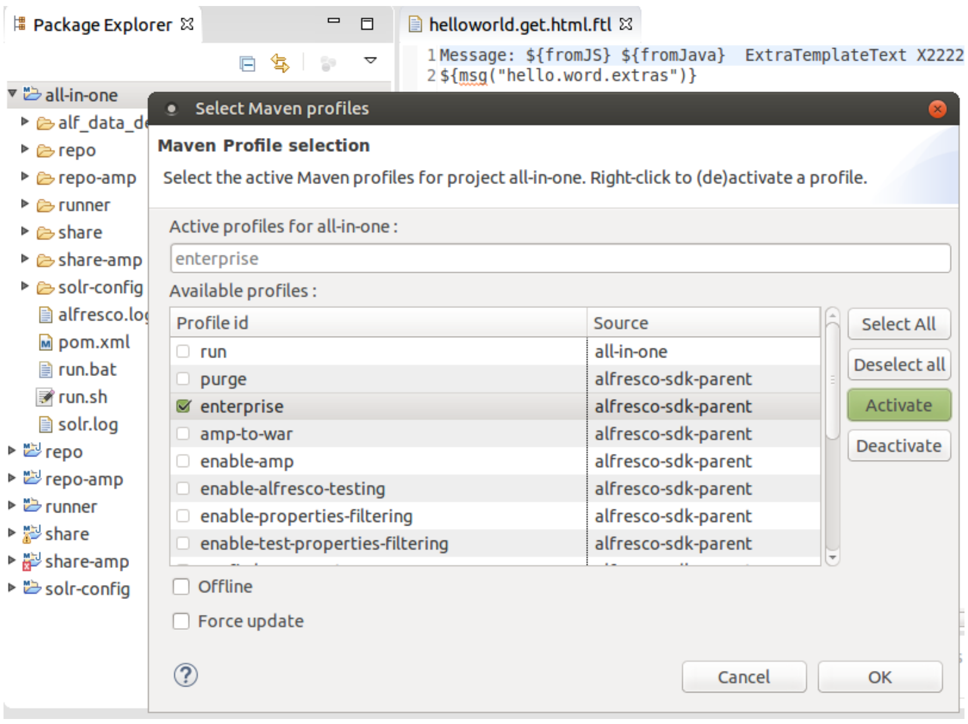
✓ Add project(s) to working set

all-in-one

5. Click **Finish**

   The project, and any sub-projects, will now be imported.
6. Enable Alfresco Enterprise edition **(Optional)**

   If your project is using the Enterprise edition of Alfresco you also want Eclipse to load the enterprise versions of the Alfresco WARs and related libraries. You can do this by enable the `enterprise` profile. In the **Package Explorer** view to the left, right click on the **all-in-one** project, then select **Maven** from the popup menu. Now in the next popup menu choose **Select Maven Profiles...**. In the dialog that appears select the `enterprise` profile, you should see a dialog looking something like this now:



Select Maven profiles

Maven Profile selection

Select the active Maven profiles for project all-in-one. Right-click to (de)activate a profile.

Active profiles for all-in-one : enterprise

Available profiles :

Profile id	Source
☐ run	all-in-one
☐ purge	alfresco-sdk-parent
✓ enterprise	alfresco-sdk-parent
☐ amp-to-war	alfresco-sdk-parent
☐ enable-amp	alfresco-sdk-parent
☐ enable-alfresco-testing	alfresco-sdk-parent
☐ enable-properties-filtering	alfresco-sdk-parent
☐ enable-test-properties-filtering	alfresco-sdk-parent

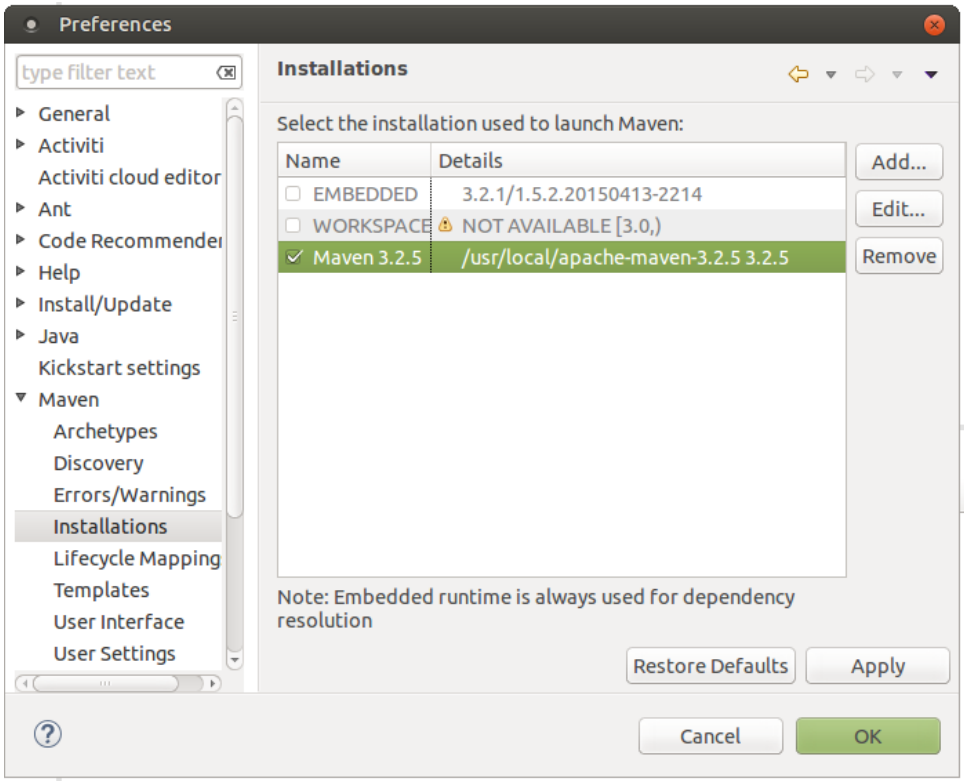
☐ Offline

☐ Force update

```
Message: ${fromJS} ${fromJava}  ExtraTemplateText X2222
${msg("hello.word.extras")}
```

7. Configure external Maven

   Use the external Maven installation. In the **Window** top menu to the right in Eclipse, click on the **Preferences** sub-menu item at the bottom, then select **Maven** from the pop-up dialog menu. Now in the Maven sub-menu choose **Installations**. In the dialog that appears to the right select the external Maven installation (or add it if it is not in the list), you should see a dialog looking something like this now:



Preferences

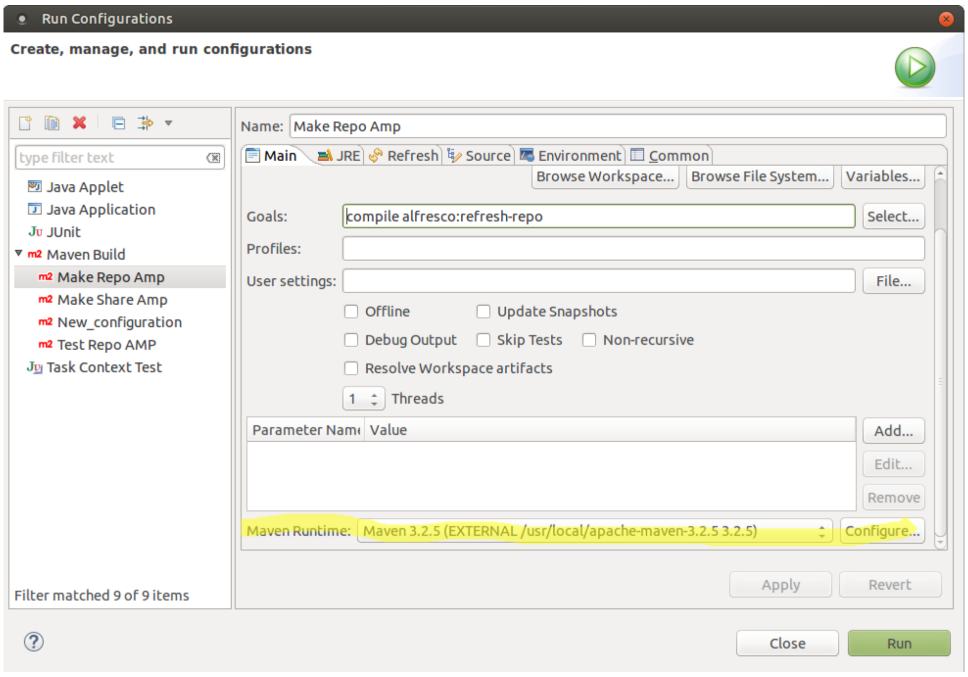
Installations

Select the installation used to launch Maven:

Name	Details
☐ EMBEDDED	3.2.1/1.5.2.20150413-2214
☐ WORKSPACE	⚠ NOT AVAILABLE [3.0,)
✓ Maven 3.2.5	/usr/local/apache-maven-3.2.5 3.2.5

Note: Embedded runtime is always used for dependency resolution

Now close this **Preferences** dialog and open the **Run Configurations** dialog. Make sure the external Maven configuration is used:



Name: Make Repo Amp

Goals: compile alfresco:refresh-repo

Profiles:

User settings:

☐ Offline ☐ Update Snapshots

☐ Debug Output ☐ Skip Tests ☐ Non-recursive

☐ Resolve Workspace artifacts

1 Threads

Maven Runtime: Maven 3.2.5 (EXTERNAL /usr/local/apache-maven-3.2.5 3.2.5)

If the EMBEDDED Maven installation is used then you might encounter the EXCEPTION_ACCESS_VIOLATION JRE error when running.

You have seen how to import your SDK project(s) into Eclipse. You can now build, run and debug them in the usual way, using RAD and TDD techniques.

Rapid Application Development in Eclipse (Hot reloading)

Hot reloading is the ability to modify your application's code, and view the changes without having to restart Alfresco Tomcat. This allows for significant savings in development time that would otherwise be wasted restarting Tomcat. Hot reloading is the key to enabling Rapid Application Development (RAD) and Test Driven Development (TDD).

You should have an extension project imported, see importing a project into Eclipse.

In this tutorial you will see how changes to your code can be carried out without having to restart Alfresco Tomcat. This tutorial demonstrates hot reloading of JavaScript, FreeMarker template, and Java code. There are three components that work together to enable the best RAD experience:

1. *Spring Loaded*: takes care of hot-reloading any Java class files that we have changed.
2. *Refresh Repository Script*: This is a script that will POST a request to the Alfresco Repository Web Application (i.e. alfresco.war) telling it to refresh the Repo Web Script container, so any changes to files related to Web Scripts will be picked up.
3. *Refresh Share Script*: This is a script that will POST a request to the Alfresco Share Web Application (i.e. share.war) telling it to refresh the Surf Web Script container, so any changes to files related to Surf Web Scripts will be picked up. This script will also clear the resource dependency caches, so JS changes etc are picked up.

Start an instance of Alfresco Tomcat that will be used for hot-reloading.

1. Use the run script to start the Application server with the Alfresco extension project deployed, for example:

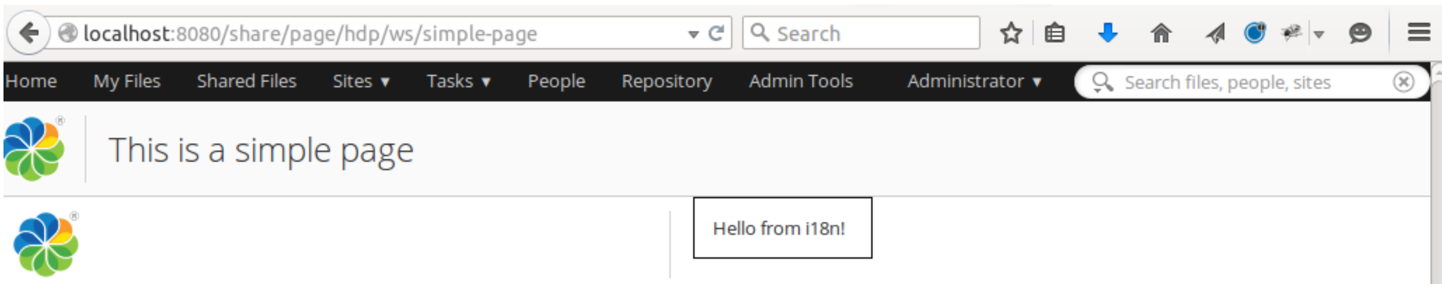
```
alfresco-extensions/all-in-one$ ./run.sh
```

This is usually done outside the IDE.

2. Test the custom Repository Web Script

   The All-in-One project (and the Repository AMP project) have a sample Web Script included. You can invoke it by pointing your web browser at `http://localhost:8080/alfresco/service/sample/helloworld`. If you need to login then use `admin` with password `admin`. Running this Web Script produces the output "Message: Hello World from JS! HelloFromJava".
3. Test the custom Share Aikau Page

   The All-in-One project (and the Share AMP project) have a sample Aikau page included. You can display it by pointing your web browser at `http://localhost:8080/share/page/hdp/ws/simple-page`. If you need to login then use `admin` with password `admin`. The page should display as follows:



Enabling Rapid Application Development (RAD) in Eclipse.

4. This is enabled by default and there is no specific configuration needed. If you want to have more control over when web applications are refreshed, then see the last tutorial about Run Configurations.

Testing RAD when doing Repository customizations (alfresco.war).

5. In the Package Explorer, navigate to and expand the `all-in-one/repo-amp/src/main/amp/config/alfresco/extension/templates/webscripts` folder
6. Locate the `helloworld.get.js` file and load it into the editor by double-clicking it.

   This is the controller for the Web Script that we tried after starting the server. Update the controller code by adding an 'UPDATED' string as follows:

```
model["fromJS"] = "Hello World from JS! UPDATED";
```

7. Now build(Make) the project by saving, i.e. click **Ctrl+S**

   The application server log should display messages about the web scripts being refreshed:

```
2015-05-12 11:13:40,652  INFO
 [extensions.webscripts.DeclarativeRegistry] [http-bio-8080-exec-9]
 Registered 407 Web Scripts (+0 failed), 549 URLs
 2015-05-12 11:13:40,653  INFO
 [extensions.webscripts.DeclarativeRegistry] [http-bio-8080-exec-9]
 Registered 1 Package Description Documents (+0 failed)
 2015-05-12 11:13:40,653  INFO
 [extensions.webscripts.DeclarativeRegistry] [http-bio-8080-exec-9]
 Registered 0 Schema Description Documents (+0 failed)
 2015-05-12 11:13:40,656  INFO
 [extensions.webscripts.AbstractRuntimeContainer] [http-bio-8080-
exec-9] Initialised Repository Web Script Container (in 2215.1865ms)
 2015-05-12 11:13:42,414  INFO
 [extensions.webscripts.DeclarativeRegistry]
 [asynchronouslyRefreshedCacheThreadPool1] Registered 407 Web Scripts
 (+0 failed), 549 URLs
 2015-05-12 11:13:42,414  INFO
 [extensions.webscripts.DeclarativeRegistry]
 [asynchronouslyRefreshedCacheThreadPool1] Registered 1 Package
 Description Documents (+0 failed)
 2015-05-12 11:13:42,414  INFO
 [extensions.webscripts.DeclarativeRegistry]
 [asynchronouslyRefreshedCacheThreadPool1] Registered 0 Schema
 Description Documents (+0 failed)
```

Note that there is no output in the Eclipse console, or other window. Also, it is only the Repository Application (i.e. alfresco.war) that is being refreshed, the Share application is not touched.

8. Refresh the `http://localhost:8080/alfresco/service/sample/helloworld` Repository Web Script page from the Browser

   The output from the Web Script should change to "Message: Hello World from JS! UPDATED HelloFromJava". Note that there is no need to restart the application server, just a Make of the project, and a refresh of the Web Script page from the browser (you are basically invoking the Web Script again and the update should be immediately visible).

9. Now locate the `helloworld.get.html.ftl` file and load it into the editor by double-clicking it.

   This is the FreeMarker template for the Web Script. Update the template by adding an 'ExtraTemplateText' string as follows:

   ```
   Message: ${fromJS} ${fromJava} ExtraTemplateText
   ```

10. Now build(Make) the project by clicking **Ctrl+S**

    The application server log should display messages about the web scripts being refreshed.

11. Refresh the `http://localhost:8080/alfresco/service/sample/helloworld` Repository Web Script page from the Browser

    The output from the Web Script should change to "Message: Hello World from JS! UPDATED HelloFromJava ExtraTemplateText ". Again, note that there is no need to restart the application server, only a Make of the project and a refresh of the Web Script page are necessary.

12. In the next demo we will add a properties file for the Web Script, create a `helloworld.get.properties` file next to the other files we have been working with.

    The properties file should have one property as follows:

    ```
    hello.word.extras=Extra Stuff From Props
    ```

13. Add this property to the template, open up `helloworld.get.html.ftl`.

    The FreeMarker template should now look like this:

    ```
    Message: ${fromJS} ${fromJava}  ExtraTemplateText
    ```

```
${msg("hello.word.extras")}
```

14. Now build(Make) the project by clicking **Ctrl+S**

    The application server log should display messages about the web scripts being refreshed.

15. Refresh the `http://localhost:8080/alfresco/service/sample/helloworld` Repository Web Script page from the Browser

    The output of the Web Script should change to "Message: Hello World from JS! UPDATED HelloFromJava ExtraTemplateText Extra Stuff From Props". No restart of application server should be needed, just a Make of the project and a refresh of the Web Script page from the browser.

16. In the last Web Script demo we will change the Java controller

17. Open up `all-in-one/repo-amp/src/main/java/{your package path}/demoamp/HelloWorldWebScript.java`.

    Change the property text as follows:

```
public class HelloWorldWebScript extends DeclarativeWebScript {
    protected Map<String, Object> executeImpl(
            WebScriptRequest req, Status status, Cache cache) {
        Map<String, Object> model = new HashMap<String, Object>();
        model.put("fromJava", "HelloFromJavaUPDATED");
        return model;
    }
}
```

18. Now build(Make) the project by clicking **Ctrl+S**

    The application server log should display messages about the web scripts being refreshed.

19. Refresh the `http://localhost:8080/alfresco/service/sample/helloworld` Repository Web Script page from the Browser

    The output of the Web Script should change to "Message: Hello World from JS! UPDATED HelloFromJavaUPDATED ExtraTemplateText Extra Stuff From Props". No restart of application server should be needed, just a Make of the project and a refresh of the Web Script page from the browser.

Test Driven Development (TDD) and RAD when doing Repository customizations (alfresco.war).

20. In the Package Explorer expand `all-in-one/repo-amp/src/test/java/{your package path}/demoamp/test` and then locate the `DemoComponentTest.java` source file.
21. Load it into the editor by double-clicking it.
22. Now set up a Run Configuration to run `repo-amp` tests. From the main menu select **Run** > **Run Configurations...**.
23. In the **Run Configurations** dialog, select **Maven Build** in the left list. Then right click on it and select **New**.
24. Set the **Name** field of the configuration to "Test Repo AMP".
25. Set the **Base Directory** field of the configuration to "${workspace_loc:/repo-amp}".
26. Set the **Goals** field "test".
27. In the **JRE** tab set JDK 8.
28. Click **Apply**.
29. Now run the test by selecting **Run**.

The test will run, and three tests will pass, the Console will have logs as follows:

```
[INFO] Scanning for projects...
[INFO]

[INFO]
 ------------------------------------------------------------------------
[INFO] Building Alfresco Repository AMP Module 1.0-SNAPSHOT
[INFO]
 ------------------------------------------------------------------------
[INFO]
[INFO] --- alfresco-maven-plugin:2.1.0-SNAPSHOT:set-version (default-
set-version) @ repo-amp ---
[INFO] Removed -SNAPSHOT suffix from version - 1.0
[INFO] Added timestamp to version - 1.0.1505121136
[INFO]
[INFO] --- build-helper-maven-plugin:1.9.1:add-test-resource (add-env-
test-properties) @ repo-amp ---
[INFO]
[INFO] --- maven-resources-plugin:2.7:resources (default-resources) @
 repo-amp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/martin/src/alfresco-
extensions/all-in-one/repo-amp/src/main/resources
[INFO] Copying 16 resources to ../repo-amp
[INFO]
[INFO] --- yuicompressor-maven-plugin:1.5.1:compress (compress-js) @
 repo-amp ---
[INFO] nothing to do, /home/martin/src/alfresco-extensions/all-in-
one/repo-amp/target/classes/../repo-amp/web/scripts/demoamp-min.js is
 younger than original, use 'force' option or clean your target
[INFO] nb warnings: 0, nb errors: 0
[INFO]
[INFO] --- alfresco-maven-plugin:2.1.0-SNAPSHOT:refresh (refresh-
webscripts-repo-and-share) @ repo-amp ---
[INFO] Successfull Refresh Web Scripts for Alfresco Repository
[INFO]
[INFO] --- maven-compiler-plugin:3.2:compile (default-compile) @ repo-
amp ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.7:testResources (default-
testResources) @ repo-amp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 2 resources
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-resources-plugin:2.7:copy-resources (add-module-
properties-to-test-classpath) @ repo-amp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource to alfresco/module/repo-amp
[INFO]
[INFO] --- maven-resources-plugin:2.7:copy-resources (add-module-
config-to-test-classpath) @ repo-amp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 11 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.2:testCompile (default-testCompile)
 @ repo-amp ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.18:test (default-test) @ repo-amp
 ---
[INFO] Surefire report directory: /home/martin/src/alfresco-extensions/
all-in-one/repo-amp/target/surefire-reports
```

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running org.alfresco.allinone.demoamp.test.DemoComponentTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.088
 sec - in org.alfresco.allinone.demoamp.test.DemoComponentTest

Results :

Tests run: 3, Failures: 0, Errors: 0, Skipped: 0

[INFO]
 ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO]
 ------------------------------------------------------------------------
[INFO] Total time: 7.232 s
[INFO] Finished at: 2015-05-12T11:36:13+01:00
[INFO] Final Memory: 27M/338M
[INFO]
 ------------------------------------------------------------------------
```

30. Now, in `DemoComponentTest.java`, modify one of the tests so that it will fail. For example, you could change the line `assertEquals(7, childNodeCount);` to `assertEquals(8, childNodeCount);`
31. Now run the test again by selecting **Run** from main menu and then from **Run History** select **Run 'Test Repo AMP'**.

    Note the test will run again and this time fail. But you did not need to restart Alfresco. This demonstrates hot reloading of Java code.
32. Change the code back and re-run the test.

    Now you will see that all tests pass. The code has re-run without any reloading of Alfresco! This allows for Test Driven Development with very low overhead.

Testing RAD when doing Share customizations (share.war).

33. In the Package Explorer, navigate to and expand the `all-in-one/share-amp/src/main/amp/config/alfresco/web-extension/site-webscripts/com/example/pages` folder
34. Locate the `simple-page.get.js` file and load it into the editor by double-clicking it.

    This is the controller for the Aikau Page Web Script that we tried after starting the server. Update the controller code by adding an 'UPDATED' string as follows to the page title, also change the layout from HorizontalWidgets to VerticalWidgets:

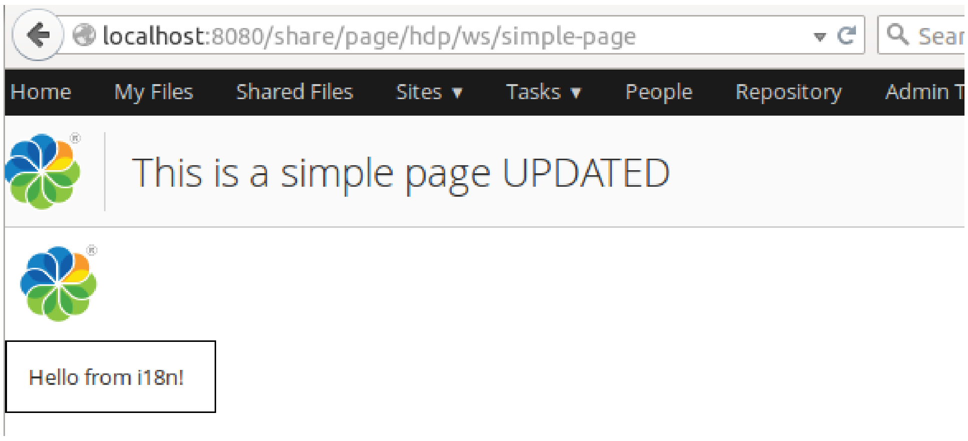
```
model.jsonModel = {
    widgets: [{
        id: "SET_PAGE_TITLE",
        name: "alfresco/header/SetTitle",
        config: {
            title: "This is a simple page UPDATED"
        }
    },
        {
            id: "MY_HORIZONTAL_WIDGET_LAYOUT",
            name: "alfresco/layout/VerticalWidgets",
            config: {
                widgetWidth: 50,
                widgets: [
                    {
                        id: "DEMO_SIMPLE_LOGO",
                        name: "alfresco/logo/Logo",
                        config: {
                            logoClasses: "alfresco-logo-only"
                        }
                    },
                    {
                        id: "DEMO_SIMPLE_MSG",
                        name: "example/widgets/TemplateWidget"
                    }
                ]
            }
        }]
};
```

35. Now build(Make) the project by clicking **Ctrl+S**

    The Share web application will now have the web script container refreshed and the resouce cache cleared.

36. Refresh the `http://localhost:8080/share/page/hdp/ws/simple-page` Aikau Page from the Browser

    The page should now display as follows:



This is a simple page UPDATED

Hello from i18n!

37. Now, let's update some HTML, CSS, and Properties for the sample widget that is used by the Aikau page, navigate to and expand the `all-in-one/share-amp/src/main/amp/web/js/example/widgets` folder

38. Locate the `css/TemplateWidget.css` file and load it into the editor by double-clicking it.

    This is the Stylesheet for the Aikau Widget. Update the widget style as follows:

```
.my-template-widget {
    border: 2px #000000 solid;
    padding: 1em;
    width: 100px;
    color: white;
    background-color: blue;
}
```

39. Then locate the `i18n/TemplateWidget.properties` file and load it into the editor by double-clicking it.

    This is the resource file for the Aikau Widget. Update the properties as follows:

```
hello-label=Hello from i18n UPDATED!
hello-test=Going to use this label too now!
```

40. Then locate the `templates/TemplateWidget.html` file and load it into the editor by double-clicking it.

    This is the HTML template file for the Aikau Widget. Update so it looks as follows:

    ```
    <div class="my-template-widget">${greeting} and ${greeting2}</div>
    ```

41. And finally, locate the `TemplateWidget.js` file and load it into the editor by double-clicking it.

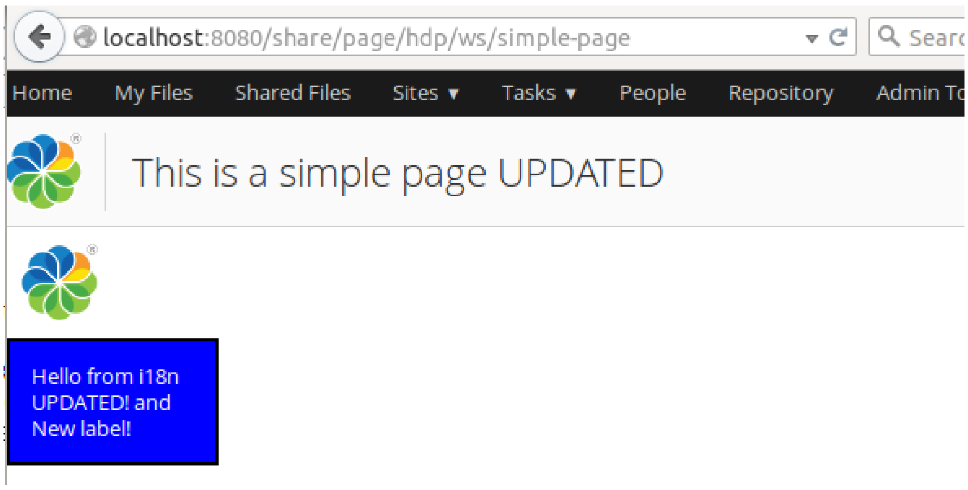
    This is the main JavaScript implementation for the Aikau Widget. Update so it also sets the new property used in template:

    ```
    define(["dojo/_base/declare",
            "dijit/_WidgetBase",
            "alfresco/core/Core",
            "dijit/_TemplatedMixin",
            "dojo/text!./templates/TemplateWidget.html"
        ],
        function(declare, _Widget, Core, _Templated, template) {
            return declare([_Widget, Core, _Templated], {
                templateString: template,
                i18nRequirements: [ {i18nFile: "./i18n/
    TemplateWidget.properties"} ],
                cssRequirements: [{cssFile:"./css/TemplateWidget.css"}],

                buildRendering: function
     example_widgets_TemplateWidget__buildRendering() {
                    this.greeting = this.message('hello-label');
                    this.greeting2 = this.message('hello-test');

                    this.inherited(arguments);

                }
            });
    });
    ```

42. Now build(Make) the project by clicking **Ctrl+S**.

43. Refresh the `http://localhost:8080/share/page/hdp/ws/simple-page` Aikau Page from the Browser.

    The page should now display as follows:


localhost:8080/share/page/hdp/ws/simple-page
Searc
Home
My Files
Shared Files
Sites
Tasks
People
Repository
Admin To
This is a simple page UPDATED
Hello from i18n UPDATED! and New label!

Enabling RAD in Eclipse with Run Configurations.

44. Introduction

In this article we have seen how we can achieve Rapid Application Development within Eclipse by having the alfresco maven plugin refresh goals executed automatically (magically) after a Make, which is triggered by saving the file. This auto-refresh feature is enabled by default when you use Eclipse. If you don't want that, and instead want to have more control over when web application refreshs happens etc, then you can use a Run Configuration instead and disable auto-refresh. In the following tutorial you will see how run configurations can be used for better control of when the refresh call is being made.

45. Start by disabling auto-refresh, set the following property in the top project POM (i.e. in `alfresco-extensions/all-in-one/pom.xml`):

    ```
            ...
            <share.client.url>http://localhost:8080/share</
    share.client.url>

            <!--  Turn off auto-refresh of web applications -->
            <maven.alfresco.refresh.mode>none</maven.alfresco.refresh.mode>
        </properties>
    ```

    Besides `none`, the other values for this property are:

    - `auto` - (**default**) Checks packaging and app.amp.client.war.artifactId to determine if it should refresh Respository (alfresco.war) or Share (share.war)
    - `both` - Will refresh both Respository and Share
    - `share` - Refresh only Share
    - `repo` - Refresh only Respository
    - `none` - Disables refreshing all together
46. Then set up a Run Configuration to run `repo-amp` builds and Repository webapp refresh (alfresco.war). From the main menu select **Run** > **Run Configurations...**.
47. In the **Run Configurations** dialog, select **Maven Build** in the left list. Then right click on it and select **New**.
48. Set the **Name** field of the configuration to "Make Repo AMP".
49. Set the **Base Directory** field of the configuration to "${workspace_loc:/repo-amp}".
50. Set the **Goals** field "compile alfresco:refresh-repo".
51. In the **JRE** tab set JDK 8.
52. Click **Apply**.
53. Now run the repo-amp build by selecting **Run**.

    The build will run with the following result, the Eclipse Console will output:

    ```
    [INFO] Scanning for projects...
    [INFO]

    [INFO]
     ------------------------------------------------------------------------
    [INFO] Building Alfresco Repository AMP Module 1.0-SNAPSHOT
    [INFO]
     ------------------------------------------------------------------------
    [INFO]
    [INFO] --- alfresco-maven-plugin:2.1.0-SNAPSHOT:set-version (default-
    set-version) @ repo-amp ---
    ```

```
[INFO] Removed -SNAPSHOT suffix from version - 1.0
[INFO] Added timestamp to version - 1.0.1505121201
[INFO]
[INFO] --- build-helper-maven-plugin:1.9.1:add-test-resource (add-env-
test-properties) @ repo-amp ---
[INFO]
[INFO] --- maven-resources-plugin:2.7:resources (default-resources) @
 repo-amp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/martin/src/alfresco-
extensions/all-in-one/repo-amp/src/main/resources
[INFO] Copying 16 resources to ../repo-amp
[INFO]
[INFO] --- yuicompressor-maven-plugin:1.5.1:compress (compress-js) @
 repo-amp ---
[INFO] nothing to do, /home/martin/src/alfresco-extensions/all-in-
one/repo-amp/target/classes/../repo-amp/web/scripts/demoamp-min.js is
 younger than original, use 'force' option or clean your target
[INFO] nb warnings: 0, nb errors: 0
[INFO]
[INFO] --- alfresco-maven-plugin:2.1.0-SNAPSHOT:refresh (refresh-
webscripts-repo-and-share) @ repo-amp ---
[INFO] Successfull Refresh Web Scripts for Alfresco Repository
[INFO]
[INFO] --- maven-compiler-plugin:3.2:compile (default-compile) @ repo-
amp ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- alfresco-maven-plugin:2.1.0-SNAPSHOT:refresh-repo (default-
cli) @ repo-amp ---
[INFO] Successfull Refresh Web Scripts for Alfresco Repository
[INFO]
 ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO]
 ------------------------------------------------------------------------
[INFO] Total time: 8.440 s
[INFO] Finished at: 2015-05-12T12:01:13+01:00
[INFO] Final Memory: 29M/342M
[INFO]
 ------------------------------------------------------------------------
```

Note that only the `repo-amp` is built and only the Repository webapp is refreshed (alfresco.war), the Share application is not touched. This demonstrates how you can have better control of the build and refresh when you have a many different AMPs. You can create a similar run configuration for the `share-amp` project, and for any other AMP project.

In this tutorial you have seen how to add and modify code within Eclipse and and then see how these changes take effect immediately, without the need to manually restart or refresh any Alfresco Web Applications.

Importing SDK projects into IntelliJ IDEA

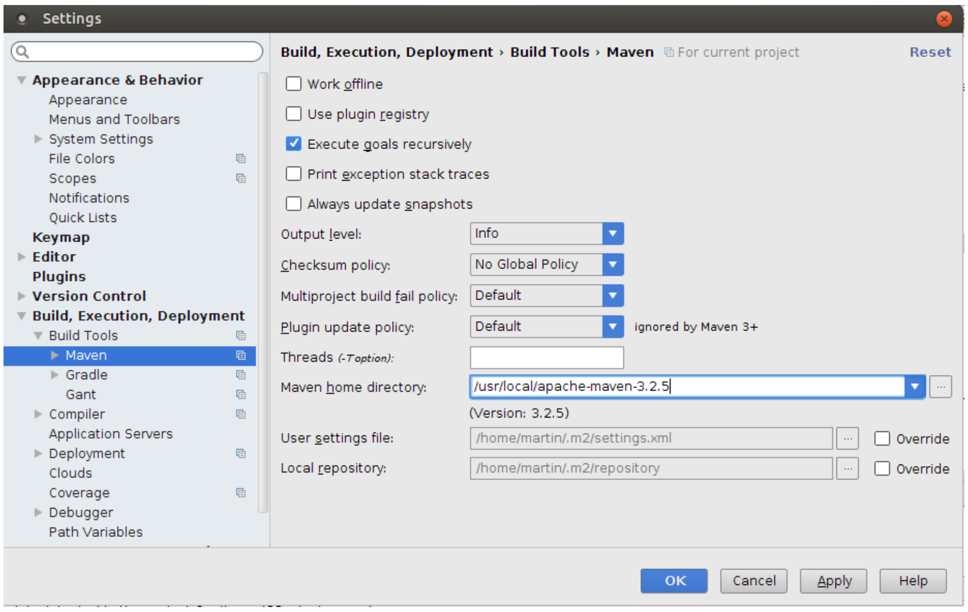
The Alfresco SDK is designed to work well with IntelliJ IDEA. This support includes the ability to import existing SDK projects (created via the command line) into IDEA.

You should have completed Installing and configuring software and generated a project.

You will learn how to import an existing Maven project into IDEA.

1. First, make sure IDEA is using the correct Maven installation (3.2.5 or newer). In the main menu, select **File** > **Settings...**.

   In the IDEA Settings dialog you should see something like this:

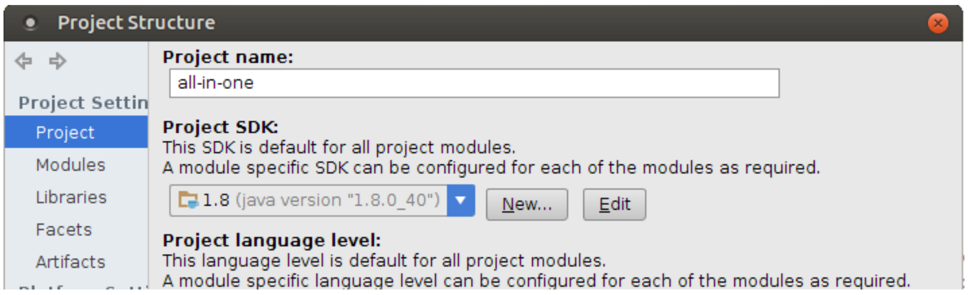

Build, Execution, Deployment › Build Tools › Maven — For current project

- [ ] Work offline
- [ ] Use plugin registry
- [x] Execute goals recursively
- [ ] Print exception stack traces
- [ ] Always update snapshots

Output level:	Info
Checksum policy:	No Global Policy
Multiproject build fail policy:	Default
Plugin update policy:	Default (ignored by Maven 3+)
Threads (-T option):	
Maven home directory:	/usr/local/apache-maven-3.2.5 (Version: 3.2.5)
User settings file:	/home/martin/.m2/settings.xml [ ] Override
Local repository:	/home/martin/.m2/repository [ ] Override

2. Then check that you are using JDK 8, In the main menu, select **File** > **Project Structure...**.

   In the IDEA Project Structure dialog you should see something like this:


Project name: all-in-one

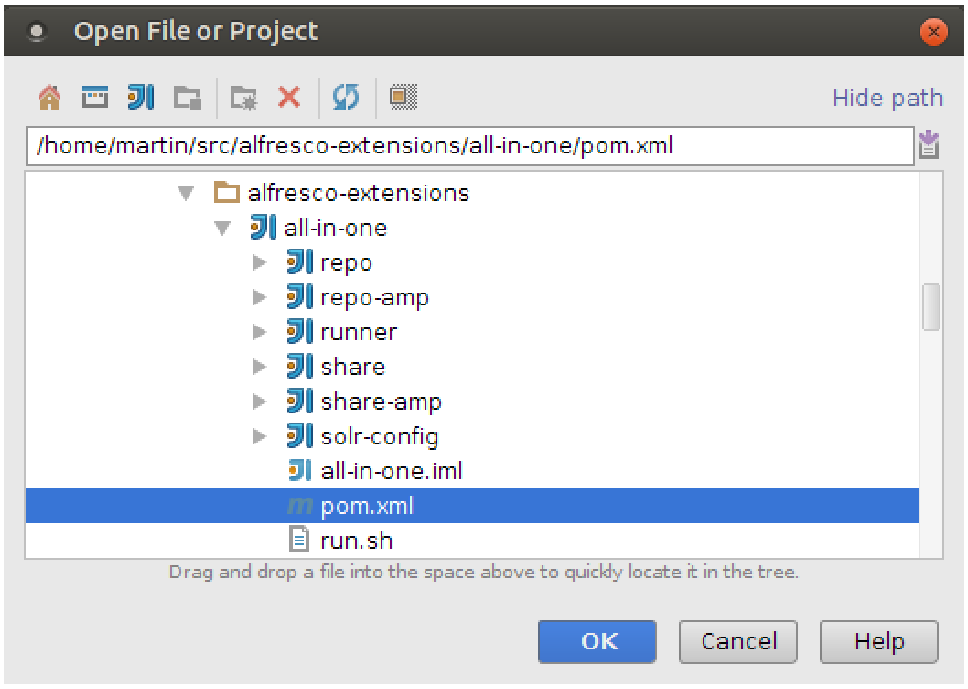
Project SDK:
This SDK is default for all project modules.
A module specific SDK can be configured for each of the modules as required.

1.8 (java version "1.8.0_40")

Project language level:
This language level is default for all project modules.
A module specific language level can be configured for each of the modules as required.

3. Now, in the main menu, select **File** > **Open...**
4. Navigate to where the project's parent pom is located, in this case the AIO parent pom.

   You should see a dialog looking something like this when you have located an All-in-One (AIO) project:



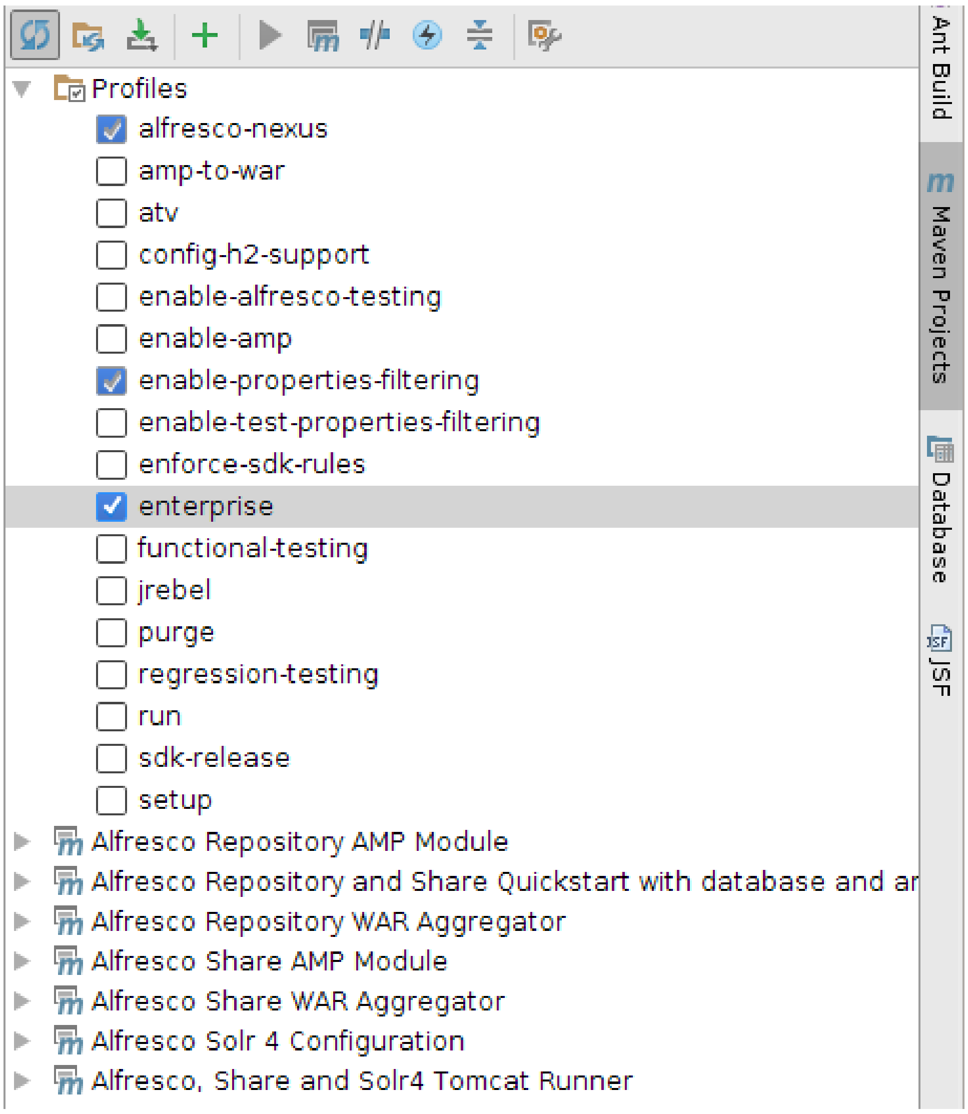
/home/martin/src/alfresco-extensions/all-in-one/pom.xml

alfresco-extensions
all-in-one
repo
repo-amp
runner
share
share-amp
solr-config
all-in-one.iml
pom.xml
run.sh

Drag and drop a file into the space above to quickly locate it in the tree.

5. Select the `pom.xml` file for the All-in-One parent project.
6. Click **OK**. The project is now imported and should appear in the **Project** tool view to the left.
7. Enable Alfresco Enterprise edition **(Optional)**

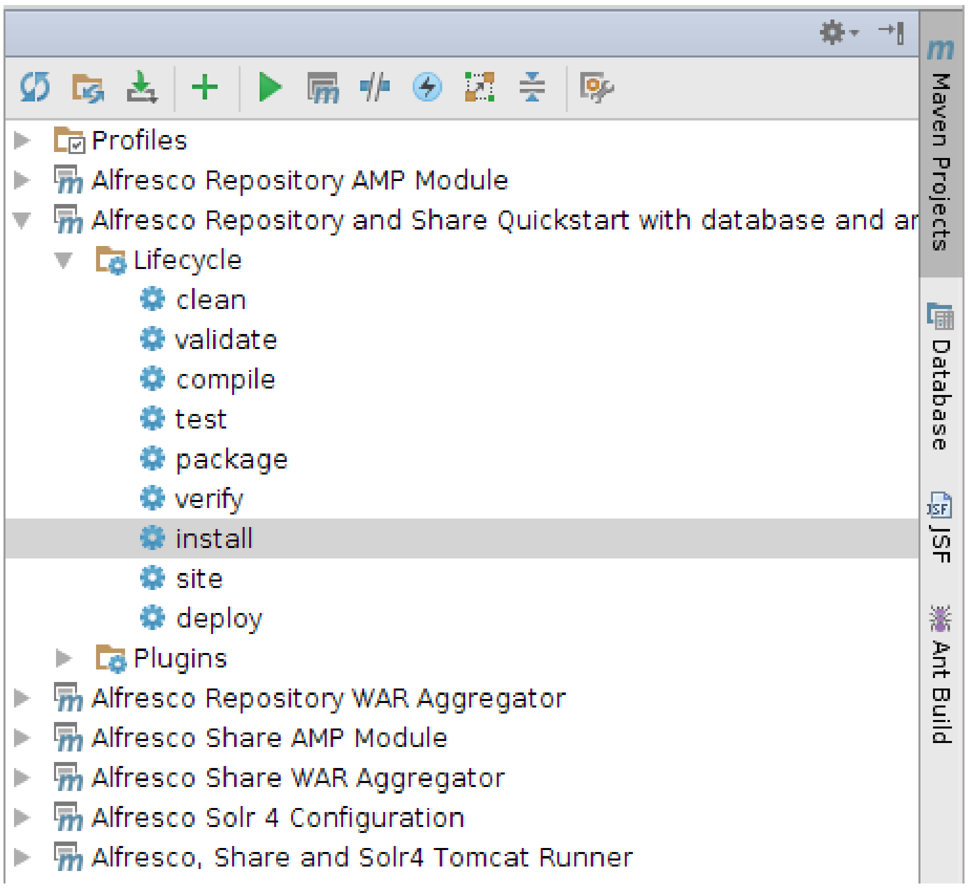
   If your project is using the Enterprise edition of Alfresco you also want IDEA to load the enterprise versions of the Alfresco WARs and related libraries. You can do this by enable the `enterprise` profile. In the **Maven Projects** tool view to the right expand the **Profiles** folder, then check the **enterprise** profile. You should see a dialog looking something like this:



```
Profiles
  [✓] alfresco-nexus
  [ ] amp-to-war
  [ ] atv
  [ ] config-h2-support
  [ ] enable-alfresco-testing
  [ ] enable-amp
  [✓] enable-properties-filtering
  [ ] enable-test-properties-filtering
  [ ] enforce-sdk-rules
  [✓] enterprise
  [ ] functional-testing
  [ ] jrebel
  [ ] purge
  [ ] regression-testing
  [ ] run
  [ ] sdk-release
  [ ] setup
Alfresco Repository AMP Module
Alfresco Repository and Share Quickstart with database and ar
Alfresco Repository WAR Aggregator
Alfresco Share AMP Module
Alfresco Share WAR Aggregator
Alfresco Solr 4 Configuration
Alfresco, Share and Solr4 Tomcat Runner
```

8. Now use Maven to build the All-in-One (AIO) project. Do this by executing the `install` command on the parent AIO project. In the **Maven Projects** tool view to the right expand the parent pom, then expand the **Lifecycle** folder.

   You should see a dialog looking something like this:



Profiles
Alfresco Repository AMP Module
Alfresco Repository and Share Quickstart with database and ar
Lifecycle
clean
validate
compile
test
package
verify
install
site
deploy
Plugins
Alfresco Repository WAR Aggregator
Alfresco Share AMP Module
Alfresco Share WAR Aggregator
Alfresco Solr 4 Configuration
Alfresco, Share and Solr4 Tomcat Runner

9. Double-click the `install` plugin goal.

   The project will build, and information will be displayed in the Console. You will see a message similar to the following, indicating that the project was successfully built.

```
/usr/lib/jvm/java-8-oracle/bin/java -Xms256m -Xmx1G -XX:PermSize=500m -
javaagent:/home/martin/libs/springloaded-1.2.3.RELEASE.jar -noverify -
Dmaven.home=/usr/local/apache-maven-3.2.5 ......  install

[INFO] Reactor Summary:
[INFO]
[INFO] Alfresco Repository and Share Quickstart with database and an
 embedded Tomcat runner. SUCCESS [  0.387 s]
[INFO] Alfresco Repository AMP Module ...................... SUCCESS
 [ 38.368 s]
[INFO] Alfresco Share AMP Module .......................... SUCCESS [
  0.734 s]
[INFO] Alfresco Repository WAR Aggregator ................. SUCCESS
 [ 11.000 s]
[INFO] Alfresco Solr 4 Configuration ...................... SUCCESS [
  0.028 s]
[INFO] Alfresco Share WAR Aggregator ...................... SUCCESS [
  9.903 s]
[INFO] Alfresco, Share and Solr4 Tomcat Runner ............ SUCCESS [
  0.007 s]
[INFO]
 ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO]
 ------------------------------------------------------------------------
[INFO] Total time: 01:01 min
[INFO] Finished at: 2015-05-06T11:28:02+01:00
```

```
[INFO] Final Memory: 61M/503M
[INFO]
 ------------------------------------------------------------------------

Process finished with exit code 0
```

You should see that the `MAVEN_OPTS` setting has been picked up by IDEA, look at the first line of the log.

You have seen how to import your SDK project into IntelliJ and how to build it.

Rapid Application Development in IntelliJ IDEA (Hot reloading)

Hot reloading is the ability to modify your application's code, and view the changes without having to restart Alfresco Tomcat. This allows for significant savings in development time that would otherwise be wasted restarting Tomcat. Hot reloading is the key to enabling Rapid Application Development (RAD) and Test Driven Development (TDD).

You should have an extension project imported, see importing a project into IDEA.

In this tutorial you will see how changes to your code can be carried out without having to restart Alfresco Tomcat. This tutorial demonstrates hot reloading of JavaScript, FreeMarker template, and Java code. There are three components that work together to enable the best RAD experience:

1. *Spring Loaded*: takes care of hot-reloading any Java class files that we have changed.
2. *Refresh Repository Script*: This is a script that will POST a request to the Alfresco repository web application (that is, `alfresco.war`) telling it to refresh the Repo web script container, so any changes to files related to web scripts will be picked up.
3. *Refresh Share Script*: This is a script that will POST a request to the Alfresco Share web application (that is, `share.war`) telling it to refresh the Surf web script container, so any changes to files related to Surf web scripts will be picked up. This script will also clear the resource dependency caches, so JS changes etc are picked up.

Start an instance of Alfresco Tomcat that will be used for hot-reloading.

1. Use the run script to start the Application server with the Alfresco extension project deployed, for example:

   ```
   alfresco-extensions/all-in-one$ ./run.sh
   ```

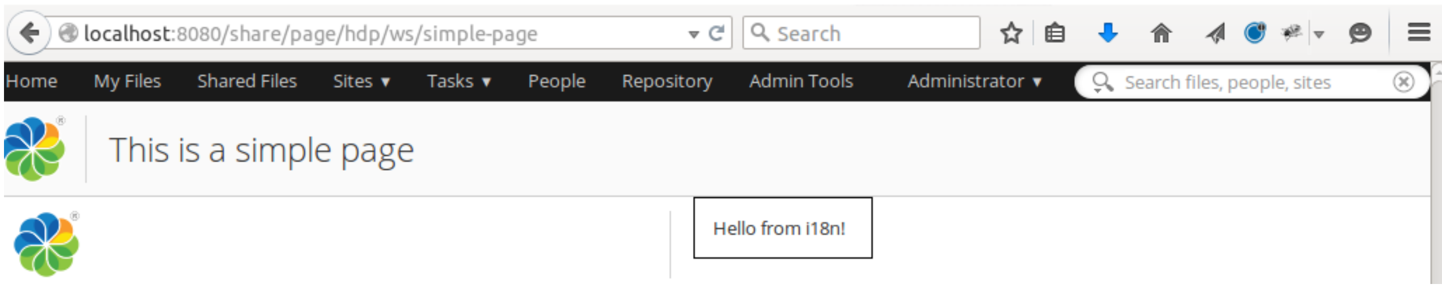
   This is usually done outside the IDE.

2. Test the custom repository web script

   The All-in-One project (and the repository AMP project) have a sample web script included. You can invoke it by pointing your web browser at `http://localhost:8080/alfresco/service/sample/helloworld`. If you need to login then use `admin` with password `admin`. Running this web script produces the output "Message: Hello World from JS! HelloFromJava".

3. Test the custom Share Aikau Page

   The All-in-One project (and the Share AMP project) have a sample Aikau page included. You can display it by pointing your web browser at `http://localhost:8080/share/page/hdp/ws/simple-page`. If you need to login then use `admin` with password `admin`. The page should display as follows:



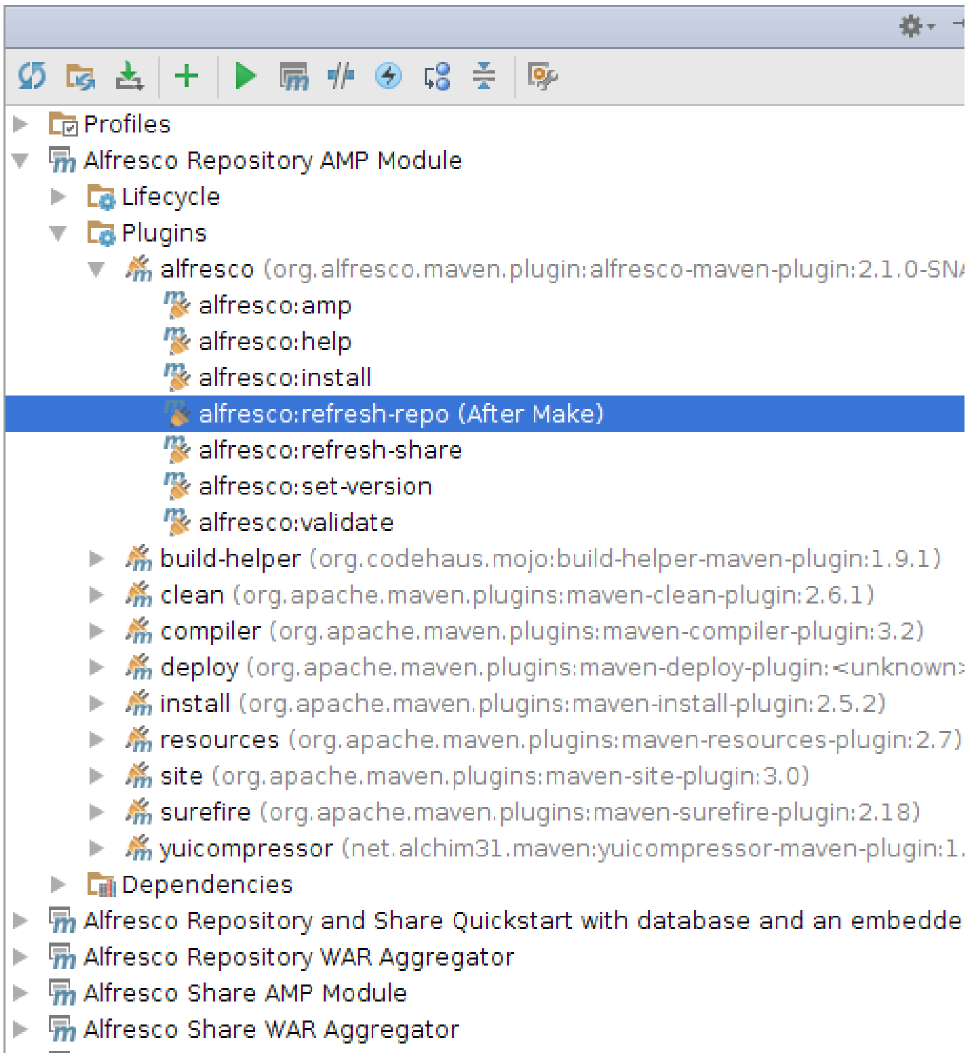
This is a simple page

Hello from i18n!

Enabling Rapid Application Development (RAD) in IDEA.

4. Configure repository AMP projects to Refresh Webapp (`alfresco.war`)

   The All-in-One project has one Repository AMP project by default (all-in-one/repo-amp), we need to set up IDEA so that when we build via IDEA (that is, not via Maven) a script runs that will refresh the Repo Web Script container. In the **Maven Projects** tool view to the right expand the **Alfresco Repository AMP Module** folder, then expand the **Plugins** folder. Now expand the **alfresco** plugin folder. Right click on the **alfresco:refresh-repo** goal of the plugin. In the drop down menu select **Execute After Make**. You should now see a dialog looking something like this:



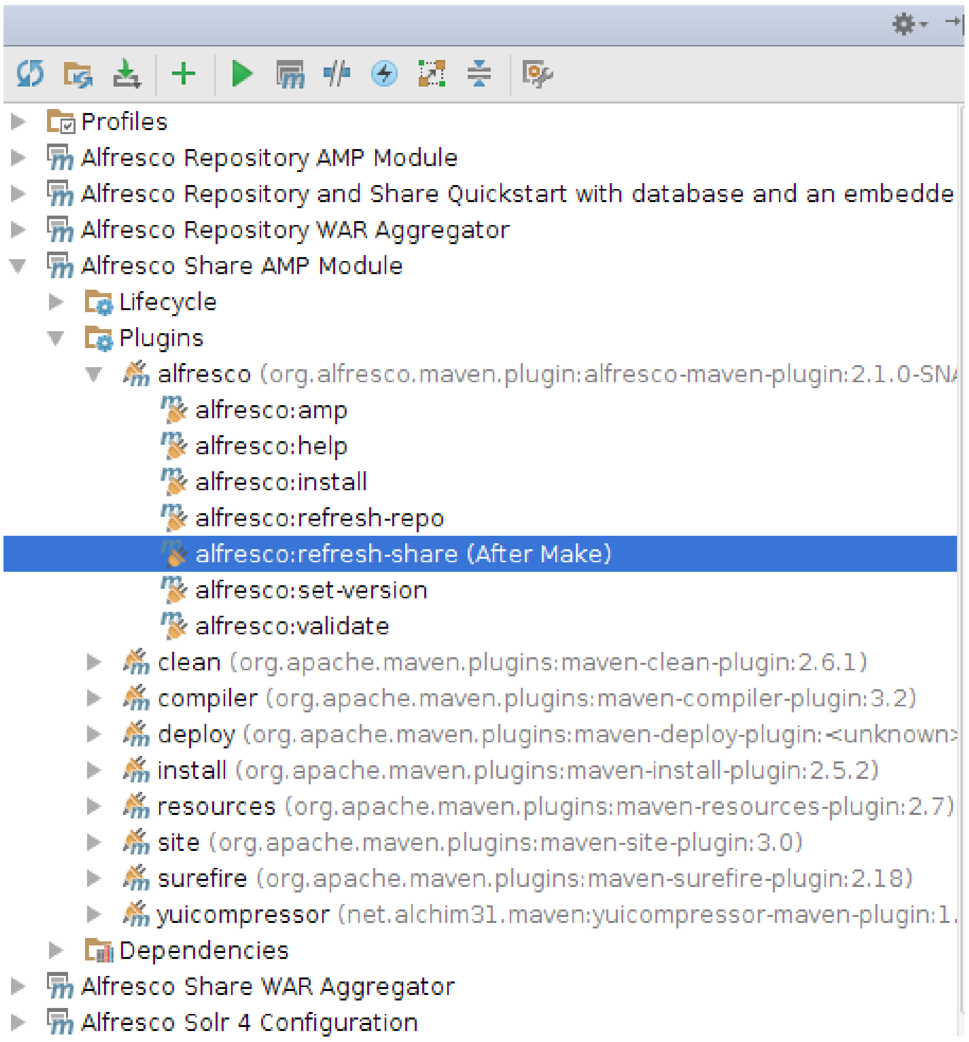
```
▶ Profiles
▼ Alfresco Repository AMP Module
  ▶ Lifecycle
  ▼ Plugins
    ▼ alfresco (org.alfresco.maven.plugin:alfresco-maven-plugin:2.1.0-SN
        alfresco:amp
        alfresco:help
        alfresco:install
        alfresco:refresh-repo (After Make)
        alfresco:refresh-share
        alfresco:set-version
        alfresco:validate
    ▶ build-helper (org.codehaus.mojo:build-helper-maven-plugin:1.9.1)
    ▶ clean (org.apache.maven.plugins:maven-clean-plugin:2.6.1)
    ▶ compiler (org.apache.maven.plugins:maven-compiler-plugin:3.2)
    ▶ deploy (org.apache.maven.plugins:maven-deploy-plugin:<unknown:
    ▶ install (org.apache.maven.plugins:maven-install-plugin:2.5.2)
    ▶ resources (org.apache.maven.plugins:maven-resources-plugin:2.7)
    ▶ site (org.apache.maven.plugins:maven-site-plugin:3.0)
    ▶ surefire (org.apache.maven.plugins:maven-surefire-plugin:2.18)
    ▶ yuicompressor (net.alchim31.maven:yuicompressor-maven-plugin:1.
  ▶ Dependencies
▶ Alfresco Repository and Share Quickstart with database and an embedde
▶ Alfresco Repository WAR Aggregator
▶ Alfresco Share AMP Module
▶ Alfresco Share WAR Aggregator
```

   If you are working with multiple Repo AMPs, then you need to do this configuration for each one of them.

You **only** need to do this if you are doing customizations for the Alfresco repository webapp (that is, customizing `alfresco.war`). If you are just customizing the Share UI, see the next configuration.

5. Configure Share AMP projects to Refresh Webapp (`share.war`)

   The All-in-One project has one Share AMP project by default (all-in-one/share-amp), we need to set up IDEA so that when we build via IDEA (that is, not via Maven) a script runs that will refresh the Surf Web Script container and clear dependency caches. In the **Maven Projects** tool view to the right expand the **Alfresco Share AMP Module** folder, then expand the **Plugins** folder. Now expand the **alfresco** plugin folder. Right click on the **alfresco:refresh-share** goal of the plugin. In the drop down menu select **Execute After Make**. You should now see a dialog looking something like this:



```
▶ Profiles
▶ Alfresco Repository AMP Module
▶ Alfresco Repository and Share Quickstart with database and an embedde
▶ Alfresco Repository WAR Aggregator
▼ Alfresco Share AMP Module
    ▶ Lifecycle
    ▼ Plugins
        ▼ alfresco (org.alfresco.maven.plugin:alfresco-maven-plugin:2.1.0-SN
              alfresco:amp
              alfresco:help
              alfresco:install
              alfresco:refresh-repo
              alfresco:refresh-share (After Make)
              alfresco:set-version
              alfresco:validate
        ▶ clean (org.apache.maven.plugins:maven-clean-plugin:2.6.1)
        ▶ compiler (org.apache.maven.plugins:maven-compiler-plugin:3.2)
        ▶ deploy (org.apache.maven.plugins:maven-deploy-plugin:<unknown>
        ▶ install (org.apache.maven.plugins:maven-install-plugin:2.5.2)
        ▶ resources (org.apache.maven.plugins:maven-resources-plugin:2.7)
        ▶ site (org.apache.maven.plugins:maven-site-plugin:3.0)
        ▶ surefire (org.apache.maven.plugins:maven-surefire-plugin:2.18)
        ▶ yuicompressor (net.alchim31.maven:yuicompressor-maven-plugin:1.
    ▶ Dependencies
▶ Alfresco Share WAR Aggregator
▶ Alfresco Solr 4 Configuration
```

   If you are working with multiple Share AMPs, then you need to do this configuration for each one of them.

   You **only** need to do this if you are doing customizations for the Alfresco Share webapp (that is, customizing `share.war`). If you are just customizing the repository, see the previous configuration.

Testing RAD when doing repository customizations (`alfresco.war`).

6. In the Project Explorer, navigate to and expand the `all-in-one/repo-amp/src/main/amp/config/alfresco/extension/templates/webscripts` folder
7. Locate the `helloworld.get.js` file and load it into the editor by double-clicking it.

This is the controller for the web script that we tried after starting the server. Update the controller code by adding an 'UPDATED' string as follows:

```
model["fromJS"] = "Hello World from JS! UPDATED";
```

8. Now build(Make) the project by clicking **Ctrl+F9**

   The message console in IDEA should display BUILD SUCCESS for both the `repo-amp` and `share-amp` modules as follows:

```
[INFO] Scanning for projects...
[INFO]

[INFO]
 ------------------------------------------------------------------------
[INFO] Building Alfresco Repository AMP Module 1.0-SNAPSHOT
[INFO]
 ------------------------------------------------------------------------
[INFO]
[INFO] --- alfresco-maven-plugin:2.1.0-SNAPSHOT:refresh-repo (default-
cli) @ repo-amp ---
[INFO] Successfull Refresh Web Scripts for Alfresco Repository
[INFO]
 ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO]
 ------------------------------------------------------------------------
[INFO] Total time: 3.854 s
[INFO] Finished at: 2015-05-11T10:20:17+01:00
[INFO] Final Memory: 15M/315M
[INFO]
 ------------------------------------------------------------------------
[INFO] Scanning for projects...
[INFO]

[INFO]
 ------------------------------------------------------------------------
[INFO] Building Alfresco Share AMP Module 1.0-SNAPSHOT
[INFO]
 ------------------------------------------------------------------------
[INFO]
[INFO] --- alfresco-maven-plugin:2.1.0-SNAPSHOT:refresh-share (default-
cli) @ share-amp ---
[INFO] Successfull Refresh Web Scripts for Alfresco Share
[INFO] Successfull Clear Dependency Caches for Alfresco Share
[INFO]
 ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO]
 ------------------------------------------------------------------------
[INFO] Total time: 4.173 s
[INFO] Finished at: 2015-05-11T10:20:24+01:00
[INFO] Final Memory: 18M/309M
[INFO]
 ------------------------------------------------------------------------
[INFO] Maven execution finished
```

   Note the refresh calls to the web applications. If you would prefer to **only** refresh the repository webapp (that is, `alfresco.war`) have a look later on in this article for a different approach to refreshing the webapp via run configurations.

9. Refresh the `http://localhost:8080/alfresco/service/sample/helloworld` Repository Web Script page from the Browser

   The output from the web script should change to "Message: Hello World from JS! UPDATED HelloFromJava". Note that there is no need to restart the application server, just a Make of the project, and a refresh of the web script page from the browser (you are basically invoking the web script again and the update should be immediately visible).
10. Now locate the `helloworld.get.html.ftl` file and double-click it to load it into the editor.

    This is the FreeMarker template for the web script. Update the template by adding an 'ExtraTemplateText' string as follows:

    ```
    Message: ${fromJS} ${fromJava} ExtraTemplateText
    ```
11. Now build(Make) the project by clicking **Ctrl+F9**

    The message console in IDEA should display BUILD SUCCESS
12. Refresh the `http://localhost:8080/alfresco/service/sample/helloworld` Repository Web Script page from the Browser

    The output from the web script should change to "Message: Hello World from JS! UPDATED HelloFromJava ExtraTemplateText ". Again, note that there is no need to restart the application server, only a Make of the project and a refresh of the web script page are necessary.
13. In the next demo we will add a properties file for the web script, create a `helloworld.get.properties` file next to the other files we have been working with.

    The properties file should have one property as follows:

    ```
    hello.word.extras=Extra Stuff From Props
    ```
14. Add this property to the template, open up `helloworld.get.html.ftl`.

    The FreeMarker template should now look like this:

    ```
    Message: ${fromJS} ${fromJava}  ExtraTemplateText
    ${msg("hello.word.extras")}
    ```
15. Now build(Make) the project by clicking **Ctrl+F9**

    The message console in IDEA should display BUILD SUCCESS.
16. Refresh the `http://localhost:8080/alfresco/service/sample/helloworld` Repository Web Script page from the Browser

    The output of the web script should change to "Message: Hello World from JS! UPDATED HelloFromJava ExtraTemplateText Extra Stuff From Props". No restart of application server should be needed, just a Make of the project and a refresh of the web script page from the browser.
17. In the last web script demo we will change the Java controller
18. Open up `all-in-one/repo-amp/src/main/java/{your package path}/demoamp/HelloWorldWebScript.java`.

    Change the property text as follows:

    ```
    public class HelloWorldWebScript extends DeclarativeWebScript {
    ```

```
    protected Map<String, Object> executeImpl(
            WebScriptRequest req, Status status, Cache cache) {
        Map<String, Object> model = new HashMap<String, Object>();
        model.put("fromJava", "HelloFromJavaUPDATED");
        return model;
    }
}
```

19. Now build(Make) the project by clicking **Ctrl+F9**

    The message console in IDEA should display BUILD SUCCESS.
20. Refresh the `http://localhost:8080/alfresco/service/sample/helloworld` Repository Web Script page from the Browser

    The output of the web script should change to "Message: Hello World from JS! UPDATED HelloFromJavaUPDATED ExtraTemplateText Extra Stuff From Props". No restart of application server should be needed, just a Make of the project and a refresh of the web script page from the browser.

Test Driven Development (TDD) and RAD when doing Repository customizations (`alfresco.war`).

21. In the IntelliJ Project Explorer expand `all-in-one/repo-amp/src/test/java/{your package path}/demoamp/test` and then locate the `DemoComponentTest.java` source file.
22. Load it into the editor by double-clicking it.
23. Now set up a Run Configuration to run repo-amp tests. From the main menu select **Run** > **Edit Configurations**.
24. In the **Run/Debug Configurations** dialog, click '+' to create a new configuration. Select **Maven** from the list of available configuration types.
25. Set the **Name** field of the configuration to "Test Repo AMP".
26. Set the **Working Directory** field of the configuration to "alfresco-extensions/all-in-one/repo-amp".
27. Set the **Command line** field "test".
28. Click **OK**.
29. Now run the test by selecting **Run** > **Run 'Test Repo AMP'** from the main menu.

    The test will run, and three tests will pass:

```


/usr/lib/jvm/java-8-oracle/bin/java -Xms256m -Xmx1G -javaagent:/home/
martin/libs/springloaded-1.2.3.RELEASE.jar -noverify -Dmaven.home=/usr/
local/apache-maven-3.2.5 ....
[INFO] Scanning for projects...
[INFO]

[INFO]
 ------------------------------------------------------------------------
[INFO] Building Alfresco Repository AMP Module 1.0-SNAPSHOT
[INFO]
 ------------------------------------------------------------------------
[INFO]
[INFO] --- alfresco-maven-plugin:2.1.0-SNAPSHOT:set-version (default-
set-version) @ repo-amp ---
[INFO] Removed -SNAPSHOT suffix from version - 1.0
[INFO] Added timestamp to version - 1.0.1505120757
[INFO]
[INFO] --- build-helper-maven-plugin:1.9.1:add-test-resource (add-env-
test-properties) @ repo-amp ---
```

```
[INFO]
[INFO] --- maven-resources-plugin:2.7:resources (default-resources) @
 repo-amp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/martin/src/alfresco-
extensions/all-in-one/repo-amp/src/main/resources
[INFO] Copying 14 resources to ../repo-amp
[INFO]
[INFO] --- yuicompressor-maven-plugin:1.5.1:compress (compress-js) @
 repo-amp ---
[INFO] nothing to do, /home/martin/src/alfresco-extensions/all-in-
one/repo-amp/target/classes/../repo-amp/web/scripts/demoamp-min.js is
 younger than original, use 'force' option or clean your target
[INFO] nb warnings: 0, nb errors: 0
[INFO]
[INFO] --- alfresco-maven-plugin:2.1.0-SNAPSHOT:refresh (refresh-
webscripts-repo-and-share) @ repo-amp ---
[INFO] Successfull Refresh Web Scripts for Alfresco Repository
[INFO]
[INFO] --- maven-compiler-plugin:3.2:compile (default-compile) @ repo-
amp ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.7:testResources (default-
testResources) @ repo-amp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 2 resources
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-resources-plugin:2.7:copy-resources (add-module-
properties-to-test-classpath) @ repo-amp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource to alfresco/module/repo-amp
[INFO]
[INFO] --- maven-resources-plugin:2.7:copy-resources (add-module-
config-to-test-classpath) @ repo-amp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 9 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.2:testCompile (default-testCompile)
 @ repo-amp ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.18:test (default-test) @ repo-amp
 ---
[INFO] Surefire report directory: /home/martin/src/alfresco-extensions/
all-in-one/repo-amp/target/surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running org.alfresco.allinone.demoamp.test.DemoComponentTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.085
 sec - in org.alfresco.allinone.demoamp.test.DemoComponentTest

Results :

Tests run: 3, Failures: 0, Errors: 0, Skipped: 0

[INFO]
 -------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO]
 -------------------------------------------------------------------------
[INFO] Total time: 6.946 s
[INFO] Finished at: 2015-05-12T07:57:24+01:00
[INFO] Final Memory: 25M/443M
```

```
[INFO]
 ------------------------------------------------------------------------
```

30. Now, in `DemoComponentTest.java`, modify one of the tests so that it will fail. For example, you could change the line `assertEquals(7, childNodeCount);` to `assertEquals(8, childNodeCount);`
31. Now run the test again by right-clicking the file in Project Explorer and selecting **Run 'Test Repo AMP'**.

    Note the test will run again and this time fail. But you did not need to restart Alfresco. This demonstrates hot reloading of Java code.
32. Change the code back and re-run the test.

    Now you will see that all tests pass. The code has re-run without any reloading of Alfresco! This allows for Test Driven Development with very low overhead.

Testing RAD when doing Share customizations (`share.war`).

33. In the Project Explorer, navigate to and expand the `all-in-one/share-amp/src/main/amp/config/alfresco/web-extension/site-webscripts/com/example/pages` folder
34. Locate the `simple-page.get.js` file and load it into the editor by double-clicking it.

    This is the controller for the Aikau Page web script that we tried after starting the server. Update the controller code by adding an 'UPDATED' string as follows to the page title, also change the layout from HorizontalWidgets to VerticalWidgets:

```
model.jsonModel = {
    widgets: [{
        id: "SET_PAGE_TITLE",
        name: "alfresco/header/SetTitle",
        config: {
            title: "This is a simple page UPDATED"
        }
    },
        {
            id: "MY_HORIZONTAL_WIDGET_LAYOUT",
            name: "alfresco/layout/VerticalWidgets",
            config: {
                widgetWidth: 50,
                widgets: [
                    {
                        id: "DEMO_SIMPLE_LOGO",
                        name: "alfresco/logo/Logo",
                        config: {
                            logoClasses: "alfresco-logo-only"
                        }
                    },
                    {
                        id: "DEMO_SIMPLE_MSG",
                        name: "example/widgets/TemplateWidget"
                    }
                ]
            }
        }]
};
```

35. Now build(Make) the project by clicking **Ctrl+F9**

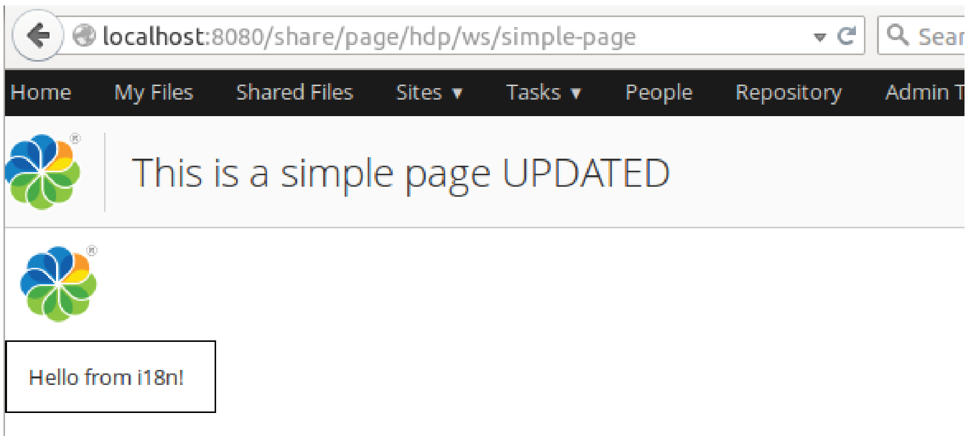
The message console in IDEA should display BUILD SUCCESS for both the `repo-amp` and `share-amp` modules as follows:

```
[INFO] Scanning for projects...
[INFO]

[INFO]
 ------------------------------------------------------------------------
[INFO] Building Alfresco Repository AMP Module 1.0-SNAPSHOT
[INFO]
 ------------------------------------------------------------------------
[INFO]
[INFO] --- alfresco-maven-plugin:2.1.0-SNAPSHOT:refresh-repo (default-
cli) @ repo-amp ---
[INFO] Successfull Refresh Web Scripts for Alfresco Repository
[INFO]
 ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO]
 ------------------------------------------------------------------------
[INFO] Total time: 3.854 s
[INFO] Finished at: 2015-05-11T10:20:17+01:00
[INFO] Final Memory: 15M/315M
[INFO]
 ------------------------------------------------------------------------
[INFO] Scanning for projects...
[INFO]

[INFO]
 ------------------------------------------------------------------------
[INFO] Building Alfresco Share AMP Module 1.0-SNAPSHOT
[INFO]
 ------------------------------------------------------------------------
[INFO]
[INFO] --- alfresco-maven-plugin:2.1.0-SNAPSHOT:refresh-share (default-
cli) @ share-amp ---
[INFO] Successfull Refresh Web Scripts for Alfresco Share
[INFO] Successfull Clear Dependency Caches for Alfresco Share
[INFO]
 ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO]
 ------------------------------------------------------------------------
[INFO] Total time: 4.173 s
[INFO] Finished at: 2015-05-11T10:20:24+01:00
[INFO] Final Memory: 18M/309M
[INFO]
 ------------------------------------------------------------------------
[INFO] Maven execution finished
```

Also, note the refresh calls to the Web Applications to refresh. If you would prefer to **only** refresh the Share webapp (that is, `share.war`) have a look later on in this article for a different approach to refreshing the webapp via run configurations.

36. Refresh the `http://localhost:8080/share/page/hdp/ws/simple-page` Aikau Page from the Browser

    The page should now display as follows:



This is a simple page UPDATED

Hello from i18n!

37. Now, let's update some HTML, CSS, and Properties for the sample widget that is used by the Aikau page, navigate to and expand the `all-in-one/share-amp/src/main/amp/web/js/example/widgets` folder
38. Locate the `css/TemplateWidget.css` file and load it into the editor by double-clicking it.

    This is the Stylesheet for the Aikau Widget. Update the widget style as follows:

```
.my-template-widget {
    border: 2px #000000 solid;
    padding: 1em;
    width: 100px;
    color: white;
    background-color: blue;
}
```

39. Then locate the `i18n/TemplateWidget.properties` file and load it into the editor by double-clicking it.

    This is the resource file for the Aikau Widget. Update the properties as follows:

```
hello-label=Hello from i18n UPDATED!
hello-test=Going to use this label too now!
```

40. Then locate the `templates/TemplateWidget.html` file and load it into the editor by double-clicking it.

    This is the HTML template file for the Aikau Widget. Update so it looks as follows:

```
<div class="my-template-widget">${greeting} and ${greeting2}</div>
```

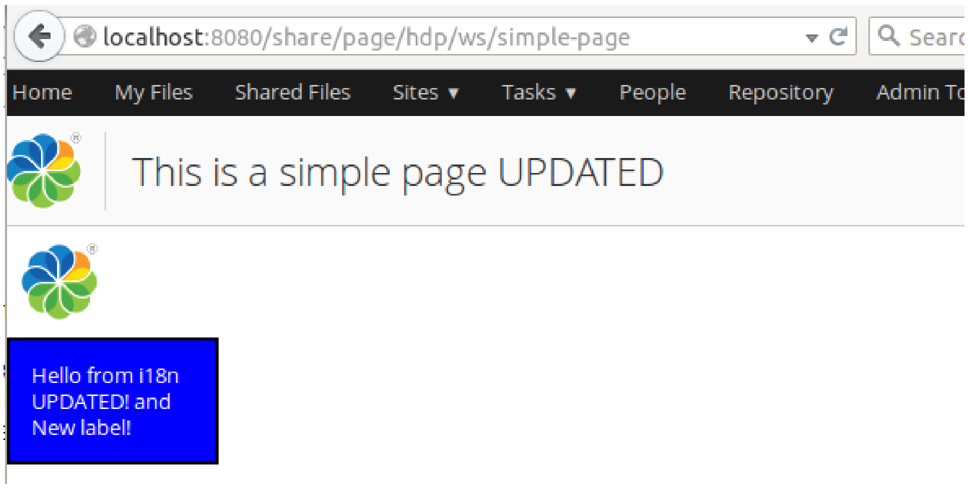
41. And finally, locate the `TemplateWidget.js` file and load it into the editor by double-clicking it.

    This is the main JavaScript implementation for the Aikau Widget. Update so it also sets the new property used in template:

```
define(["dojo/_base/declare",
        "dijit/_WidgetBase",
        "alfresco/core/Core",
        "dijit/_TemplatedMixin",
        "dojo/text!./templates/TemplateWidget.html"
    ],
    function(declare, _Widget, Core, _Templated, template) {
        return declare([_Widget, Core, _Templated], {
            templateString: template,
            i18nRequirements: [ {i18nFile: "./i18n/
TemplateWidget.properties"} ],
            cssRequirements: [{cssFile:"./css/TemplateWidget.css"}],

            buildRendering: function
 example_widgets_TemplateWidget__buildRendering() {
                this.greeting = this.message('hello-label');
                this.greeting2 = this.message('hello-test');

                this.inherited(arguments);

            }
        });
});
```

42. Now build(Make) the project by clicking **Ctrl+F9**.
43. Refresh the `http://localhost:8080/share/page/hdp/ws/simple-page` Aikau Page from the Browser.

    The page should now display as follows:


localhost:8080/share/page/hdp/ws/simple-page
Search
Home
My Files
Shared Files
Sites
Tasks
People
Repository
Admin To
This is a simple page UPDATED
Hello from i18n UPDATED! and New label!

Enabling RAD in IDEA with run configurations.

44. Introduction

    In this article we have seen how we can achieve Rapid Application Development within IDEA by executing alfresco maven plugin refresh goals after a Make. This is an easy way to configure RAD in IDEA when you are only working with 1 or 2 AMPs. However, when you start to get a number of AMPs that you are working on simultaneously, then there will be a lot of refresh calls going on as every AMP's alfresco plugin goal config will be executed. This could be seen in the tutorials above where the `refresh-share` and `refresh-repo` goals were always executed even if we were just working with one of the associated AMPs. In the following tutorial you will see how run configurations can be used for better control of when the refresh call is being made.
45. Set up a Run Configuration to run `repo-amp` builds and Repository webapp refresh (alfresco.war). From the main menu select **Run** > **Edit Configurations**.
46. In the **Run/Debug Configurations** dialog, click '+' to create a new configuration. Select **Maven** from the list of available configuration types.
47. Set the **Name** field of the configuration to "Make Repo AMP".
48. Set the **Working Directory** field of the configuration to "alfresco-extensions/all-in-one/repo-amp".
49. Set the **Command line** field "compile alfresco:refresh-repo".
50. Click **OK**.

51. Now do a Make(build) by selecting **Run** > **Run 'Make Repo AMP'** from the main menu.

    The build will run with the following result:

```
/usr/lib/jvm/java-8-oracle/bin/java -Xms256m -Xmx1G -javaagent:/home/
martin/libs/springloaded-1.2.3.RELEASE.jar -noverify -Dmaven.home=/usr/
local/apache-maven-3.2.5 ....
[INFO] Scanning for projects...
[INFO]

[INFO]
 ------------------------------------------------------------------------
[INFO] Building Alfresco Repository AMP Module 1.0-SNAPSHOT
[INFO]
 ------------------------------------------------------------------------
[INFO]
[INFO] --- alfresco-maven-plugin:2.1.0-SNAPSHOT:set-version (default-
set-version) @ repo-amp ---
[INFO] Removed -SNAPSHOT suffix from version - 1.0
[INFO] Added timestamp to version - 1.0.1505120823
[INFO]
[INFO] --- build-helper-maven-plugin:1.9.1:add-test-resource (add-env-
test-properties) @ repo-amp ---
[INFO]
[INFO] --- maven-resources-plugin:2.7:resources (default-resources) @
 repo-amp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/martin/src/alfresco-
extensions/all-in-one/repo-amp/src/main/resources
[INFO] Copying 14 resources to ../repo-amp
[INFO]
[INFO] --- yuicompressor-maven-plugin:1.5.1:compress (compress-js) @
 repo-amp ---
[INFO] nothing to do, /home/martin/src/alfresco-extensions/all-in-
one/repo-amp/target/classes/../repo-amp/web/scripts/demoamp-min.js is
 younger than original, use 'force' option or clean your target
[INFO] nb warnings: 0, nb errors: 0
[INFO]
[INFO] --- alfresco-maven-plugin:2.1.0-SNAPSHOT:refresh (refresh-
webscripts-repo-and-share) @ repo-amp ---
[INFO] Successfull Refresh Web Scripts for Alfresco Repository
[INFO]
[INFO] --- maven-compiler-plugin:3.2:compile (default-compile) @ repo-
amp ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- alfresco-maven-plugin:2.1.0-SNAPSHOT:refresh-repo (default-
cli) @ repo-amp ---
[INFO] Successfull Refresh Web Scripts for Alfresco Repository
[INFO]
 ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO]
 ------------------------------------------------------------------------
[INFO] Total time: 6.389 s
[INFO] Finished at: 2015-05-12T08:23:57+01:00
[INFO] Final Memory: 30M/490M
[INFO]
 ------------------------------------------------------------------------
```

Note that only the `repo-amp` is built and only the repository webapp is refreshed (`alfresco.war`), the Share application is not touched. This demonstrates how you can have better control of the build and refresh when you have a many different

AMPs. You can create a similar run configuration for the `share-amp` project, and for any other AMP project.

In this tutorial you have seen how to add and modify code within IntelliJ IDEA and then see how these changes take effect immediately, without the need to manually restart or refresh any Alfresco web applications.

Remote debugging with an IDE

It is possible to start an application ready for remote debugging by using the Maven `mvnDebug` command.Eclipse or another development environment such as IDEA can then connect to the running application for remote debugging.

You should have an extension project imported, see importing a project into Eclipse.

It is quite common that you would need to debug Java code associated with an Alfresco extension you are developing. For example, if you are developing a Java backed Web Script. The command `mvnDebug` can be used to start the application in remote debugging mode, where it will listen on port 8000, ready for a remote debugger to attach.

1. In this tutorial we assume that we are working with an All-in-One (AIO) project, change into the top directory (`alfresco-extensions/all-in-one`).

Start the AIO project in debug mode.

2. Run `mvnDebug` rather than the `mvn` command:

   ```
   $ mvnDebug clean install -Prun
   Preparing to Execute Maven in Debug Mode
   Listening for transport dt_socket at address: 8000
   ...
   ```

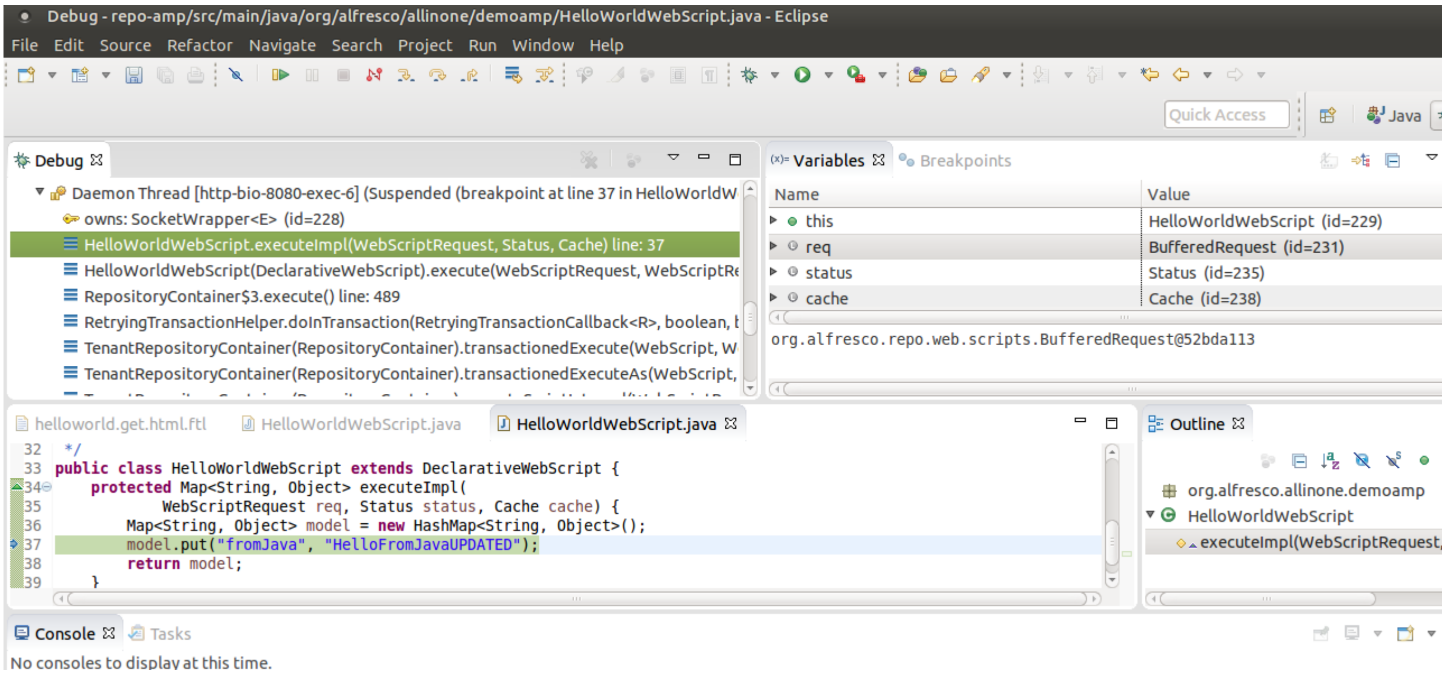
   The Maven project will start and listen for a remote debugger on port 8000.

   This is usually done outside the IDE.

Connect to the running application process from Eclipse

3. In Eclipse, select **Run** > **Debug Configurations** from the main menu.
4. Select **Remote Java Application** and click the **New** icon on the top left of the dialog.
5. Give the `Debug Configuration` a suitable name, such as "Debug Alfresco Extension".
6. Click **Browse** and select the `repo-amp` project, the rest of the fields for host and port have suitable default values.
7. Click **Apply**.
8. Click **Debug** to run the Debug Configuration and connect to the remote Alfresco server.
9. In Eclipse, enable the **Debug** perspective by selecting **Window** > **Open Perspective** > **Debug** from the main menu.
10. Set a breakpoint in the `all-in-one/repo-amp/src/main/java/org/alfresco/allinone/demoamp/HelloWorldWebScript.java`) file.
11. Invoke the `http://localhost:8080/alfresco/service/sample/helloworld` Repository Web Script from the Browser

    You should now see the process stopping in the debugger as follows:


Debug - repo-amp/src/main/java/org/alfresco/allinone/demoamp/HelloWorldWebScript.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Variables
Breakpoints
helloworld.get.html.ftl
HelloWorldWebScript.java
HelloWorldWebScript.java
Outline
Console
Tasks

```
Daemon Thread [http-bio-8080-exec-6] (Suspended (breakpoint at line 37 in HelloWorldW
    owns: SocketWrapper<E> (id=228)
    HelloWorldWebScript.executeImpl(WebScriptRequest, Status, Cache) line: 37
    HelloWorldWebScript(DeclarativeWebScript).execute(WebScriptRequest, WebScriptRe
    RepositoryContainer$3.execute() line: 489
    RetryingTransactionHelper.doInTransaction(RetryingTransactionCallback<R>, boolean, b
    TenantRepositoryContainer(RepositoryContainer).transactionedExecute(WebScript, W
    TenantRepositoryContainer(RepositoryContainer).transactionedExecuteAs(WebScript,
```

Name	Value
this	HelloWorldWebScript (id=229)
req	BufferedRequest (id=231)
status	Status (id=235)
cache	Cache (id=238)

```
org.alfresco.repo.web.scripts.BufferedRequest@52bda113
```

```
32   */
33 public class HelloWorldWebScript extends DeclarativeWebScript {
34     protected Map<String, Object> executeImpl(
35             WebScriptRequest req, Status status, Cache cache) {
36         Map<String, Object> model = new HashMap<String, Object>();
37         model.put("fromJava", "HelloFromJavaUPDATED");
38         return model;
39     }
```

```
org.alfresco.allinone.demoamp
HelloWorldWebScript
    executeImpl(WebScriptRequest,
```

```
No consoles to display at this time.
```

Advanced Topics

This information provides more advanced topics that you might come in contact with when you have been working with an SDK project for a while. We will have a look at how you can add more custom modules to an All-in-One project, how to bring in standard Alfresco modules such as Records Management (RM) and SharePoint Protocol (SPP) support, configuring SSL, and more.

Configure SSL between Repository and Solr in an AIO project

The SDK ships with SSL turned off between the Alfresco Repository and the Solr 4 search server. This article explains how to set that up when running an All-in-One (AIO) project.

You should have completed Installing and Configuring software and generated an AIO project. You will also need access to an Alfresco 5 installation as we need to copy the keystore and Tomcat users file from it.

You will learn how to setup a secure connection (SSL) between the Alfresco Repository web application (alfresco.war) and the Apache Solr 4 web application (alfresco-solr4.war). This is the normal configuration after you have installed Alfresco with a package installer. In the following instructions `ALFRESCO_INSTALL_DIR` is the directory path to where you installed Alfresco 5 with the package installer (for example /opt/alfresco5). And `AIO_PARENT_DIR` points to where the parent project directory is for the All-in-One (AIO) project (for example /home/martin/src/all-in-one).

1. Stop the embedded Tomcat instance, if it is running.
2. Copy the Repository keystore to the AIO project.

   Execute the following command to copy the keystore into the runner project:

   ```
   {ALFRESCO_INSTALL_DIR}/alf_data$ cp -R keystore/ {AIO_PARENT_DIR}/
   runner/
   ```

   We can now configure the embedded Tomcat instace to use this keystore.
3. Copy Tomcat users definition to the AIO project.

   Execute the following command to copy the tomcat users file into the runner project:

   ```
   {AIO_PARENT_DIR}/runner/tomcat$ mkdir conf
   ```

```
{ALFRESCO_INSTALL_DIR}/tomcat/conf$ cp tomcat-users.xml
 {AIO_PARENT_DIR}/runner/tomcat/conf/
```

What we do here is first create a directory to hold the tomcat users file. And then we copy the tomcat users file from the Alfresco installation to this new directory in the runner project. This file contains identities for the Repository and Solr applications when setting up SSL connections.

4. Turn on SSL for Repository.

   Open up the `alfresco-global.properties` file located in the `{AIO_PARENT_DIR}/repo/src/main/properties/local` directory. Then update the section about Solr configuration:

```
index.subsystem.name=solr4
dir.keystore={AIO_PARENT_DIR}/runner/keystore
solr.host=localhost
solr.port=8080
solr.port.ssl=8443
#solr.secureComms=none
```

   Note. You have to change {AIO_PARENT_DIR} to whatever the parent directory is for your AIO project.

5. Update the `tomcat7-maven-plugin` with keystore, port, and Tomcat users

   Open up the `pom.xml` file located in the `{AIO_PARENT_DIR}/runner` directory. Then update the plugin configuration as follows:

```
<plugin>
   <groupId>org.apache.tomcat.maven</groupId>
   <artifactId>tomcat7-maven-plugin</artifactId>
   <executions>
       <execution>
           <id>run-wars</id>
           <goals>
               <goal>run</goal>
           </goals>
           <phase>pre-integration-test</phase>
       </execution>
   </executions>
   <configuration>
       <httpsPort>8443</httpsPort>
       <keystoreFile>${project.basedir}/keystore/ssl.keystore</
keystoreFile>
       <keystorePass>kT9X6oe68t</keystorePass>
       <keystoreType>JCEKS</keystoreType>
       <truststoreFile>${project.basedir}/keystore/ssl.truststore</
truststoreFile>
       <truststorePass>kT9X6oe68t</truststorePass>
       <truststoreType>JCEKS</truststoreType>
       <tomcatUsers>${project.basedir}/tomcat/conf/tomcat-users.xml</
tomcatUsers>
```

6. Change Solr 4 configuration package to the one that has SSL enabled.

   Open up the `pom.xml` file located in the `{AIO_PARENT_DIR}/solr-config` directory. Then update the dependency and plugin configuration as follows:

```
<dependencies>
   <dependency>
       <groupId>org.alfresco</groupId>
       <artifactId>alfresco-solr4</artifactId>
       <version>${alfresco.version}</version>
       <classifier>config-ssl</classifier>
       <type>zip</type>
   </dependency>
```

```
</dependencies>

<build>
   <plugins>
       <plugin>
           <groupId>org.apache.maven.plugins</groupId>
           <artifactId>maven-dependency-plugin</artifactId>
           <executions>
               <execution>
                   <id>unpack-alfresco-config</id>
                   <goals>
                       <goal>unpack</goal>
                   </goals>
                   <phase>generate-resources</phase>
                   <configuration>
                       <outputDirectory>${alfresco.solr.home.dir}</
outputDirectory>
                       <artifactItems>
                           <artifactItem>
                               <groupId>org.alfresco</groupId>
                               <artifactId>alfresco-solr4</artifactId>
                               <version>${alfresco.version}</version>
                               <classifier>config-ssl</classifier>
                               <type>zip</type>
                           </artifactItem>
                       </artifactItems>
```

This Solr 4 configuration comes preconfigured with SSL enabled, keystore and truststore, including the keystores themselves.

7. Delete previous "no-ssl" configuration directory.

```
{AIO_PARENT_DIR}/alf_data_dev/solr4$ rm -rf config/
```

This is so the new SSL enabled configuration is downloaded and installed correctly under `alf_data_dev`.

8. Make sure that the Alfresco Repository (alfresco.war) web application is using SSL.

Open up the `pom.xml` file located in the `{AIO_PARENT_DIR}/repo` directory. Then update the `maven-war-plugin` configuration as follows:

```
<plugin>
   <artifactId>maven-war-plugin</artifactId>
   <executions>
       <execution>
           <id>prepare-exploded-war</id>
           <goals>
               <goal>exploded</goal>
           </goals>
           <phase>prepare-package</phase>
       </execution>
       <execution>
           <id>default-war</id>
          <!-- <configuration>
                <webXml>${project.build.directory}/
${project.build.finalName}-nossl/WEB-INF/web.xml</webXml>
           </configuration>-->
       </execution>
   </executions>
```

What we do here is just commenting out the `web.xml` file that we normally use when we don't want to use SSL.

9. Start it up and make sure it works.

You should see something like this in the logs:

```
{AIO_PARENT_DIR}$ mvn clean install -Prun
...
Jun 05, 2015 10:56:33 AM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8080"]
Jun 05, 2015 10:56:33 AM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8443"]
```

Try accessing Share securely: `https://localhost:8443/share`. Make sure search works by adding a text file with a unique word, then search for it. Then access Solr 4 securely: `https://localhost:8443/solr4`.

You have now setup SSL between the Alfresco Repository and the Solr 4 server.

Adding internal and external JARs to a Repository AMP project

This article explains how to add an *external* JAR to a Repository AMP project via a dependency. It also looks at how to extract some AMP code into its own JAR project, what we call an *internal* JAR, and then have the AMP project include it.

This tutorial assumes that you have completed Installing and Configuring software and generated a Repo AMP project. To try out the examples in this article you will need to install a local SMTP server such as Fake SMTP.

Sometimes when you are developing a Repository AMP you need to include external libraries (JARs) that are not part/are not available in the `tomcat/webapps/alfresco/WEBINF/lib` directory. Being able to do this is one of the benefits of AMPs compared to other extension models. The Repository AMP might also start to grow to a size where it would make sense to move some of the functionality over to separate JAR projects, and have the AMP depend on them. This article goes through how to do these things.

Adding an external JAR to a Repository AMP project.

An external JAR is needed when you want to use some library that is not part of the Alfresco Repository (alfresco.war) web application. Meaning it is not present in the `tomcat/webapps/alfresco/WEB-INF/lib` directory. So you should always first scan this directory in an Alfresco Community Edition or Enterprise installation to see if the library is available. If it is available, then you can include it in the `componentX-repo/pom.xml` as a `provided` dependency.

In the following example we will update the Hello World Web Script code (it is part of the Repository AMP source code) so that it can send emails. However, when we select a Mail library to use for this, we forget to check if the library is already available as part of the Alfresco Repository WAR. We will see what happens then and how to fix it:

1. Stop the embedded Tomcat instance, if it is running (i.e. via -Pamp-to-war).
2. Find a library that can be used to send emails from Java and add a dependency for it to the Repo AMP project.

   We know that the Java Mail library can be used for this, so we look up the maven dependency for it and add it to the `componentX-repo/pom.xml`:

```
...
<dependencies>
    <dependency>
       <groupId>${alfresco.groupId}</groupId>
       <artifactId>alfresco-repository</artifactId>
   </dependency>
   <!-- Added a new dependency for Java Mail -->
   <dependency>
       <groupId>com.sun.mail</groupId>
```

```
        <artifactId>javax.mail</artifactId>
        <version>1.5.3</version>
    </dependency>
</dependencies>
...
```

We can now start implementing some code that uses the Java Mail library.

3. Update the Hello World Web Script to send an email.

   The implementation can be done as follows in the Java controller (`componentX-repo/src/main/java/com/acme/componentX/demoamp/HelloWorldWebScript.java`):

```
import javax.mail.*;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;

/**
* A demonstration Java controller for the Hello World Web Script.
*
* @author martin.bergljung@alfresco.com
* @since 2.1.0
*/
public class HelloWorldWebScript extends DeclarativeWebScript {
   protected Map<String, Object> executeImpl(
           WebScriptRequest req, Status status, Cache cache) {
       Map<String, Object> model = new HashMap<String, Object>();
       model.put("fromJava", "HelloFromJava");
       sendEmail();
       return model;
   }

   private void sendEmail() {
       String to = "somebody@example.com";
       String subject = "Test email from Web Script";
       String body = "Hello World!";

       try {
           // Create mail session
           Properties props = new Properties();
           props.put("mail.smtp.host", "<yourhost>");  // localhost
 will not work
           props.put("mail.smtp.port", "2525");        // non-
privileged port
           Session session = Session.getDefaultInstance(props, null);
           session.setDebug(false);

           // Define message
           Message message = new MimeMessage(session);
           String fromAddress = "no-reply@alfresco.com";
           message.setFrom(new InternetAddress(fromAddress));
           message.addRecipient(Message.RecipientType.TO, new
 InternetAddress(to));
           message.setSubject(subject);

           // Create the message part with body text
           BodyPart messageBodyPart = new MimeBodyPart();
           messageBodyPart.setText(body);
           Multipart multipart = new MimeMultipart();
           multipart.addBodyPart(messageBodyPart);

           // Put parts in message
```

```
            message.setContent(multipart);

            // Send the message
            Transport.send(message);

        } catch (MessagingException me) {
            me.printStackTrace();
        }
    }
}
```

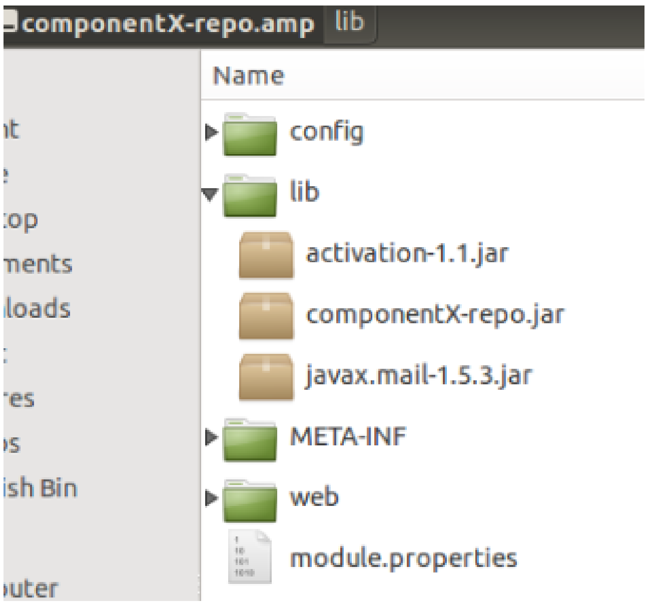
For the code to work you will need to update `<yourhost>` to whatever hostname your machine has. The code uses standard Java Mail classes to send a simple email without attachments.

4. Build the Repository AMP.

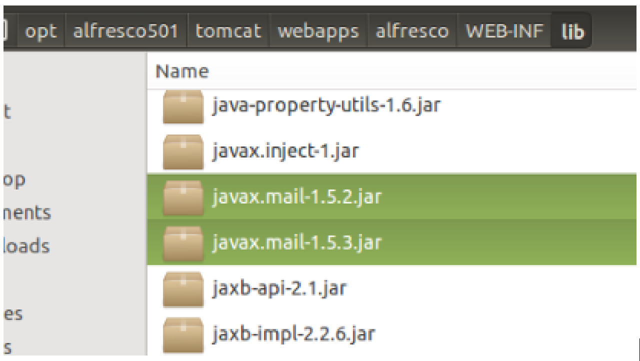
   Standing in the Repo AMP project directory, type the following Maven command:

```
componentX-repo$ mvn clean install -DskipTests=true
```

   What we end up with now is a Repository AMP with the following JARs:



   This might seem okay at first, but it is actually not a good idea. When we apply this AMP to an Alfresco5.WAR file, the JAR files will be added to the `tomcat/webapps/alfresco/WEB-INF/lib` directory, and we end up with duplicate Java Mail libraries as it already contains one version of the library:



So the correct approach is to always check what JARs that are available in the Alfresco 5 WAR, and then for any JAR that is available use `<scope>provided</scope>` when specifying the dependency for it.

5. Update the Java Mail dependency to have scope provided.

   Open up the `componentX-repo/pom.xml` and update the Java Mail dependency as follows:

```
...
<dependencies>
   <dependency>
       <groupId>${alfresco.groupId}</groupId>
       <artifactId>alfresco-repository</artifactId>
   </dependency>

   <dependency>
       <groupId>com.sun.mail</groupId>
       <artifactId>javax.mail</artifactId>
       <version>1.5.3</version>
       <scope>provided</scope>
   </dependency>
...
</dependencies>
```

   This will mean that the Java Mail JAR is not included in the AMP but we can still compile and build against it, which is what we want.

   So to summarize, if the JAR is not in `tomcat/webapps/alfresco/WEB-INF/lib`, then do **not** use `provided` scope as you want it included with the AMP. On the other hand, if the JAR is available, then use `provided` scope to exclude it from the AMP deliverable.

Adding an internal JAR to a Repository AMP project.

Another requirement that might come up when you are working with a Repository AMP project is that it is growing big, so big that it would make sense to move some AMP project code into its own JAR projects, but still have those part of the same AMP build. Basically turning the Repository AMP project into a multi-module project. So let's demonstrate how to do this by moving the Hello World Web Script code into its own JAR module and have the AMP dependent on it.

6. Create a Repository AMP parent project directory.

   To achieve what we want, we will have to turn the Repository AMP project into a multi-module project, and for this we need a parent project. Create a new directory called `componentX-parent` and then copy the Repository AMP directory into it:

```
componentX-repo$ cd ..
~/src/$ mkdir componentX-parent
~/src/$ mv componentX-repo/ componentX-parent/
```

7. Create a Repository AMP parent project POM file.

   Add the following `pom.xml` to the `componentX-parent` directory:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.acme</groupId>
   <artifactId>componentX-parent</artifactId>
   <version>1.0-SNAPSHOT</version>
   <name>componentX-repo Repository AMP parent</name>
```

```
    <description>This is the parent project for the multi-module
 componentX-repo Repository AMP project</description>
    <packaging>pom</packaging>

    <parent>
        <groupId>org.alfresco.maven</groupId>
        <artifactId>alfresco-sdk-parent</artifactId>
        <version>2.2.0</version>
    </parent>

    <modules>
        <module>componentX-repo</module>
    </modules>
</project>
```

Note how this parent project now has the SDK Project as a parent. And it includes the Repository AMP project as a sub-module. The parent project packaging is `pom`, which means it is not producing an artefact like a JAR, AMP, or WAR, it just acts as a parent for other sub-modules.

8. Update parent definition in the Repository AMP project POM file.

   Open up the `componentX-parent/componentX-repo/pom.xml` file. Then update the `parent` definition as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <artifactId>componentX-repo</artifactId>
    <name>componentX-repo Repository AMP project</name>
    <packaging>amp</packaging>
    <description>Manages the lifecycle of the componentX-repo Repository
 AMP (Alfresco Module Package)</description>

    <parent>
        <groupId>com.acme</groupId>
        <artifactId>componentX-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    ...
```

   Note the `parent` section and how it now points to our new parent. Also, we have removed the `version` and `groupId` properties from the AMP project as they are inherited from the parent project.

9. Build the Repository AMP parent project.

   You should see something like this in the logs:

```
componentX-parent$ mvn clean install -DskipTests=true
. . .
[INFO] Reactor Summary:
[INFO]
[INFO] componentX-repo Repository AMP parent .............. SUCCESS [
  0.607 s]
[INFO] componentX-repo Repository AMP project ............. SUCCESS [
  6.850 s]
[INFO]
 ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

So we now got a parent project that can contain more sub-projects then just the Repository AMP project.

10. Add a JAR project and move the Hello World Web Script code.

    We are going to create a new JAR sub-project and then move the Hello World Web Script code over to it from the AMP. Let’s start by adding a new JAR project. Stand in the parent directory and issue the following command, which will generate a starting point for a JAR module:

    ```
    componentX-parent$ mvn archetype:generate -DgroupId=com.acme -
    DartifactId=componentX-web-script -DarchetypeArtifactId=maven-
    archetype-quickstart -DinteractiveMode=false
    ```

    We are using the Maven Quickstart archetype to generate the JAR module. Our parent directory now looks like this:

    ```
    componentX-parent$ ls -l
    total 16
    drwxrwxr-x 6 martin martin 4096 Jun  5 15:27 componentX-repo
    drwxrwxr-x 3 martin martin 4096 Jun  5 15:35 componentX-web-script
    -rw-rw-r-- 1 martin martin  881 Jun  5 15:35 pom.xml
    ```

    When we generate a new module from the parent directory it is automatically added to the `modules` section of the parent POM as follows:

    ```
    ...
    <modules>
           <module>componentX-repo</module>
           <module>componentX-web-script</module>
     </modules>
    </project>
    ```

    And the `parent` section is setup correctly also in the new JAR module, open up `componentX-parent/componentX-web-script/pom.xml` and have a look:

    ```
    <?xml version="1.0"?>
    <project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
    maven.apache.org/xsd/maven-4.0.0.xsd"
            xmlns="http://maven.apache.org/POM/4.0.0"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
       <modelVersion>4.0.0</modelVersion>

       <parent>
           <groupId>com.acme</groupId>
           <artifactId>componentX-parent</artifactId>
           <version>1.0-SNAPSHOT</version>
       </parent>

       <artifactId>componentX-web-script</artifactId>
      <name>componentX-web-script JAR project</name>
       <url>http://maven.apache.org</url>
       ...
    ```

    You might notice that I have removed the `version` and `groupId` properties from the JAR project as they are inherited from our parent project. I also updated the name so it is easy to see what type of artefact that is produced.

11. Build the Repository AMP parent project.

    You should see something like this in the logs:

    ```
    componentX-parent$ mvn clean install -DskipTests=true
    ```

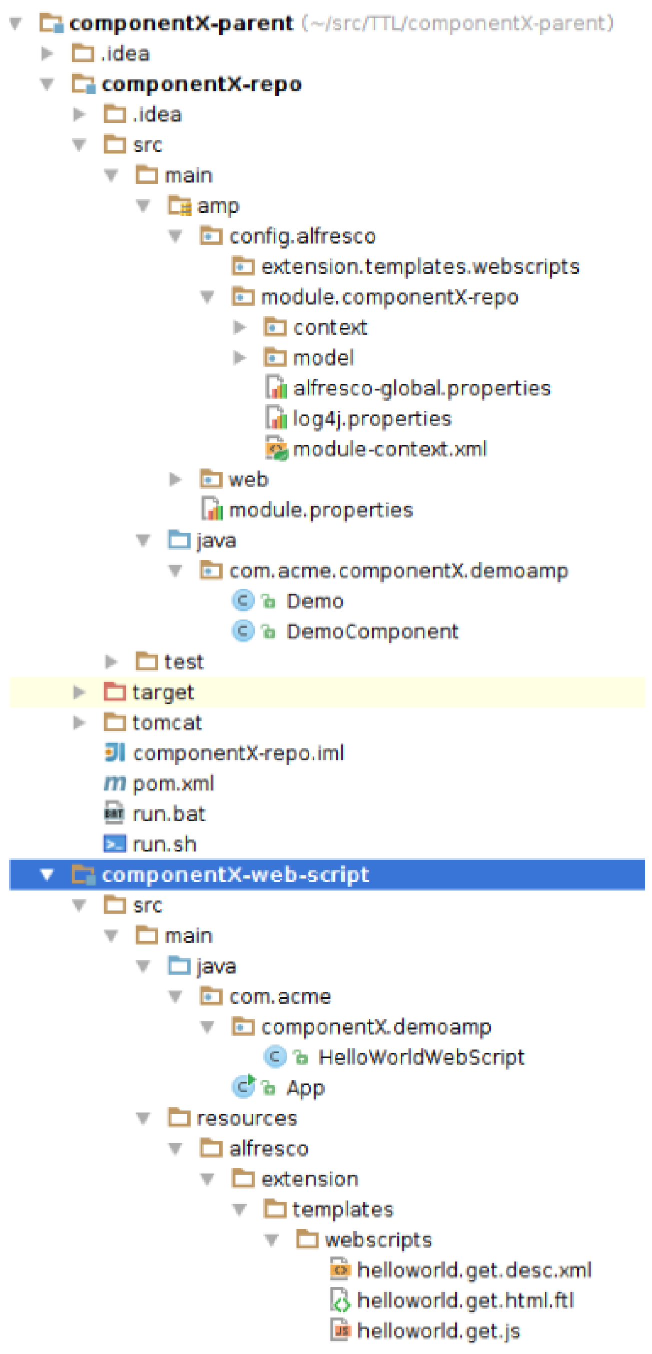
```
. . .
[INFO] Reactor Summary:
[INFO]
[INFO] componentX-repo Repository AMP parent .............. SUCCESS [
  0.685 s]
[INFO] componentX-repo Repository AMP project ............. SUCCESS [
  7.131 s]
[INFO] componentX-web-script JAR project .................. SUCCESS [
  0.539 s]
[INFO]
 ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

12. Move the Hello World Web Script files over to the new JAR project.

    Here we are doing this via command line:

```

componentX-parent$ cd componentX-web-script/
componentX-parent/componentX-web-script$ cd src/main/java/com/acme/
componentX-parent/componentX-web-script/src/main/java/com/acme$ mkdir
 componentX
componentX-parent/componentX-web-script/src/main/java/com/acme$ cd
 componentX/
componentX-parent/componentX-web-script/src/main/java/com/acme/
componentX$ mkdir demoamp
componentX-parent/componentX-web-script/src/main/java/com/acme/
componentX$ cd demoamp/
componentX-parent/componentX-web-script/src/main/java/com/acme/
componentX/demoamp$ mv ~/src/componentX-parent/componentX-repo/src/
main/java/com/acme/componentX/demoamp/HelloWorldWebScript.java .
componentX-parent/componentX-web-script/src/main/java/com/acme/
componentX/demoamp$ cd ../../../../../
componentX-parent/componentX-web-script/src/main$ mkdir resources
componentX-parent/componentX-web-script/src/main$ cd resources/
componentX-parent/componentX-web-script/src/main/resources$ mkdir
 alfresco
componentX-parent/componentX-web-script/src/main/resources$ cd
 alfresco/
componentX-parent/componentX-web-script/src/main/resources/alfresco$
 mkdir extension
componentX-parent/componentX-web-script/src/main/resources/alfresco$ cd
 extension/
componentX-parent/componentX-web-script/src/main/resources/alfresco/
extension$ mkdir templates
componentX-parent/componentX-web-script/src/main/resources/alfresco/
extension$ cd templates/
componentX-parent/componentX-web-script/src/main/resources/alfresco/
extension/templates$ mkdir webscripts
componentX-parent/componentX-web-script/src/main/resources/alfresco/
extension/templates$ cd webscripts/
componentX-parent/componentX-web-script/src/main/resources/alfresco/
extension/templates/webscripts$ mv ~/src/componentX-parent/componentX-
repo/src/main/amp/config/alfresco/extension/templates/webscripts/* .
```

    If you have followed along, then you should see something like this in your IDE now:



```
▼ componentX-parent (~/src/TTL/componentX-parent)
  ▶ .idea
  ▼ componentX-repo
    ▶ .idea
    ▼ src
      ▼ main
        ▼ amp
          ▼ config.alfresco
              extension.templates.webscripts
            ▼ module.componentX-repo
              ▶ context
              ▶ model
                alfresco-global.properties
                log4j.properties
                module-context.xml
          ▶ web
            module.properties
        ▼ java
          ▼ com.acme.componentX.demoamp
              Demo
              DemoComponent
      ▶ test
    ▶ target
    ▶ tomcat
      componentX-repo.iml
      pom.xml
      run.bat
      run.sh
  ▼ componentX-web-script
    ▼ src
      ▼ main
        ▼ java
          ▼ com.acme
            ▼ componentX.demoamp
                HelloWorldWebScript
              App
        ▼ resources
          ▼ alfresco
            ▼ extension
              ▼ templates
                ▼ webscripts
                    helloworld.get.desc.xml
                    helloworld.get.html.ftl
                    helloworld.get.js
```

We are not moving the spring context file with the Java controller bean definition over to the JAR project as we want there only to be one module bringing in the spring context (i.e. the Repository AMP Module). We are just keeping the Web Script files in a separate JAR to be able to work with them in an easier way. It will also give us the opportunity to completely lift out this JAR as a stand-alone project in the future, and release it separately to Nexus, if we should want to do that.

13. Move the Java Mail dependency from the AMP project to the JAR project.

    Open up the `componentX-parent/componentX-web-script/pom.xml` file and update it so it looks like this:

```
<?xml version="1.0"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd"
         xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
   <modelVersion>4.0.0</modelVersion>

   <parent>
       <groupId>com.acme</groupId>
       <artifactId>componentX-parent</artifactId>
       <version>1.0-SNAPSHOT</version>
   </parent>

   <artifactId>componentX-web-script</artifactId>
   <name>componentX-web-script JAR project</name>

   <url>http://maven.apache.org</url>

   <dependencies>
       <dependency>
           <groupId>junit</groupId>
           <artifactId>junit</artifactId>
           <version>3.8.1</version>
           <scope>test</scope>
       </dependency>

       <dependency>
           <groupId>${alfresco.groupId}</groupId>
           <artifactId>alfresco-repository</artifactId>
       </dependency>

       <dependency>
           <groupId>com.sun.mail</groupId>
           <artifactId>javax.mail</artifactId>
           <version>1.5.3</version>
           <scope>provided</scope>
       </dependency>
   </dependencies>
</project>
```

We also needed to add the `alfresco-repository` dependency as the Java Web Script controller uses Declarative Web Script classes. It does not have `version` specified as part of dependency definition, so we need to make sure it works anyway.

14. Move the Dependency Management section from the AMP project to the Parent project.

    Open up the `componentX-parent/pom.xml` file and update it so it looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.acme</groupId>
   <artifactId>componentX-parent</artifactId>
   <version>1.0-SNAPSHOT</version>
   <name>componentX-repo Repository AMP parent</name>
   <description>This is the parent project for the multi-module
 componentX-repo Repository AMP project</description>
   <packaging>pom</packaging>

   <parent>
       <groupId>org.alfresco.maven</groupId>
       <artifactId>alfresco-sdk-parent</artifactId>
       <version>2.2.0</version>
   </parent>

   <modules>
       <module>componentX-repo</module>
       <module>componentX-web-script</module>
   </modules>
   <dependencyManagement>
       <dependencies>
           <dependency>
               <groupId>${alfresco.groupId}</groupId>
               <artifactId>alfresco-platform-distribution</artifactId>
               <version>${alfresco.version}</version>
               <type>pom</type>
               <scope>import</scope>
           </dependency>
       </dependencies>
   </dependencyManagement>
</project>
```

It’s a good idea to move the Dependency Management section to the parent POM as you might have more JAR extensions added that needs Alfresco libraries, and then you want to be consitent with what version of these libraries you are using. The Repository AMP POM now looks like this (i.e. `componentX-parent/componentX-repo/pom.xml`) after removing the Java Mail dependency, moving Dependency Management, and putting in the Web Script JAR dependency:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.acme</groupId>
   <artifactId>componentX-parent</artifactId>
   <version>1.0-SNAPSHOT</version>
   <name>componentX-repo Repository AMP parent</name>
   <description>This is the parent project for the multi-module
 componentX-repo Repository AMP project</description>
   <packaging>pom</packaging>

   <parent>
       <groupId>org.alfresco.maven</groupId>
       <artifactId>alfresco-sdk-parent</artifactId>
       <version>2.2.0</version>
   </parent>

   <modules>
       <module>componentX-repo</module>
       <module>componentX-web-script</module>
   </modules>

   <dependencyManagement>
       <dependencies>
           <dependency>
               <groupId>${alfresco.groupId}</groupId>
               <artifactId>alfresco-platform-distribution</artifactId>
               <version>${alfresco.version}</version>
               <type>pom</type>
               <scope>import</scope>
           </dependency>
       </dependencies>
   </dependencyManagement>
</project>
```

We are now ready to build the project again.

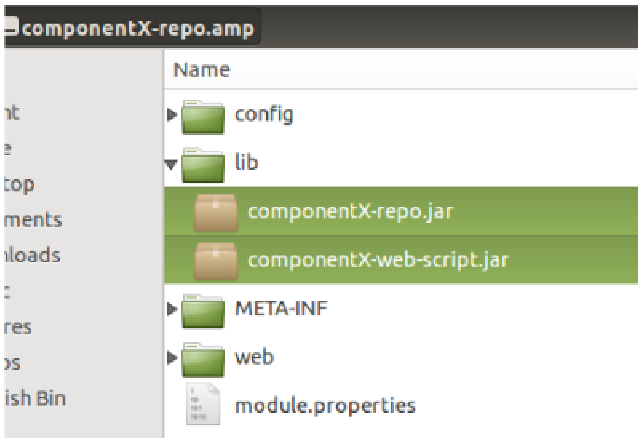
15. Build the Repository AMP parent project.

    You should see something like this in the logs:

```
componentX-parent$ mvn clean install -DskipTests=true
```

```
[INFO] Reactor Summary:
[INFO]
[INFO] componentX-repo Repository AMP parent .............. SUCCESS [
  0.556 s]
[INFO] componentX-web-script JAR project .................. SUCCESS [
  2.662 s]
[INFO] componentX-repo Repository AMP project ............. SUCCESS [
  4.688 s]
```

The Repository AMP now contains the following libraries:



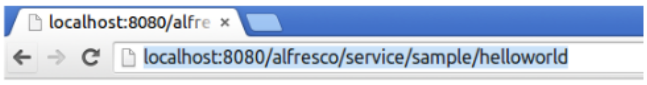
Now it looks a bit better, we have split up the AMP code into two JARs and the Java Mail dependency is set to `provided` so no extra libraries are contained in the produced AMP.

16. Run and try out the Hello World Web Script.

    To make sure this really works we need to try out the Web Script. Step into the AMP directory and do `run.sh`:

```
componentX-parent$ cd componentX-repo/
componentX-parent/componentX-repo$ chmod +x run.sh
componentX-parent/componentX-repo$ ./run.sh
```

You can test the Web Script as follows from a web browser:



localhost:8080/alfresco/service/sample/helloworld

Remember, if you got the send email code in there, then it will try to send an email. And if you don't have an email server (SMTP) running locally, then the Web Script call will eventually time-out as in the following example:

```
2015-06-05 16:43:00,105  INFO
 [alfresco.util.OpenOfficeConnectionTester] [DefaultScheduler_Worker-1]
 The OpenOffice connection was re-established.
 com.sun.mail.util.MailConnectException: Couldn't connect to host,
 port: brutor, 2525; timeout -1;
  nested exception is:
    java.net.ConnectException: Connection timed out
```

We have now seen how external libraries can be used in a Repository AMP project. We have also looked at how to turn the AMP project into a multi-module project to be able to split up the AMP code into multiple JAR extensions.

Linking Standard Alfresco AMPs to an AIO project

Some functionality of the Alfresco content management system is delivered as extra modules, such as Records Management (RM), Google Docs Integration, and Alfresco Office Services, which provides SharePoint Protocol support. You can link such modules to an All-in-One (AIO) project.

You should have completed Installing and Configuring software and generated an AIO project.

You will learn how to link standard Alfresco AMPs to the AIO project so you can use the extra Alfresco functionality that they provide. Most of these modules are implemented with two AMPs. One for server side (Repository) customizations that should be added to the `alfresco.war`, and one with the custom UI functionality that should be added to the `share.war`. **As an example we will add/link the Records Management (RM) module to the AIO project.** It comes implemented in two AMPs. Note that there are different AMPs for Community and Enterprise.

1. Update to latest RM version.

   In the IDE, open up the `all-in-one/pom.xml` parent project file. Scroll down so you see the `properties` section. Uncomment the `alfresco.rm.version` property and set to latest Community release:

   ```
   <properties>
       <!-- The following are default values for data location, Alfresco
    Community version, and Records Management Module version.
            Uncomment if you need to change (Note. current default version
    for Enterprise edition is 5.1)
         <alfresco.version>5.1.e</alfresco.version>
         <alfresco.data.location>/absolute/path/to/alf_data_dev</
   alfresco.data.location> -->

       <alfresco.rm.version>2.4.b</alfresco.rm.version>
   ```

   If you are using the Enterprise edition set the version to `2.4`.

Linking the RM repository AMP to the `alfresco.war`.

2. Add the RM repository AMP dependency.

   In the IDE, open up the `all-in-one/repo/pom.xml` project file. Scroll down so you see the `dependencies` section. Then uncomment the RM dependency and update the `artifactId` to match the Community RM release:

   ```
   <!-- Uncomment if you are using the RM (Records Management) module. -->
       <!-- Set alfresco.rm.version in parent pom to appropriate version
    for 5.1 -->
       <dependency>
           <groupId>${alfresco.groupId}</groupId>
           <artifactId>alfresco-rm-community-repo</artifactId>
           <version>${alfresco.rm.version}</version>
           <type>amp</type>
       </dependency>
   ```

   If you are using the Enterprise edition then you need to define two AMP dependencies, first the `alfresco-rm-core-repo` artifact and then the `alfresco-rm-enterprise-repo` artifact, group and version are the same as for Community.

3. Overlay the RM repository AMP on the `alfresco.war`.

   The RM repository AMP will not be automatically added to the `alfresco.war` by just adding the AMP dependency (JARs will though). We need to add some configuration to the war plugin. Scroll further down in the `all-in-one/repo/pom.xml` project file until you see the `maven-war-plugin` section. Uncomment the RM overlay specification and update `artifactId`:

```
<!-- Uncomment if you are using RM -->
    <overlay>
        <groupId>${alfresco.groupId}</groupId>
        <artifactId>alfresco-rm-community-repo</artifactId>
        <type>amp</type>
    </overlay>
```

If you are using the Enterprise edition then you need to define two AMP overlays, first the `alfresco-rm-core-repo` artifact and then the `alfresco-rm-enterprise-repo` artifact.

Linking the RM Share AMP to the `share.war`.

4. Add the RM Share AMP dependency.

   In the IDE, open up the `all-in-one/share/pom.xml`. Scroll down so you see the `dependencies` section. Then uncomment the RM dependency and update the `artifactId` to match the Community RM release:

```
<!-- Uncomment if you are using RM (Records Management) module -->
    <!-- Make sure to set the correct version for 5.1 with
 alfresco.rm.version property in parent POM -->
    <dependency>
        <groupId>${alfresco.groupId}</groupId>
        <artifactId>alfresco-rm-community-share</artifactId>
        <version>${alfresco.rm.version}</version>
        <type>amp</type>
    </dependency>
```

   If you are using the Enterprise edition then you need to define two AMP dependencies, first the `alfresco-rm-core-share` artifact and then the `alfresco-rm-enterprise-share` artifact, group and version are the same as for Community.

5. Overlay the RM Share AMP on the `share.war`.

   The RM Share AMP will not be automatically added to the `share.war` by just adding the AMP dependency (JARs will though). We need to add some configuration to the war plugin. Scroll further down in the `all-in-one/share/pom.xml` file until you see the `maven-war-plugin` section:

```
<!-- Uncomment if you are using RM -->
     <overlay>
        <groupId>${alfresco.groupId}</groupId>
        <artifactId>alfresco-rm-community-share</artifactId>
        <type>amp</type>
    </overlay>
```

   If you are using the Enterprise edition then you need to define two AMP overlays, first the `alfresco-rm-core-share` artifact and then the `alfresco-rm-enterprise-share` artifact.

Verify that the AIO project has been configured with the RM module.

6. Build and Run the AIO project.

   Use the `all-in-one/run.sh` script to run Alfresco Tomcat with the customized WARs.

7. Check the logs for installation of RM module.

```

... Repository Side:
2016-05-04 07:49:31,752  INFO  [repo.module.ModuleServiceImpl]
 [localhost-startStop-1] Found 3 module package(s).
```

```
2016-05-04 07:49:31,777  INFO  [repo.module.ModuleServiceImpl]
 [localhost-startStop-1] Installing module 'org_alfresco_module_rm'
 version 2.4.
2016-05-04 07:49:32,328  INFO  [repo.module.ModuleServiceImpl]
 [localhost-startStop-1] Installing module 'alfresco-share-services'
 version 5.1.0.
2016-05-04 07:49:32,363  INFO  [repo.module.ModuleServiceImpl]
 [localhost-startStop-1] Installing module 'aio-220-rm-repo-amp'
 version 1.0-SNAPSHOT.
... Share Side:
2016-05-04 07:50:01,595  INFO  [config.packaging.ModulePackageManager]
 [localhost-startStop-1] Found 2 module package(s)
2016-05-04 07:50:01,597  INFO  [config.packaging.ModulePackageManager]
 [localhost-startStop-1] Alfresco Share AMP Module, 1.0-SNAPSHOT
Alfresco Record Management Share Extension, 2.4, Alfresco Record
 Management Share Extension
...
INFO: Starting ProtocolHandler ["http-bio-8080"]
```

We can see here that version 2.4 of the RM module has been installed.

8. Check that the Site Type `Records Management` is available.

   Login to Share via http://localhost:8080/share and then create a new site. When you create the site select `Records Management Site` from the **Type** drop down. If this type is not available then something is not configured correctly, go back and verify that you have followed all the steps correctly.

You have now seen how a standard Alfresco extension module, such as RM, can be brought into the All-in-One project. Other standard modules, such as SPP, can be added in a similar way. Note that some extension modules are implemented in only one AMP. For example, the SPP AMP is implementing the SharePoint Protocol, which only touches the repository functionality, and so there is only an SPP dependency and overlay in the `all-in-one/repo/pom.xml` project.

Adding more custom AMPs to an AIO project

When you generate an All-in-One project you get one Repository extension AMP (repo-amp) and one Share extension AMP (share-amp). These AMPs are just starting point AMPs, showing you how to create extension AMPs for the Alfresco WAR and Share WAR applications. When the project grows you are likely to want to add more extension modules for different types of functionality.

The are many benefits to this:

- It will be easier for multiple developers to work in parallel with different modules/ functionality as it is not all baked into one big AMP.
- You can release and tag modules separately, which is really handy as you are not constantly working with SNAPSHOTS in your main AIO project. Meaning you can decide when you want to bring in new functionality.
- You can very easily do a maintenance release for a specific bit of functionality.
- It encourages re-use by not having all the extension functionality in one big AMP.

There are two ways to go about this though:

1. *Create new custom AMPs as separate projects outside the AIO project*, **this is the recommended approach** as you get all the benefits listed above.
2. *Create new custom AMPs as part of the AIO project*, this does not give all of the above benefits. However, it is useful if you want to split up your extension code and structure it a bit, and you have only a very small team. It is also easier to implement as you don't need access to your own Maven Repository such as Nexus. It does however mean that you will

be working with SNAPSHOT dependencies for all AMPs, so it will be difficult to decide when functionality should be included or not in the build. Basically, if the functionality exist in an AMP in the AIO project, it will be included in the build, whether it is complete or not.

Creating new stand-alone custom AMPs and linking them to the AIO project

Use this information to create a new custom Repository AMP project and a new custom Share AMP project and then link those as dependencies in the AIO project.

You should have completed Installing and Configuring software and generated an AIO project.

You will learn how to create separate AMPs, that are not part of the AIO project, and then how to link those AMPs as dependencies in the AIO project.

If you only need to add a Repository AMP, then skip the steps related to the Share AMP, and vice versa.

Generating a new custom Repository AMP and linking it to the Alfresco.war.

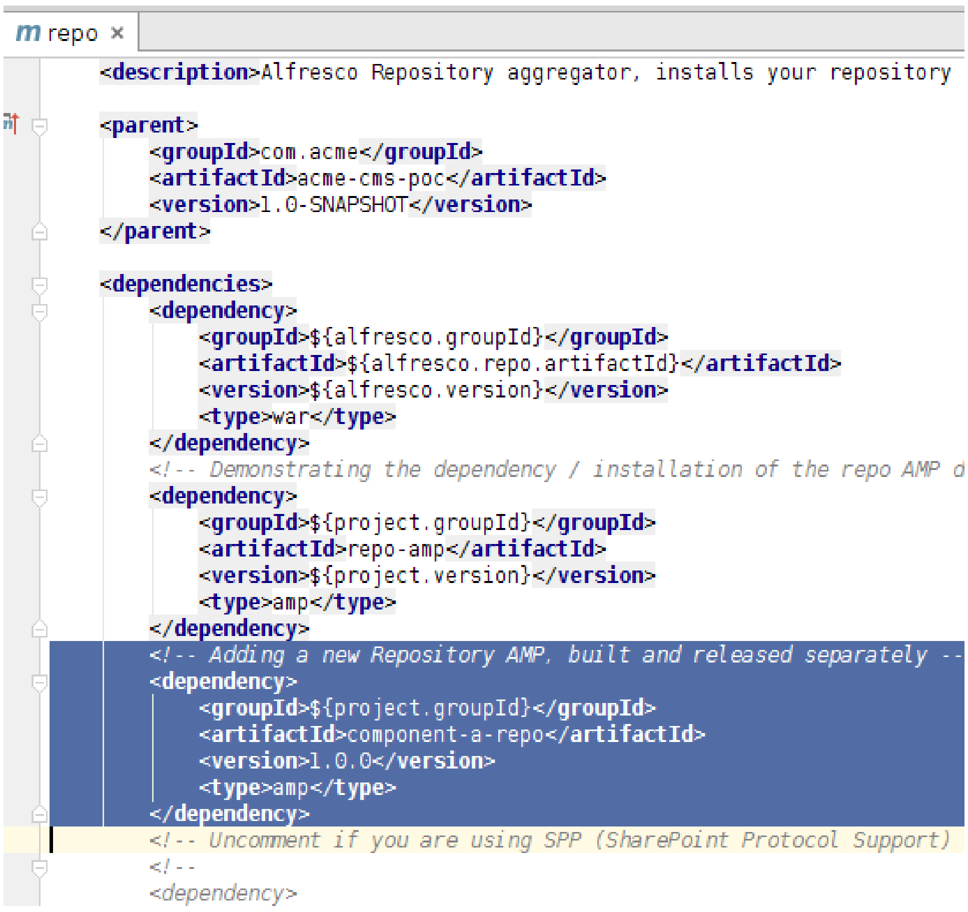
1. Generate the custom Repository AMP.

   Follow instructions in the create Respository extension project (AMP) section. Give the new Repo AMP a unique artifact id that does not clash with any other ids or the one that is part of the AIO project (i.e. repo-amp). For this example I have used the id `component-a-repo`. Use the same group id as for the rest of your project artifacts, I'm using `com.acme`. The AMP is stored in the `alfresco-extensions/component-a-repo` folder and is not part of the AIO build.
2. Build and release version 1.0.0 of the Repository AMP. **(Optional)**

   It is best to avoid SNAPSHOTS when this AMP is included in the All-in-One project. So, use the `maven-release-plugin` and release and tag the AMP so it is ready to be include in the main AIO project.

   Going through how to configure and set up the `maven-release-plugin` is out of scope for this article.
3. Add the custom Repository AMP Dependency

   In the IDE, open up the `alfresco-extensions/acme-cms-poc/repo/pom.xml` project file. Scroll down so you see the `dependencies` section. Then add a dependency to `component-a-repo`:

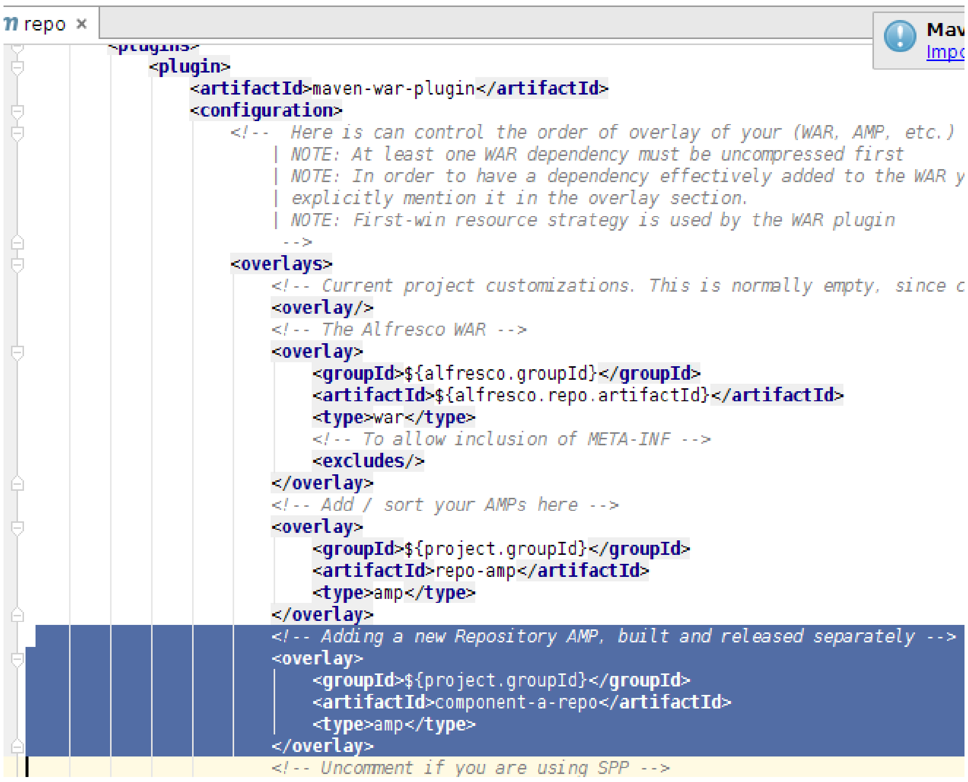


```
m repo ×
    <description>Alfresco Repository aggregator, installs your repository

    <parent>
        <groupId>com.acme</groupId>
        <artifactId>acme-cms-poc</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>${alfresco.groupId}</groupId>
            <artifactId>${alfresco.repo.artifactId}</artifactId>
            <version>${alfresco.version}</version>
            <type>war</type>
        </dependency>
        <!-- Demonstrating the dependency / installation of the repo AMP d
        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>repo-amp</artifactId>
            <version>${project.version}</version>
            <type>amp</type>
        </dependency>
        <!-- Adding a new Repository AMP, built and released separately --
        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>component-a-repo</artifactId>
            <version>1.0.0</version>
            <type>amp</type>
        </dependency>
        <!-- Uncomment if you are using SPP (SharePoint Protocol Support)
        <!--
        <dependency>
```

Note that dependency for the AMP uses the `project.groupId`, which is the same as what we used for the custom AMP, `com.acme`. If you skipped the build and release step (2), then use version 1.0-SNAPSHOT instead.

4. Overlaying the custom Repository AMP on the alfresco.war

   The Repository AMP will not be automatically added to the alfresco.war by just adding the dependency. We need to add some configuration to the war plugin. Scroll further down in the `alfresco-extensions/acme-cms-poc/repo/pom.xml` file until you see the `maven-war-plugin` section. Then add a overlay for the `component-a-repo`:



```
<plugins>
    <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <configuration>
            <!--  Here is can control the order of overlay of your (WAR, AMP, etc.)
                | NOTE: At least one WAR dependency must be uncompressed first
                | NOTE: In order to have a dependency effectively added to the WAR y
                | explicitly mention it in the overlay section.
                | NOTE: First-win resource strategy is used by the WAR plugin
                 -->
            <overlays>
                <!-- Current project customizations. This is normally empty, since c
                <overlay/>
                <!-- The Alfresco WAR -->
                <overlay>
                    <groupId>${alfresco.groupId}</groupId>
                    <artifactId>${alfresco.repo.artifactId}</artifactId>
                    <type>war</type>
                    <!-- To allow inclusion of META-INF -->
                    <excludes/>
                </overlay>
                <!-- Add / sort your AMPs here -->
                <overlay>
                    <groupId>${project.groupId}</groupId>
                    <artifactId>repo-amp</artifactId>
                    <type>amp</type>
                </overlay>
                <!-- Adding a new Repository AMP, built and released separately -->
                <overlay>
                    <groupId>${project.groupId}</groupId>
                    <artifactId>component-a-repo</artifactId>
                    <type>amp</type>
                </overlay>
                <!-- Uncomment if you are using SPP -->
```

5. Run the AIO project and verify that the new repo module is recognized

```
...
2015-05-07 14:18:44,770  INFO  [repo.module.ModuleServiceImpl]
 [localhost-startStop-1] Found 2 module(s).
2015-05-07 14:18:44,791  INFO  [repo.module.ModuleServiceImpl]
 [localhost-startStop-1] Installing module 'component-a-repo' version
 1.0.0.
2015-05-07 14:18:44,808  INFO  [repo.module.ModuleServiceImpl]
 [localhost-startStop-1] Installing module 'repo-amp' version
 1.0.1505071417.
....
```

Generating a new custom Share AMP and adding it to the Share.war.

6. Generate the custom Share AMP.

   Follow instructions in the create Share extension project (AMP) section. Give the new Share AMP a unique artifact id that does not clash with any other ids or the one that is part of the AIO project (i.e. share-amp). For this example I have used the id `component-a-share`. Use the same group id as for the rest of your project artifacts, I'm using `com.acme`. The AMP is stored in the `alfresco-extensions/component-a-share` folder and is not part of the AIO build.

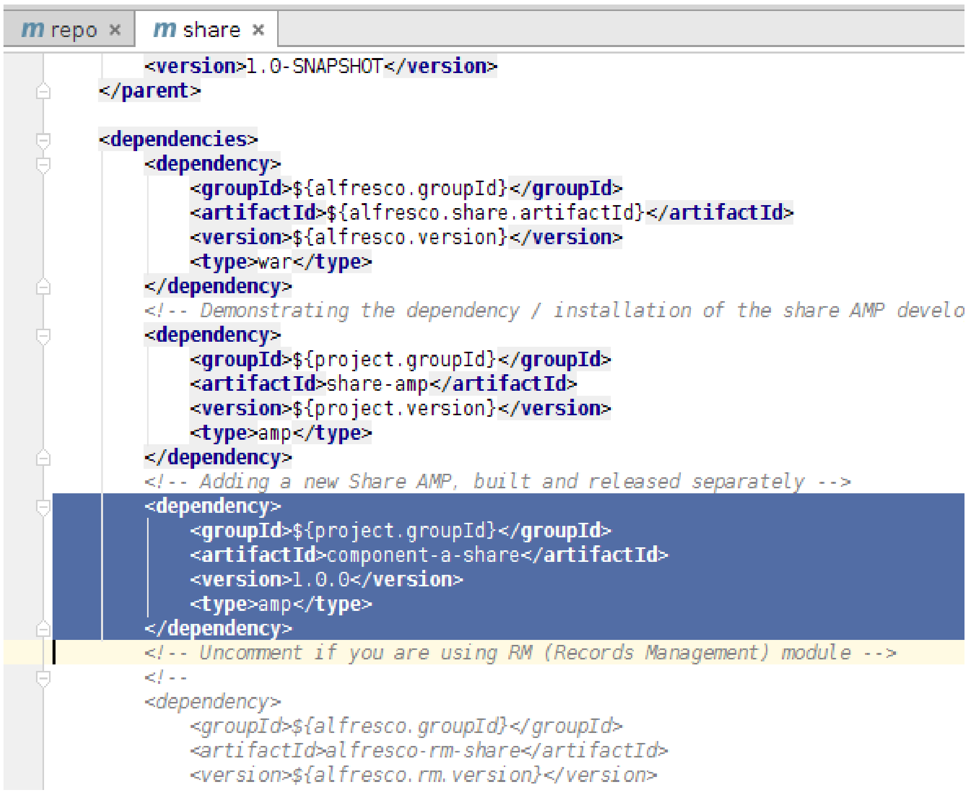
7. Build and release version 1.0.0 of the Share AMP. **(Optional)**

   It is best to avoid SNAPSHOTS when this AMP is included in the All-in-One project. So, use the `maven-release-plugin` and release and tag the AMP so it is ready to be include in the main AIO project.

   Going through how to configure and set up the `maven-release-plugin` is out of scope for this article.

8. Add the custom Share AMP Dependency.

In the IDE, open up the `alfresco-extensions/acme-cms-poc/share/pom.xml` project file. Scroll down so you see the `dependencies` section. Then add a dependency to `component-a-share`:

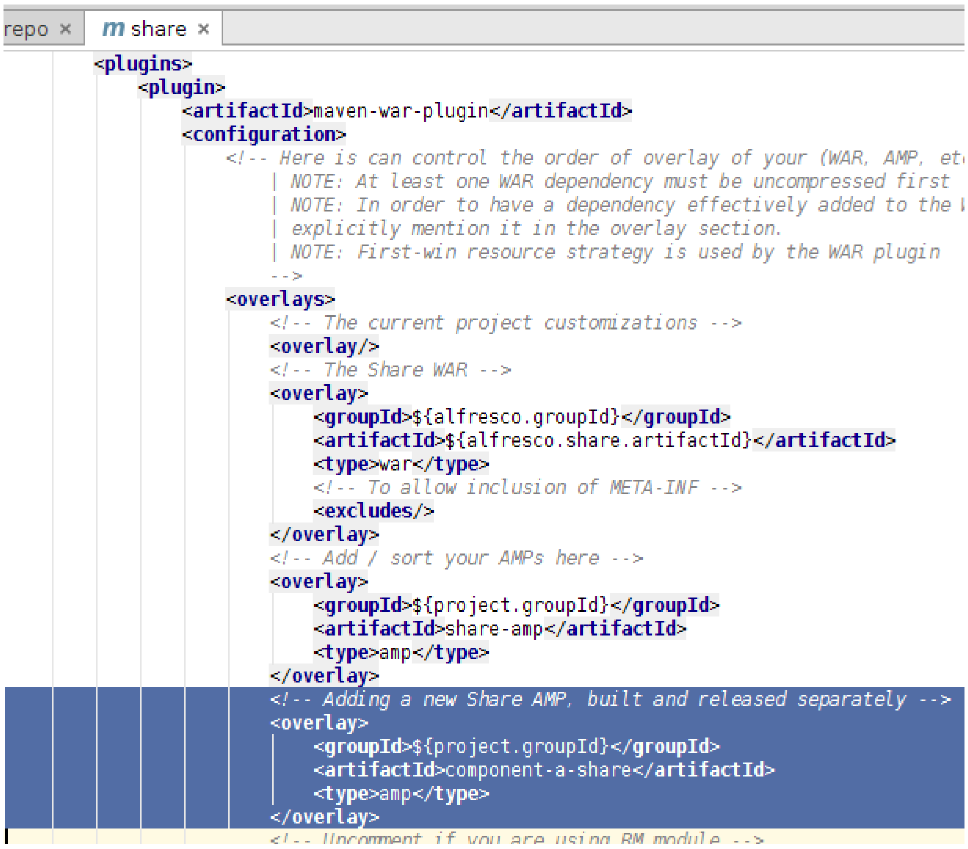


```
        <version>1.0-SNAPSHOT</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>${alfresco.groupId}</groupId>
            <artifactId>${alfresco.share.artifactId}</artifactId>
            <version>${alfresco.version}</version>
            <type>war</type>
        </dependency>
        <!-- Demonstrating the dependency / installation of the share AMP develo
        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>share-amp</artifactId>
            <version>${project.version}</version>
            <type>amp</type>
        </dependency>
        <!-- Adding a new Share AMP, built and released separately -->
        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>component-a-share</artifactId>
            <version>1.0.0</version>
            <type>amp</type>
        </dependency>
        <!-- Uncomment if you are using RM (Records Management) module -->
        <!--
        <dependency>
            <groupId>${alfresco.groupId}</groupId>
            <artifactId>alfresco-rm-share</artifactId>
            <version>${alfresco.rm.version}</version>
```

Note that dependency for the AMP uses the `project.groupId`, which is the same as what we used for the custom AMP, `com.acme`. If you skipped the build and release step (2), then use version 1.0-SNAPSHOT instead.

9. Overlaying the custom Share AMP on the share.war

   The Share AMP will not be automatically added to the share.war by just adding the dependency. We need to add some configuration to the war plugin. Scroll further down in the `alfresco-extensions/acme-cms-poc/share/pom.xml` file until you see the `maven-war-plugin` section. Then add a overlay for the `component-a-share`:



```
<plugins>
    <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <configuration>
            <!-- Here is can control the order of overlay of your (WAR, AMP, et
              | NOTE: At least one WAR dependency must be uncompressed first
              | NOTE: In order to have a dependency effectively added to the 
              | explicitly mention it in the overlay section.
              | NOTE: First-win resource strategy is used by the WAR plugin
              -->
            <overlays>
                <!-- The current project customizations -->
                <overlay/>
                <!-- The Share WAR -->
                <overlay>
                    <groupId>${alfresco.groupId}</groupId>
                    <artifactId>${alfresco.share.artifactId}</artifactId>
                    <type>war</type>
                    <!-- To allow inclusion of META-INF -->
                    <excludes/>
                </overlay>
                <!-- Add / sort your AMPs here -->
                <overlay>
                    <groupId>${project.groupId}</groupId>
                    <artifactId>share-amp</artifactId>
                    <type>amp</type>
                </overlay>
                <!-- Adding a new Share AMP, built and released separately -->
                <overlay>
                    <groupId>${project.groupId}</groupId>
                    <artifactId>component-a-share</artifactId>
                    <type>amp</type>
                </overlay>
                <!-- Uncomment if you are using RM module -->
```

10. There is no logs indicating the AMPs that have been installed on the Alfresco Share web application, so the only way to test is to use whatever custom functionality it is implementing, and see if it works.

You have seen how to build AMP projects separately from the main AIO project, and then how you can incorporate those AMP projects in the main AIO project.

Create new custom AMPs as part of the AIO project

Use this information to create a new custom Repository AMP project and a new custom Share AMP project and have them added as sub-projects of a multi module AIO project.

This tutorial assumes that you completed Installing and Configuring software and generated an AIO project.

You will learn how to create new custom AMPs and have them be part of a multi module AIO project. These AMPs will not be built separately but as part of the AIO project build. And they will be SNAPSHOT references in the AIO project. They will be very similar to the `repo-amp` and `share-amp` demo extension projects that comes with the AIO archetype.

If you only need to add a Repository AMP, then skip the steps related to the Share AMP, and vice versa.

Generating a new custom Repository AMP and linking it to the Alfresco.war.

1. Generate the custom Repository AMP as a sub-module to the AIO parent.

   Follow instructions in the create Respository extension project (AMP) section. Give the new Repo AMP a unique artifact id that does not clash with any other ids or the one that is part of the AIO project (i.e. repo-amp). For this example I have used the id `component-a-repo`. Use the same group id as for the rest of your project artifacts, I'm using `com.acme`. The AMP is generated and stored in the `alfresco-extensions/acme-cms-poc/component-a-repo` folder and is going to **be part of the AIO multi module project**.

2. Update parent definition in the Repository AMP pom.xml and remove group and version definition

   By default the Repository AMP will be generated with the SDK Parent set. We need to change it to be the AIO project parent instead. You can grab a parent definition from one of the other sub-projects, such as the `repo-amp` project. The `alfresco-extensions/acme-cms-poc/component-a-repo/pom.xml` file should now start like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <artifactId>component-a-repo</artifactId>
    <name>component-a-repo Repository AMP project</name>
    <packaging>amp</packaging>
    <description>Manages the lifecycle of the component-a-repo
 Repository AMP (Alfresco Module Package)</description>

    <parent>
        <groupId>com.acme</groupId>
        <artifactId>acme-cms-poc</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    ...
```

   You should also remove <groupId> and <version> as these values will be picked up from the AIO parent.

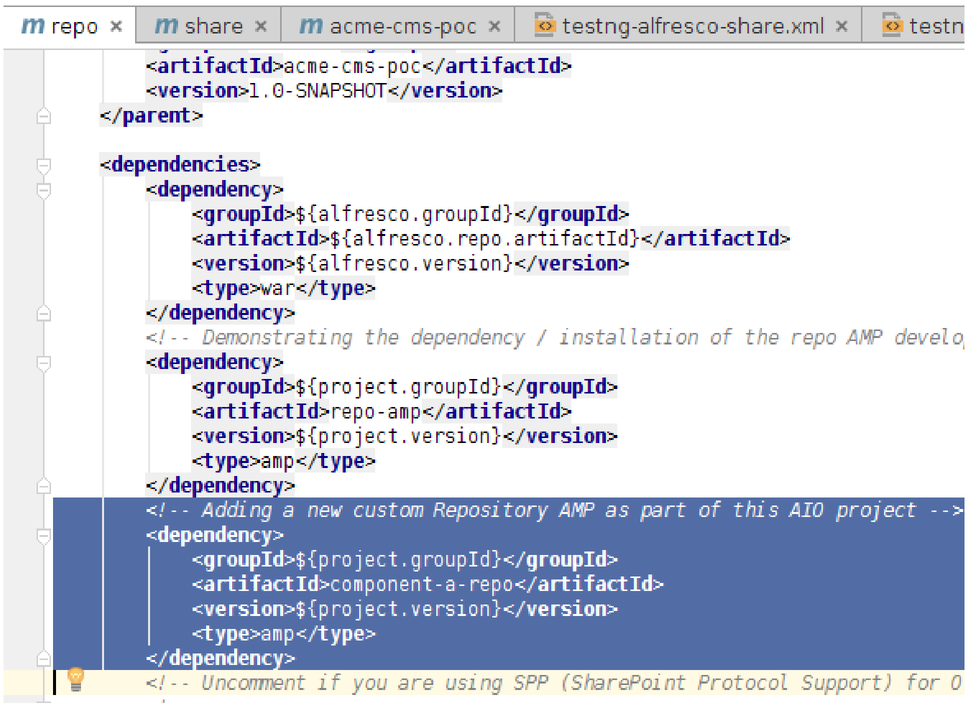
3. Make sure the new Repository AMP is included as a module in the AIO parent pom.xml

   This should happen automatically when you generate the new project in a sub-directory to the parent AIO project directory. Open up the `alfresco-extensions/acme-cms-poc/pom.xml` file and verify that the `component-a-repo` module is there:

```
...
  <modules>
    <module>repo-amp</module>
    <module>share-amp</module>
    <module>repo</module>
    <module>solr-config</module>
    <module>share</module>
    <module>runner</module>
    <module>component-a-repo</module>
  </modules>
</project>
```

4. Add the custom Repository AMP Dependency to Alfresco.war project

   In the IDE, open up the `alfresco-extensions/acme-cms-poc/repo/pom.xml` project file. Scroll down so you see the `dependencies` section. Then add a dependency to `component-a-repo`:

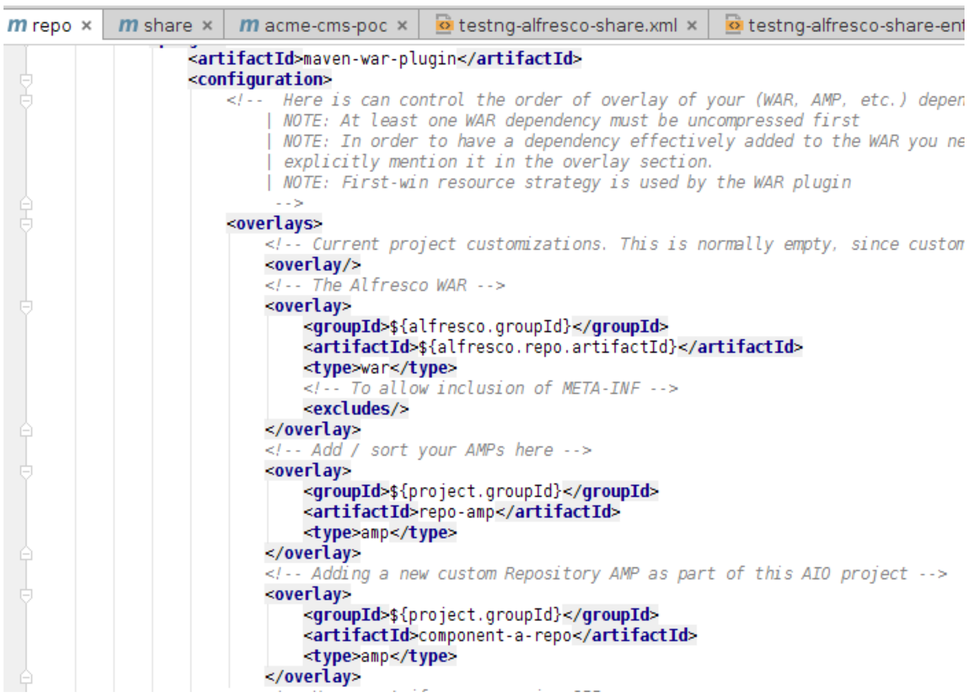


```
m repo ×   m share ×   m acme-cms-poc ×   testng-alfresco-share.xml ×   testn
        <artifactId>acme-cms-poc</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>${alfresco.groupId}</groupId>
            <artifactId>${alfresco.repo.artifactId}</artifactId>
            <version>${alfresco.version}</version>
            <type>war</type>
        </dependency>
        <!-- Demonstrating the dependency / installation of the repo AMP develo
        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>repo-amp</artifactId>
            <version>${project.version}</version>
            <type>amp</type>
        </dependency>
        <!-- Adding a new custom Repository AMP as part of this AIO project -->
        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>component-a-repo</artifactId>
            <version>${project.version}</version>
            <type>amp</type>
        </dependency>
        <!-- Uncomment if you are using SPP (SharePoint Protocol Support) for O
```

Note that the dependency for the AMP uses the `project.groupId`, which is what we used for the custom AMP, `com.acme`. It will also use whatever `project.version` is currently used.

5. Overlaying the custom Repository AMP on the alfresco.war

   The Repository AMP will not be automatically added to the alfresco.war by just adding the dependency. We need to add some configuration to the war plugin. Scroll further down in the `alfresco-extensions/acme-cms-poc/repo/pom.xml` file until you see the `maven-war-plugin` section. Then add a overlay for the `component-a-repo`:



```
m repo ×   m share ×   m acme-cms-poc ×   testng-alfresco-share.xml ×   testng-alfresco-share-ent
                <artifactId>maven-war-plugin</artifactId>
                <configuration>
                    <!--  Here is can control the order of overlay of your (WAR, AMP, etc.) depen
                        | NOTE: At least one WAR dependency must be uncompressed first
                        | NOTE: In order to have a dependency effectively added to the WAR you ne
                        | explicitly mention it in the overlay section.
                        | NOTE: First-win resource strategy is used by the WAR plugin
                         -->
                    <overlays>
                        <!-- Current project customizations. This is normally empty, since custom
                        <overlay/>
                        <!-- The Alfresco WAR -->
                        <overlay>
                            <groupId>${alfresco.groupId}</groupId>
                            <artifactId>${alfresco.repo.artifactId}</artifactId>
                            <type>war</type>
                            <!-- To allow inclusion of META-INF -->
                            <excludes/>
                        </overlay>
                        <!-- Add / sort your AMPs here -->
                        <overlay>
                            <groupId>${project.groupId}</groupId>
                            <artifactId>repo-amp</artifactId>
                            <type>amp</type>
                        </overlay>
                        <!-- Adding a new custom Repository AMP as part of this AIO project -->
                        <overlay>
                            <groupId>${project.groupId}</groupId>
                            <artifactId>component-a-repo</artifactId>
                            <type>amp</type>
                        </overlay>
```

6. Enable the new custom Repository AMP for Rapid Application Development

   To be able to have hot-reloading work for the code that is going to be part of the new `component-a-repo` AMP, we need to update the virtual webapp context for the

Repository webapp (i.e. for the alfresco.war webapp). In the IDE, open up the `alfresco-extensions/acme-cms-poc/runner/tomcat/context-repo.xml` file. Update the `Resource` section configuration with the new AMP's resource path:

```
    <!-- IMPORTANT! The extraResourcePaths string need to be on one
 continues line -->
     <Resources
 className="org.apache.naming.resources.VirtualDirContext"
                 extraResourcePaths="/=${project.parent.basedir}/repo-
amp/target/repo-amp/web,/=${project.parent.basedir}/component-a-repo/
target/component-a-repo/web" />
```

And update the `Loader` section configuration with the new AMP's classpaths:

```

     <Loader className="org.apache.catalina.loader.VirtualWebappLoader"
             searchVirtualFirst="true"
             virtualClasspath="${project.parent.basedir}/repo-amp/
target/classes;
             ${project.parent.basedir}/repo-amp/target/repo-amp/config;
             ${project.parent.basedir}/repo-amp/target/test-classes;
             ${project.parent.basedir}/component-a-repo/target/classes;
             ${project.parent.basedir}/component-a-repo/target/
component-a-repo/config;
             ${project.parent.basedir}/component-a-repo/target/test-
classes" />
```

The Tomcat context file located in the `alfresco-extensions/acme-cms-poc/component-a-repo/tomcat` directory is obsolete when the AMP is contained within an AIO project, it is only used when the AMP is run stand-alone, and it can be deleted.

7. Start it up and verify that the new AMP is installed

```

...
 2015-05-08 13:40:37,688  INFO  [repo.module.ModuleServiceImpl]
 [localhost-startStop-1] Found 2 module(s).
 2015-05-08 13:40:37,713  INFO  [repo.module.ModuleServiceImpl]
 [localhost-startStop-1] Upgrading module 'component-a-repo' version
 1.0.1505081338 (was 1.0.1505071304).
 2015-05-08 13:40:37,746  INFO  [repo.module.ModuleServiceImpl]
 [localhost-startStop-1] Upgrading module 'repo-amp' version
 1.0.1505081338 (was 1.0.1505081106).
...
```

Generating a new custom Share AMP and and linking it to the Share.war.

8. Generate the custom Share AMP as a sub-module to the AIO parent.

   Follow instructions in the create Share extension project (AMP) section. Give the new Share AMP a unique artifact id that does not clash with any other ids or the one that is part of the AIO project (i.e. share-amp). For this example I have used the id `component-a-share`. Use the same group id as for the rest of your project artifacts, I'm using `com.acme`. The AMP is generated and stored in the `alfresco-extensions/acme-cms-poc/component-a-share` folder and is going to **be part of the AIO multi module project**.

9. Update parent definition in the Share AMP pom.xml and remove group and version definition

   By default the Share AMP will be generated with the SDK Parent set. We need to change it to be the AIO project parent instead. You can grab a parent defintion from one of the other sub-projects, such as the `share-amp` project. The `alfresco-extensions/acme-cms-poc/component-a-share/pom.xml` file should now start like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <artifactId>component-a-share</artifactId>
    <name>component-a-share AMP project</name>
    <packaging>amp</packaging>
    <description>Manages the lifecycle of the component-a-share AMP
 (Alfresco Module Package)</description>

    <parent>
        <groupId>com.acme</groupId>
        <artifactId>acme-cms-poc</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    ...
```

You should also remove <groupId> and <version> as these values will be picked up from the AIO parent.

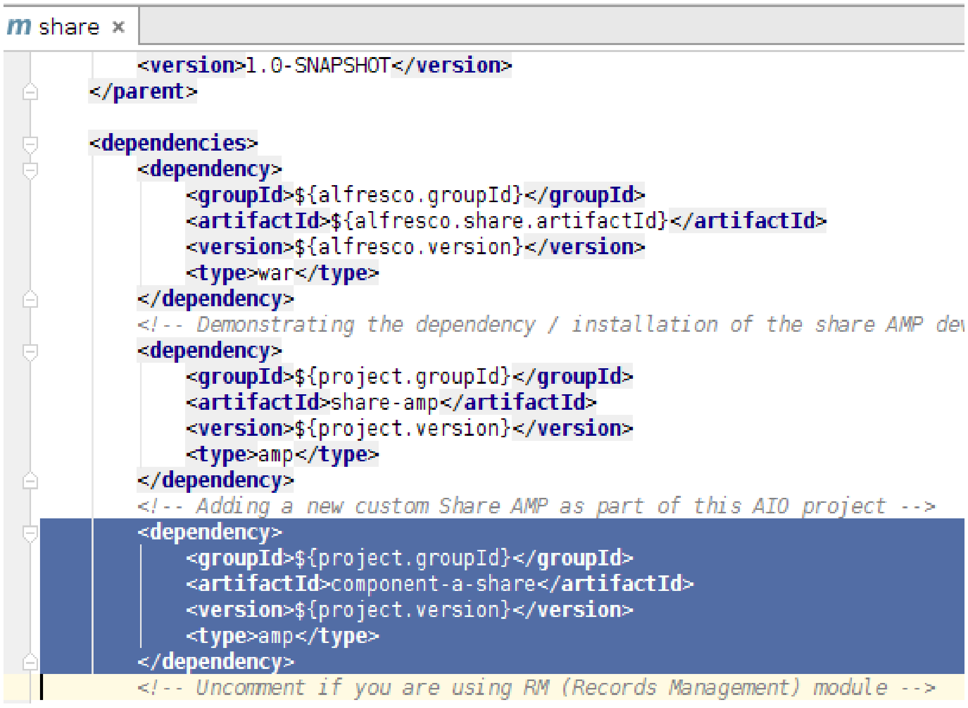
10. Make sure the new Share AMP is included as a module in the AIO parent pom.xml

    This should happen automatically when you generate the new project in a sub-directory to the parent AIO project directory. Open up the `alfresco-extensions/acme-cms-poc/pom.xml` file and verify that the `component-a-share` module is there:

```
...
 <modules>
    <module>repo-amp</module>
    <module>share-amp</module>
    <module>repo</module>
    <module>solr-config</module>
    <module>share</module>
    <module>runner</module>
    <module>component-a-repo</module>
    <module>component-a-share</module>
  </modules>
</project>
```

11. Add the custom Share AMP Dependency.

    In the IDE, open up the `alfresco-extensions/acme-cms-poc/share/pom.xml` project file. Scroll down so you see the `dependencies` section. Then add a dependency to `component-a-share`:

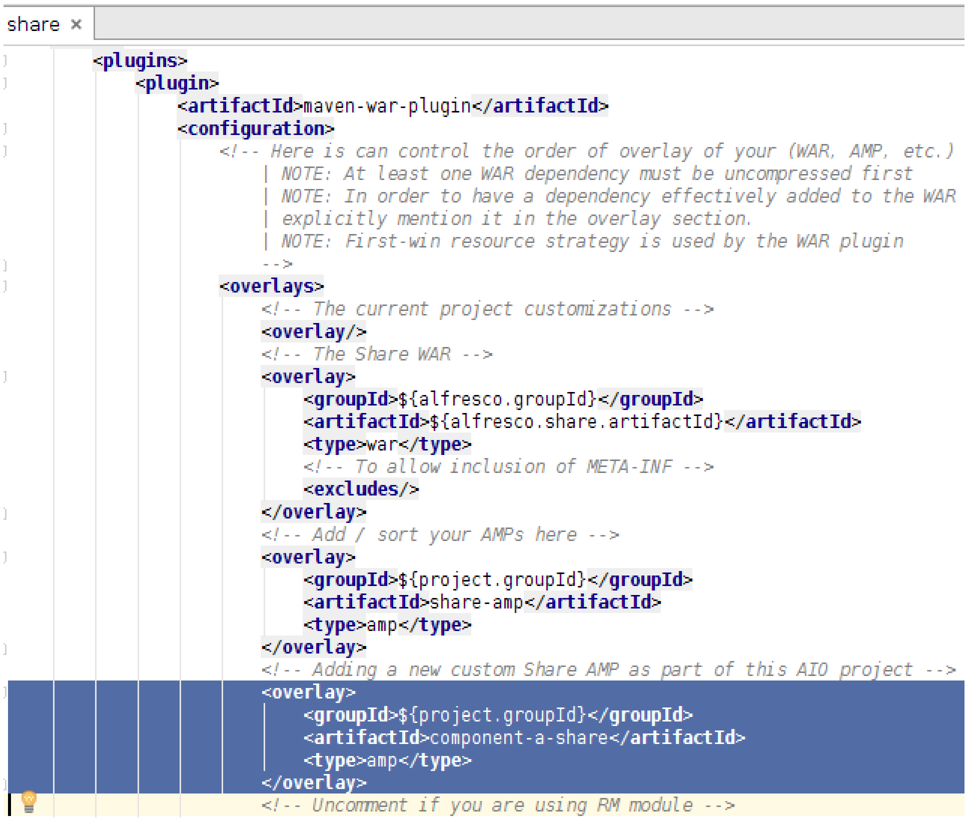


```
m share ×
        <version>1.0-SNAPSHOT</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>${alfresco.groupId}</groupId>
            <artifactId>${alfresco.share.artifactId}</artifactId>
            <version>${alfresco.version}</version>
            <type>war</type>
        </dependency>
        <!-- Demonstrating the dependency / installation of the share AMP dev
        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>share-amp</artifactId>
            <version>${project.version}</version>
            <type>amp</type>
        </dependency>
        <!-- Adding a new custom Share AMP as part of this AIO project -->
        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>component-a-share</artifactId>
            <version>${project.version}</version>
            <type>amp</type>
        </dependency>
        <!-- Uncomment if you are using RM (Records Management) module -->
```

Note that the dependency for the AMP uses the `project.groupId`, which is what we used for the custom AMP, `com.acme`. It will also use whatever `project.version` is currently used.

12. Overlaying the custom Share AMP on the share.war

The Share AMP will not be automatically added to the share.war by just adding the dependency. We need to add some configuration to the war plugin. Scroll further down in the `alfresco-extensions/acme-cms-poc/share/pom.xml` file until you see the `maven-war-plugin` section. Then add a overlay for the `component-a-share`:



```
share ×
        <plugins>
            <plugin>
                <artifactId>maven-war-plugin</artifactId>
                <configuration>
                    <!-- Here is can control the order of overlay of your (WAR, AMP, etc.)
                        | NOTE: At least one WAR dependency must be uncompressed first
                        | NOTE: In order to have a dependency effectively added to the WAR
                        | explicitly mention it in the overlay section.
                        | NOTE: First-win resource strategy is used by the WAR plugin
                        -->
                    <overlays>
                        <!-- The current project customizations -->
                        <overlay/>
                        <!-- The Share WAR -->
                        <overlay>
                            <groupId>${alfresco.groupId}</groupId>
                            <artifactId>${alfresco.share.artifactId}</artifactId>
                            <type>war</type>
                            <!-- To allow inclusion of META-INF -->
                            <excludes/>
                        </overlay>
                        <!-- Add / sort your AMPs here -->
                        <overlay>
                            <groupId>${project.groupId}</groupId>
                            <artifactId>share-amp</artifactId>
                            <type>amp</type>
                        </overlay>
                        <!-- Adding a new custom Share AMP as part of this AIO project -->
                        <overlay>
                            <groupId>${project.groupId}</groupId>
                            <artifactId>component-a-share</artifactId>
                            <type>amp</type>
                        </overlay>
                        <!-- Uncomment if you are using RM module -->
```

13. Enable the new custom Share AMP for Rapid Application Development

    To be able to have hot-reloading work for the code that is going to be part of the new `component-a-share` AMP, we need to update the virtual webapp context for the Share webapp (i.e. for the share.war webapp). Also, if we don't do this any web resources such as CSS, Images, JS etc located under `/web` in the AMP will not be available when we use the `run` profile (this is because during hot reloading we only want them in one place, our project). In the IDE, open up the `alfresco-extensions/acme-cms-poc/runner/tomcat/context-share.xml` file. Update the `Resource` section configuration with the new AMP's resource path:

    ```
        <!-- IMPORTANT! The extraResourcePaths string need to be on one
      continues line -->
         <Resources
      className="org.apache.naming.resources.VirtualDirContext"
                    extraResourcePaths="/=${project.parent.basedir}/share-
     amp/target/share-amp/web,/=${project.parent.basedir}/component-a-share/
     target/component-a-share/web" />
    ```

    And update the `Loader` section configuration with the new AMP's classpaths:

    ```

         <Loader className="org.apache.catalina.loader.VirtualWebappLoader"
                 searchVirtualFirst="true"
                 virtualClasspath="${project.parent.basedir}/share-amp/
     target/classes;
                 ${project.parent.basedir}/share-amp/target/share-amp/
     config;
                 ${project.parent.basedir}/share-amp/target/test-classes;
                 ${project.parent.basedir}/component-a-share/target/classes;
                 ${project.parent.basedir}/component-a-share/target/
     component-a-share/config;
                 ${project.parent.basedir}/component-a-share/target/test-
     classes;
                 ${project.parent.basedir}/share/target/test-classes" />
    ```

    The Tomcat context file located in the `alfresco-extensions/acme-cms-poc/component-a-share/tomcat` directory is obsolete when the AMP is contained within an AIO project, it is only used when the AMP is run stand-alone, and it can be deleted.

14. There is no logs indicating the AMPs that have been installed on the Alfresco Share web application, so the only way to test is to use whatever custom functionality it is implementing, and see if it works.

You have seen how to create new custom AMP projects that should be part of an All-in-One (AIO) project.

Deploying All-in-One (AIO) WARs to external environments

Use this information to deploy the WARs that are produced by the All-In-One (AIO) project to an external environment, such as QA, UAT, and PRODUCTION.

You should have completed Installing and Configuring software and generated an AIO project.

Building an AIO project generates the customized `alfresco.war` and `share.war` for use in an Alfresco installation. However, by default the `alfresco.war` will be configured with an `alfresco-global.properties` file that assumes it will be deployed in a local environment, with the `alf_data` directory in the build project and the use of the flat file H2 database. It is in fact assuming that we will run the environment from Maven (i.e. `mvn clean install -Prun`).

Therefore, if we take the WAR files that are produced during a `mvn clean install` build and deploy them to an external Alfresco installation, it will not work.

This 'local' configuration is managed by a so called environment configuration. The SDK supports multiple environment configurations and this gives us the opportunity to manage configurations for multiple Alfresco environments. This also means that the produced WAR files will contain environment specific configuration, which might not be wanted in all situations. We can prevent this with the use of a specific property during build.

Excluding the environment configuration from the produced Alfresco Repository WAR file.

1. Build with the `app.properties.include` property set to "none".

   The `app.properties.include` is normally set to `**`, which will include all the files under `src/main/properties/${env}` (e.g. `alfresco-global.properties`). If we set this property to `none` it will not match any files. Note that it does not work to set this property to an empty string, and the Maven plug-in that is used does not have a `skip` configuration option:

   ```
   all-in-one$ mvn clean install -Dapp.properties.include=none
   ```

2. Copy the produced WAR files to the Alfresco installation

   When the AIO project has been built without environment specific configuration we can copy the WAR files to an Alfresco installation:

   ```
   all-in-one$ cd repo/target/
   all-in-one/repo/target$ mv repo.war alfresco.war
   all-in-one/repo/target$ cp alfresco.war /opt/alfresco50d/tomcat/
   webapps/
   all-in-one/repo/target$ cd ../../share/target/
   all-in-one/share/target$ cp share.war /opt/alfresco50d/tomcat/webapps/
   all-in-one/share/target$ cd /opt/alfresco50dTest/
   alfresco50d$ cd tomcat/webapps
   alfresco50d/tomcat/webapps$ rm -rf alfresco/ share/
   ```

   Note that we need to remove the exploded WAR directories for the new WARs to be picked up and deployed. Also, the Alfresco Repository WAR is generated with the `repo.war` name so we need to change it to `alfresco.war` before copying it over to the Alfresco installation.

3. Restart Alfresco Tomcat

   The new WARs are now in place so restart Tomcat to have them deployed:

   ```
   alfresco50d$ ./alfresco.sh restart tomcat
   ```

Including environment specific configuration in the produced Alfresco Repository WAR file.

4. Create a new environment directory

   This should be done in the `all-in-one/repo/src/main/properties` directory, which already contains the `local` directory representing the local environment. Name the new directory after the environment you are deploying to, such as for example `uat` (i.e. User Acceptance Testing):

   ```
   all-in-one/repo/src/main/properties$ mkdir uat
   ```

5. Copy the `alfresco-global.properties` file from the external UAT environment to the environment directory.

You will find the environment specific file located in the `alfresco/tomcat/shared/classes` directory from where it can be copied to the build project. At this point you should see something like this under the `repo` project:

```
all-in-one/repo/src/main$ tree
.
### properties
#   ### local
#   #   ### alfresco-global.properties
#   ### uat
#       ### alfresco-global.properties
### resources
    ### alfresco
        ### extension
            ### dev-log4j.properties
```

So we got the `local` environment configuration that will point to a development `alf_data` directory and the H2 database. We then have a new environment configuration called `uat` that contains an `alfresco-global.properties` file that has been copied from that environment's `alfresco/tomcat/shared/classes` directory. Looking in the `uat` environment's properties file we will see something like this (or whatever configuration we have done for the UAT environment):

```
###############################
## Common Alfresco Properties #
###############################

dir.root=/opt/alfresco/alf_data

alfresco.context=alfresco
alfresco.host=127.0.0.1
alfresco.port=8080
alfresco.protocol=http

share.context=share
share.host=127.0.0.1
share.port=8080
share.protocol=http

### database connection properties ###
db.driver=org.postgresql.Driver
db.username=alfresco
db.password=admin
db.name=alfresco
db.url=jdbc:postgresql://localhost:5432/${db.name}
# Note: your database must also be able to accept at least this many
 connections.  Please see your database documentation for instructions
 on how to configure this.
db.pool.max=275
db.pool.validate.query=SELECT 1
...
```

6. Build WARs to include the UAT environment specific configuration.

   We can now activate the UAT environment configuration by specifying the name on the command line (it defaults to `local`):

```
all-in-one$ mvn clean install -DskipTests=true -Denv=uat
```

The `alfresco-global.properties` configuration file for the `uat` environment will end up in the `tomcat/webapps/alfresco/WEB-INF/classes` directory of the final alfresco.war. It will take precedence over the tomcat/shared/classes/alfresco-global.properties file. Note here also that we need to skip Unit tests while doing this build as they require a local context to be running, which is not possible when we change environment configuration.

7. Copy the produced WAR files to the Alfresco installation

   When the AIO project has been built with the UAT environment specific configuration we can copy the WAR files to this Alfresco installation:

   ```
   all-in-one$ cd repo/target/
   all-in-one/repo/target$ mv repo.war alfresco.war
   all-in-one/repo/target$ cp alfresco.war /opt/alfresco50d/tomcat/
   webapps/
   all-in-one/repo/target$ cd ../../share/target/
   all-in-one/share/target$ cp share.war /opt/alfresco50d/tomcat/webapps/
   all-in-one/share/target$ cd /opt/alfresco50dTest/
   alfresco50d$ cd tomcat/webapps
   alfresco50d/tomcat/webapps$ rm -rf alfresco/ share/
   ```

   Note that we need to remove the exploded WAR directories for the new WARs to be picked up and deployed. Also, the Alfresco Repository WAR is generated with the repo.war name so we need to change it to alfresco.war before copying it over to the Alfresco installation.

8. Restart Alfresco Tomcat

   The new WARs are now in place so restart Tomcat to have them deployed:

   ```
   alfresco50d$ ./alfresco.sh restart tomcat
   ```

This article has shown how it is possible to generate WAR files without any environment specific configuration, making them deployable to any environment. We have also covered how to set up a new custom environment configuration, and how to have it included in the final WAR, making it deployable only to a specific Alfresco server.

Upgrading

This information walks through how to upgrade your project to use a newer version of Alfresco. It also takes you through the process of upgrading your project to use a newer version of the SDK.

There are two areas:

- Upgrading the Alfresco Product version
- Upgrading the Alfresco SDK version

Upgrading Alfresco version for an extension project

When you have been working with your extension project for a while it is highly likely that there have been some new releases of the Alfresco software. These releases will have new functionality that you might want to take advantage of in your project. It might also be that you are starting to work with this SDK version a while after it has been released, and latest Alfresco version is now newer than what is default in the SDK. This section will walk through how you can upgrade your SDK project to use the newest Alfresco version.

These instructions include information about how to upgrade projects generated from each one of the artifacts. Make sure you are following upgrade instructions for the correct "From version -> To version".

Make sure you have made a complete backup of your project before you start the upgrade process!

Upgrading SDK 2.1.1 projects from Enterprise 5.0.1 to 5.0.2

These instructions will walk through what is needed when upgrading an SDK 2.1.1 project from using Alfresco Enterprise version 5.0.1 to using Enterprise version 5.0.2.

This task assumes that you have an SDK 2.1.1 project to work with, see creating a project.

You will learn how to set a new version in all the different kinds of SDK project types.

Upgrading Alfresco version for a Repository AMP project.

1. Set new version.

   In the IDE, open up the Repository AMP project that you are working on. Then open the project file for it, for example `alfresco-extensions/component-a-repo/pom.xml`. Scroll down so you see the `properties` section:

   ```
   <properties>
       <!-- The following are default values for data location and
    Alfresco Community version.
            Uncomment if you need to change (Note. current default version
    for Enterprise edition is 5.0.1) -->
         <alfresco.version>5.0.2</alfresco.version>
         <!--<alfresco.data.location>alf_data_dev</alfresco.data.location>
    -->
   ```

   What you need to do here is uncomment the `alfresco.version` property, and then update the version to desired latest version (e.g. 5.0.2). In this case we are upgrading to a newer Enterprise Edition (default is 5.0.1).

2. Clean metadata and content.

   After setting a newer Alfresco version you will need to clean out current database (with metadata), content files, and indexes. It currently does not work to do an incremental upgrade with the SDK and the H2 database. You can clean the DB and content files by running the following command: `alfresco-extensions/component-a-repo/mvn clean -Ppurge`

Upgrading Alfresco version for a Share AMP project.

3. Set new version.

   In the IDE, open up the Share AMP project that you are working on. Then open the project file for it, for example `alfresco-extensions/component-a-share/pom.xml`. Scroll down so you see the `properties` section:

   ```
    <properties>
           <!-- The following are default values for data location and
    Alfresco version.
                Uncomment if you need to change-->
         <alfresco.version>5.0.2</alfresco.version>
   ...
   ```

   What you need to do here is uncomment the `alfresco.version` property, and then update the version to desired latest version (e.g. 5.0.2). In this case we are upgrading to a newer Enterprise Edition (default is 5.0.1).

4. Clean metadata and content

   When upgrading the Alfresco Share AMP it is not necessary to clean out a database or clean content because these are related to the Alfresco Repository application (alfresco.war) and not the Share Application (share.war).

Upgrading Alfresco version for an All-in-One (AIO) project.

5. Set new version.

   In the IDE, open up the All-in-One project that you are working on. Then open the project file for it, for example `alfresco-extensions/all-in-one/pom.xml`. Scroll down so you see the `properties` section:

   ```
   <properties>
       <!-- The following are default values for data location, Alfresco
    Community version, and Records Management Module version.
            Uncomment if you need to change (Note. current default version
    for Enterprise edition is 5.0.1) -->
         <alfresco.version>5.0.2</alfresco.version>
         <alfresco.rm.version>2.3.c</alfresco.rm.version>
   ```

   What you need to do here is uncomment the `alfresco.version` property, and then update the version to desired latest version (e.g. 5.0.2). In this case we are upgrading to a newer Enterprise Edition (default is 5.0.1). Note also that in this case I'm using the Records Management module and I am updating the version for it at the same time by uncommenting the `alfresco.rm.version` property and setting new version (note that for RM to be installed you need to also uncomment the dependencies in the `alfresco-extensions/all-in-one/repo/pom.xml` and `alfresco-extensions/all-in-one/share/pom.xml` WAR projects).

6. Clean metadata and content

   After setting a newer Alfresco version you will need to clean out current database (with metadata), content files, and indexes. It currently does not work to do an incremental upgrade with the SDK and the H2 database. You can clean the DB and content files by running the following command: `alfresco-extensions/all-in-one/mvn clean -Ppurge`

You have now seen how to upgrade the Alfresco Enterprise version to 5.0.2 for the different types of SDK projects and how to clean the database, index, and content before starting with the new version.

Upgrading SDK 2.1.1 projects from Enterprise 5.0.1 (or 5.0.2) to 5.0.3

These instructions will walk through what is needed when upgrading an SDK 2.1.1 project from using Alfresco Enterprise version 5.0.1 (or 5.0.2) to using Enterprise version 5.0.3. There are a couple of changes in the 5.0.3 code base, such as new database scripts, that are not automatically handled by SDK 2.1.1. There is also a problem running with Spring Loaded because of some new security checks in the repository layer.

This task assumes that you have an SDK 2.1.1 project to work with, see creating a project.

You will learn how to set a new version in all the different kinds of SDK project types.

Upgrading Alfresco version for a Repository AMP project.

1. Set new version.

   In the IDE, open up the Repository AMP project that you are working on. Then open the project file for it, for example `alfresco-extensions/component-a-repo/pom.xml`. Scroll down so you see the `properties` section:

   ```
   <properties>
       <!-- The following are default values for data location and
    Alfresco Community version.
            Uncomment if you need to change (Note. current default version
    for Enterprise edition is 5.0.1) -->
   ```

```
        <alfresco.version>5.0.3</alfresco.version>
        <!--<alfresco.data.location>alf_data_dev</alfresco.data.location>
 -->
```

What you need to do here is uncomment the `alfresco.version` property, and then update the version to desired latest version (e.g. 5.0.3). In this case we are upgrading to a newer Enterprise Edition (default is 5.0.1).

2. Add H2 scripts dependency and plug-in repository.

   In the same POM file add the following:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.tomcat.maven</groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <version>${maven.tomcat.version}</version>
            <dependencies>
                <dependency>
                    <groupId>org.alfresco</groupId>
                    <artifactId>alfresco-repository</artifactId>
                    <version>${alfresco.version}</version>
                    <classifier>h2scripts</classifier>
                    <exclusions>
                        <exclusion>
                            <groupId>*</groupId>
                            <artifactId>*</artifactId>
                        </exclusion>
                    </exclusions>
                </dependency>
            </dependencies>
        </plugin>
    </plugins>
</build>

<pluginRepositories>
    <pluginRepository>
        <id>alfresco-private-repository</id>
        <url>https://artifacts.alfresco.com/nexus/content/groups/
private</url>
    </pluginRepository>
</pluginRepositories>
```

   When we run (via `mvn clean install -Pamp-to-war,enterprise`) the Tomcat plug-in is going to need the H2 scripts the first time we start and create the database and the repository. These scripts are usually available in the special SDK dependency `alfresco-rad`, but this file is only updated during new SDK releases. So when a new patch release of Alfresco contains a new script, it is not available in the SDK. Instead we now keep the scripts in this alfresco-repository `h2scripts` dependency. As this dependency is released with each Alfresco version. The `h2scripts` dependency is a plug-in dependency so we need to also set up the private Alfresco Enterprise maven repository as a plug-in repository.

3. Make sure Spring Loaded is not enabled.

   The new security check on the repository side in version 5.0.3 will not work with Spring Loaded (used for hot reloading). Make sure the **run.sh** or **run.bat** scripts are not enabling Spring Loaded. Also check the environment variable `MAVEN_OPTS` so it is not specifying Spring Loaded as Java agent.

4. Clean metadata and content.

   After setting a newer Alfresco version you will need to clean out current database (with metadata), content files, and indexes. It currently does not work to do an incremental upgrade with the SDK and the H2 database. You can clean the DB and content files by running the following command: `alfresco-extensions/component-a-repo/mvn clean -Ppurge`

Upgrading Alfresco version for a Share AMP project.

5. Set new version.

   In the IDE, open up the Share AMP project that you are working on. Then open the project file for it, for example `alfresco-extensions/component-a-share/pom.xml`. Scroll down so you see the `properties` section:

   ```
    <properties>
           <!-- The following are default values for data location and
    Alfresco version.
                Uncomment if you need to change-->
         <alfresco.version>5.0.3</alfresco.version>
   ...
   ```

   What you need to do here is uncomment the `alfresco.version` property, and then update the version to desired latest version (e.g. 5.0.3). In this case we are upgrading to a newer Enterprise Edition (default is 5.0.1).

6. Clean metadata and content

   When upgrading the Alfresco Share AMP it is not necessary to clean out a database or clean content because these are related to the Alfresco Repository application (alfresco.war) and not the Share Application (share.war).

Upgrading Alfresco version for an All-in-One (AIO) project.

7. Set new version.

   In the IDE, open up the All-in-One project that you are working on. Then open the project file for it, for example `alfresco-extensions/all-in-one/pom.xml`. Scroll down so you see the `properties` section:

   ```
   <properties>
       <!-- The following are default values for data location, Alfresco
    Community version, and Records Management Module version.
            Uncomment if you need to change (Note. current default version
    for Enterprise edition is 5.0.1) -->
         <alfresco.version>5.0.3</alfresco.version>
         <alfresco.rm.version>2.3.b</alfresco.rm.version>
   ```

   What you need to do here is uncomment the `alfresco.version` property, and then update the version to desired latest version (e.g. 5.0.3). In this case we are upgrading to a newer Enterprise Edition (default is 5.0.1). Note also that in this case I'm using the Records Management module and I am updating the version for it at the same time by uncommenting the `alfresco.rm.version` property and setting new version.

8. Add H2 scripts dependency and plug-in repository.

   In the Runner project POM file (`alfresco-extensions/all-in-one/runner/pom.xml.`), add the following:

   ```
   . . .
   <pluginRepositories>
       <pluginRepository>
           <id>alfresco-private-repository</id>
           <url>https://artifacts.alfresco.com/nexus/content/groups/
   private</url>
       </pluginRepository>
   </pluginRepositories>
   ```

```
<profiles>
    <profile>
        <id>run</id>
        ...
        <build>
            <plugins>
            ...
                <!-- Run Tomcat 7 embedded with Alfresco.war and
 Share.war contexts.
                     The solr4.war is fetched directly from the Maven
 repo, it is not built like the other WARs.
                     Plugin version is picked up from alfresco-sdk-
parent.pom pluginManagement definition,
                     which also brings in the H2 database lib -->
                <plugin>
                    <groupId>org.apache.tomcat.maven</groupId>
                    <artifactId>tomcat7-maven-plugin</artifactId>
                    <dependencies>
                        <dependency>
                            <groupId>org.alfresco</groupId>
                            <artifactId>alfresco-repository</
artifactId>
                            <version>${alfresco.version}</version>
                            <classifier>h2scripts</classifier>
                            <exclusions>
                                <exclusion>
                                    <groupId>*</groupId>
                                    <artifactId>*</artifactId>
                                </exclusion>
                            </exclusions>
                        </dependency>
                    </dependencies>
                    <executions>
                    ...
```

When we run (via `mvn clean install -Prun,enterprise`) the Tomcat plug-in is going to need the H2 scripts the first time we start and create the database and the repository. These scripts are usually available in the special SDK dependency `alfresco-rad`, but this file is only updated during new SDK releases. So when a new patch release of Alfresco contains a new script, it is not available in the SDK. Instead we now keep the scripts in this alfresco-repository `h2scripts` dependency. As this dependency is released with each Alfresco version. The `h2scripts` dependency is a plug-in dependency so we need to also set up the private Alfresco Enterprise maven repository as a plug-in repository.

9. Make sure Spring Loaded is not enabled.

   The new security check on the repository side in version 5.0.3 will not work with Spring Loaded (used for hot reloading). Make sure the **run.sh** or **run.bat** scripts are not enabling Spring Loaded. Also check the environment variable `MAVEN_OPTS` so it is not specifying Spring Loaded as Java agent.

10. Clean metadata and content

    After setting a newer Alfresco version you will need to clean out current database (with metadata), content files, and indexes. It currently does not work to do an incremental upgrade with the SDK and the H2 database. You can clean the DB and content files by running the following command: `alfresco-extensions/all-in-one/mvn clean -Ppurge`

You have now seen how to upgrade the Alfresco Enterprise version to 5.0.3 for the different types of SDK projects, and how to clean the database, index, and content before starting with the new version.

Upgrading SDK 2.2.0 projects from Community 5.1.e to 5.1.[f|g]

These instructions will walk through what is needed when upgrading an SDK 2.2.0 project from using Alfresco Community version 5.1.e to using Community version 5.1.f. These instructions also apply when upgrading to version 5.1.g.

This task assumes that you have an SDK 2.2.0 project to work with, see creating a project.

You will learn how to set a new version in all the different kinds of SDK project types.

Upgrading Alfresco version for a Repository AMP project.

1. Set new version.

   In the IDE, open up the Repository AMP project that you are working on. Then open the project file for it, for example `alfresco-extensions/component-a-repo/pom.xml`. Scroll down so you see the `properties` section:

   ```
   <properties>
       <!-- The following are default values for data location and
    Alfresco Community version.
            Uncomment if you need to change (Note. current default version
    for Enterprise edition is 5.1) -->
       <alfresco.version>5.1.f</alfresco.version>
       <!--<alfresco.data.location>/absolute/path/to/alf_data_dev</
   alfresco.data.location> -->
   ```

   What you need to do here is uncomment the `alfresco.version` property, and then update the version to desired latest version (e.g. 5.1.f or 5.1.g). In this case we are upgrading to a newer Community Edition (default is 5.1.e).

2. Clean metadata and content.

   After setting a newer Alfresco version you will need to clean out current database (with metadata), content files, and indexes. It currently does not work to do an incremental upgrade with the SDK and the H2 database. You can clean the DB and content files by running the following command: `alfresco-extensions/component-a-repo/mvn clean -Ppurge`

Upgrading Alfresco version for a Share AMP project.

3. Set new version.

   In the IDE, open up the Share AMP project that you are working on. Then open the project file for it, for example `alfresco-extensions/component-a-share/pom.xml`. Scroll down so you see the `properties` section:

   ```
   <properties>
       <!-- The following are default values for data location and
    Alfresco version.
            Uncomment if you need to change -->
       <alfresco.version>5.1.f</alfresco.version>
   ...
   ```

   What you need to do here is uncomment the `alfresco.version` property, and then update the version to desired latest version (e.g. 5.1.f or 5.1.g). In this case we are upgrading to a newer Community Edition (default is 5.1.e).

4. Update spring surf API dependency

   In the same POM file update the Surf dependency so it looks like:

   ```
   <dependency>
       <groupId>org.alfresco.surf</groupId>
       <artifactId>spring-surf-api</artifactId>
       <version>${dependency.surf.version}</version>
       <scope>provided</scope>
   ```

```
</dependency>
```

Surf is no longer a Spring Framework project but instead an Alfresco managed project.

5. Clean metadata and content

   When upgrading the Alfresco Share AMP it is not necessary to clean out a database or clean content because these are related to the Alfresco Repository application (alfresco.war) and not the Share Application (share.war).

6. Run

   When running we need to specify what Surf version that should be used: `mvn clean install -Ddependency.surf.version=6.3 -Pamp-to-war` Alfresco Surf is now released independently from Alfresco Share.

Upgrading Alfresco version for an All-in-One (AIO) project.

7. Set new version.

   In the IDE, open up the All-in-One project that you are working on. Then open the project file for it, for example `alfresco-extensions/all-in-one/pom.xml`. Scroll down so you see the `properties` section:

```
<properties>
    <!-- The following are default values for data location, Alfresco
 Community version, and Records Management Module version.
         Uncomment if you need to change (Note. current default version
 for Enterprise edition is 5.1) -->
    <alfresco.version>5.1.f</alfresco.version>
    <alfresco.rm.version>2.4.b</alfresco.rm.version>
    <!-- <alfresco.data.location>/absolute/path/to/alf_data_dev</
alfresco.data.location> -->
...
```

   What you need to do here is uncomment the `alfresco.version` property, and then update the version to desired latest version (e.g. 5.1.f or 5.1.g). In this case we are upgrading to a newer Community edition (default is 5.1.e). Note also that in this case I'm using the Records Management module and I am updating the version for it at the same time by uncommenting the `alfresco.rm.version` property and setting new version (for information about how to enable RM in an All-In-One project see this article).

8. Update spring surf API dependency in Share AMP

   In the `alfresco-extensions/all-in-one/share-amp/pom.xml` update the dependency it looks like:

```
<dependency>
    <groupId>org.alfresco.surf</groupId>
    <artifactId>spring-surf-api</artifactId>
    <version>${dependency.surf.version}</version>
    <scope>provided</scope>
</dependency>
```

   Surf is no longer a Spring Framework project but instead an Alfresco managed project.

9. Clean metadata and content

   After setting a newer Alfresco version you will need to clean out current database (with metadata), content files, and indexes. It currently does not work to do an incremental upgrade with the SDK and the H2 database. You can clean the DB and content files by running the following command: `alfresco-extensions/all-in-one/mvn clean -Ppurge`

10. Run

When running we need to specify what Surf version that should be used: `mvn clean install -Ddependency.surf.version=6.3 -Prun` Alfresco Surf is now released independently from Alfresco Share.

You have now seen how to upgrade the Alfresco Community version to 5.1.f (or for example 5.1.g) for the different types of SDK projects and how to clean the database, index, and content before starting with the new version.

Upgrading SDK 2.2.0 projects from Community 5.1.e to 5.2.a

These instructions will walk through what is needed when upgrading an SDK 2.2.0 project from using Alfresco Community version 5.1.e to using Community version 5.2.a. In fact, we are upgrading to Alfresco Platform version 5.2.a and Alfresco Share version 5.1.g. Note that from now on the `alfresco.war` is no longer available using the `alfresco` artifactId. Now we have to use `alfresco-platform`.

This task assumes that you have an SDK 2.2.0 project to work with, see creating a project.

You will learn how to set a new version in all the different kinds of SDK project types.

Upgrading Alfresco version for a Repository AMP project.

1. Set new version and artifactId.

   In the IDE, open up the Repository AMP project that you are working on. Then open the project file for it, for example `alfresco-extensions/component-a-repo/pom.xml`. Scroll down so you see the `properties` section:

   ```
   <properties>
       <!-- The following are default values for data location and
    Alfresco Community version.
            Uncomment if you need to change (Note. current default version
    for Enterprise edition is 5.1) -->
        <alfresco.version>5.2.a-EA</alfresco.version>
        <alfresco.repo.artifactId>alfresco-platform</
   alfresco.repo.artifactId> <!-- new name for platform/repository war -->
       <!--<alfresco.data.location>/absolute/path/to/alf_data_dev</
   alfresco.data.location> -->
   ```

   What you need to do here is uncomment the `alfresco.version` property, and then update the version to desired latest version (e.g. 5.2.a-EA). In this case we are upgrading to a newer Platform Community Edition (default is 5.1.e). We also need to tell the project about the new artifactId for the alfresco.war using the `alfresco.repo.artifactId` property.

2. Clean metadata and content.

   After setting a newer Alfresco version you will need to clean out current database (with metadata), content files, and indexes. It currently does not work to do an incremental upgrade with the SDK and the H2 database. You can clean the DB and content files by running the following command: `alfresco-extensions/component-a-repo/mvn clean -Ppurge`

Upgrading Alfresco version for a Share AMP project.

3. Set new version.

   In the IDE, open up the Share AMP project that you are working on. Then open the project file for it, for example `alfresco-extensions/component-a-share/pom.xml`. Scroll down so you see the `properties` section:

   ```
   <properties>
       <!-- The following are default values for data location and
    Alfresco version.
            Uncomment if you need to change -->
   ```

```
    <alfresco.version>5.1.g</alfresco.version>
...
```

What you need to do here is uncomment the `alfresco.version` property, and then update the version to desired latest version (e.g. 5.1.g). In this case we are upgrading to a newer Community Edition (default is 5.1.e). **Note** that the Platform and Share have different version numbers.

4. Update spring surf API dependency

   In the same POM file update the Surf dependency so it looks like:

```
<dependency>
    <groupId>org.alfresco.surf</groupId>
    <artifactId>spring-surf-api</artifactId>
    <version>${dependency.surf.version}</version>
    <scope>provided</scope>
</dependency>
```

   Surf is no longer a Spring Framework project but instead an Alfresco managed project.

5. Clean metadata and content

   When upgrading the Alfresco Share AMP it is not necessary to clean out a database or clean content because these are related to the Alfresco Repository application (alfresco.war) and not the Share Application (share.war).

6. Run

   When running we need to specify what Surf version that should be used: `mvn clean install -Ddependency.surf.version=6.3 -Pamp-to-war` Alfresco Surf is now released independently from Alfresco Share.

Upgrading Alfresco version for an All-in-One (AIO) project.

7. Set new version and artifactId

   In the IDE, open up the All-in-One project that you are working on. Then open the project file for it, for example `alfresco-extensions/all-in-one/pom.xml`. Scroll down so you see the `properties` section:

```
<properties>
    <!-- The following are default values for data location, Alfresco
 Community version, and Records Management Module version.
         Uncomment if you need to change (Note. current default version
 for Enterprise edition is 5.1) -->
      <alfresco.version>5.2.a-EA</alfresco.version>
      <alfresco.repo.artifactId>alfresco-platform</
alfresco.repo.artifactId> <!-- new name for platform/repository war -->
      <share.version>5.1.g</share.version> <!-- new property for
 separate share version -->
      <surf.version>6.3</surf.version>
    <!-- <alfresco.data.location>/absolute/path/to/alf_data_dev</
alfresco.data.location> -->
...
```

   What you need to do here is uncomment the `alfresco.version` property, and then update the version to desired latest version (e.g. 5.2.a-EA). In this case we are upgrading to a newer Platform Community Edition (default is 5.1.e). We also need to tell the project about the new artifactId for the alfresco.war using the `alfresco.repo.artifactId` property. Share now has its own version so setting that with a new property called `share.version`. We also specifically add the surf version here via the `surf.version` property, it will be used when re-defining some of the dependencies and there will be no need to pass surf version on the mvn command line.

Note also that in this case I'm using the Records Management module and I am updating the version for it at the same time by uncommenting the `alfresco.rm.version` property and setting new version (for information about how to enable RM in an All-In-One project see this article).

8. Replace the dependencyManagement section

   In the same AIO parent pom file replace the `dependencyManagement` section so it looks like:

```
<dependencyManagement>
       <dependencies>
           <dependency>
               <groupId>${alfresco.groupId}</groupId>
               <artifactId>alfresco-platform-distribution</artifactId>
               <version>${alfresco.version}</version>
               <type>pom</type>
               <scope>import</scope>
           </dependency>

           <!-- Redefine the following Share dependencies as they have
different version numbers than platform.
               They are defined in alfresco-platform-distribution...
-->
           <dependency>
               <groupId>${alfresco.groupId}</groupId>
               <artifactId>share</artifactId>
               <version>${share.version}</version>
               <type>war</type>
               <scope>provided</scope>
           </dependency>
           <dependency>
               <groupId>${alfresco.groupId}</groupId>
               <artifactId>share</artifactId>
               <version>${share.version}</version>
               <classifier>classes</classifier>
               <scope>provided</scope>
           </dependency>
           <dependency>
               <groupId>${alfresco.groupId}</groupId>
               <artifactId>alfresco-web-framework-commons</artifactId>
               <version>${share.version}</version>
               <classifier>classes</classifier>
               <scope>provided</scope>
           </dependency>

           <!-- Redefine the following surf dependencies as they have
no resolvable version in the
                alfresco-platform-distribution artifact -->
           <dependency>
               <groupId>org.alfresco.surf</groupId>
               <artifactId>spring-surf</artifactId>
               <version>${surf.version}</version>
               <scope>provided</scope>
           </dependency>
           <dependency>
               <groupId>org.alfresco.surf</groupId>
               <artifactId>spring-surf-api</artifactId>
               <version>${surf.version}</version>
               <scope>provided</scope>
           </dependency>
       </dependencies>
   </dependencyManagement>
```

We need to re-define some of the dependencies as the Platform and Surf applications now have different version numbers. Also, Surf is no longer a Spring Framework project but instead an Alfresco managed project.

9. Update the dependencies section in Share AMP

   In the `alfresco-extensions/all-in-one/share-amp/pom.xml` update the dependencies to look like:

```
<dependencies>

        <dependency>
            <groupId>${alfresco.groupId}</groupId>
            <artifactId>share</artifactId>
            <version>${share.version}</version>   <!-- use new share
 version -->
            <classifier>classes</classifier>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.alfresco.surf</groupId>  <!-- Surf now
 maintained by Alfresco -->
            <artifactId>spring-surf-api</artifactId>
        </dependency>
        <dependency>
            <groupId>${alfresco.groupId}</groupId>
            <artifactId>share-po</artifactId>
            <version>${share.version}</version>   <!-- use new share
 version -->
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>${alfresco.groupId}</groupId>
            <artifactId>share-po</artifactId>
            <version>${share.version}</version>   <!-- use new share
 version -->
            <classifier>tests</classifier>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.seleniumhq.selenium</groupId>
                    <artifactId>selenium-java</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>org.seleniumhq.selenium</groupId>
                    <artifactId>selenium-server</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <!-- Test NG is defined with test scope in share-po, so need it
 here too -->
        <!-- Alfresco code creates a wrapper around Test NG -->
        <dependency>
            <groupId>org.alfresco.test</groupId>
            <artifactId>alfresco-testng</artifactId>
            <version>1.1</version>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.hamcrest</groupId>
                    <artifactId>hamcrest-core</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>
```

Here we now use the new `share.version`. Also, Surf is no longer a Spring Framework project but instead an Alfresco managed project.

10. Add alfresco-share-services AMP to Platform WAR

    In the `alfresco-extensions/all-in-one/repo/pom.xml` add the following AMP dependency and overlay:

```
<dependencies>
        . . .

        <!-- We need Share Services on the platform for Share UI to
 work properly -->
        <dependency>
            <groupId>${alfresco.groupId}</groupId>
            <artifactId>alfresco-share-services</artifactId>
            <version>${share.version}</version>
            <type>amp</type>
        </dependency>

    </dependencies>
    <overlays>
        . . .

        <!-- We need Share Services on the platform for Share UI to
 work properly -->
        <overlay>
            <groupId>${alfresco.groupId}</groupId>
            <artifactId>alfresco-share-services</artifactId>
            <type>amp</type>
        </overlay>
    </overlays>
```

    Here we now use the new `share.version`.

11. Change version property for Share WAR

    In the `alfresco-extensions/all-in-one/share/pom.xml` change the version property used for the Share WAR artifact as it now has separate versioning from the Platform:

```
<dependencies>
        <dependency>
            <groupId>${alfresco.groupId}</groupId>
            <artifactId>${alfresco.share.artifactId}</artifactId>
            <version>${share.version}</version>   <!-- New separate
 share version prop -->
            <type>war</type>
        </dependency>
        . . .

    </dependencies>
```

    Here we now use the new `share.version`.

12. Clean metadata and content

    After setting a newer Alfresco version you will need to clean out current database (with metadata), content files, and indexes. It currently does not work to do an incremental upgrade with the SDK and the H2 database. You can clean the DB and content files by running the following command: `alfresco-extensions/all-in-one/mvn clean -Ppurge`

13. Run

    Run with the usual command: `mvn clean install -Prun`

You have now seen how to upgrade the Alfresco Platform Community version to 5.2.a and Alfresco Share version to 5.1.g for the different types of SDK projects, and also seen how to clean the database, index, and content before starting with the new version.

Upgrading SDK 2.2.0 projects from Enterprise 5.1.0 to greater than 5.1.0

These instructions will walk through what is needed when upgrading an SDK 2.2.0 project from using Alfresco Enterprise version 5.1.0 to using Enterprise version 5.1.1. These instructions also apply when upgrading to other versions greater than 5.1.0, such as 5.1.0.5.

This task assumes that you have an SDK 2.2.0 project to work with, see creating a project.

You will learn how to set a new version in all the different kinds of SDK project types.

Upgrading Alfresco version for a Repository AMP project.

1. Set new version.

   In the IDE, open up the Repository AMP project that you are working on. Then open the project file for it, for example `alfresco-extensions/component-a-repo/pom.xml`. Scroll down so you see the `properties` section:

```
<properties>
    <!-- The following are default values for data location and
 Alfresco Enterprise version.
         Uncomment if you need to change (Note. current default version
 for Enterprise edition is 5.1) -->
    <alfresco.version>5.1.1</alfresco.version>
    <!--<alfresco.data.location>/absolute/path/to/alf_data_dev</
alfresco.data.location> -->
```

   What you need to do here is uncomment the `alfresco.version` property, and then update the version to desired latest version (e.g. 5.1.0.5, 5.1.1). In this case we are upgrading to a newer Enterprise Edition (default is 5.1.0).

2. Clean metadata and content.

   After setting a newer Alfresco version you will need to clean out current database (with metadata), content files, and indexes. It currently does not work to do an incremental upgrade with the SDK and the H2 database. You can clean the DB and content files by running the following command: `alfresco-extensions/component-a-repo/mvn clean -Ppurge`

Upgrading Alfresco version for a Share AMP project.

3. Set new version.

   In the IDE, open up the Share AMP project that you are working on. Then open the project file for it, for example `alfresco-extensions/component-a-share/pom.xml`. Scroll down so you see the `properties` section:

```
<properties>
    <!-- The following are default values for data location and
 Alfresco version.
         Uncomment if you need to change -->
    <alfresco.version>5.1.1</alfresco.version>
...
```

   What you need to do here is uncomment the `alfresco.version` property, and then update the version to desired latest version (e.g. 5.1.0.5, 5.1.1). In this case we are upgrading to a newer Enterprise Edition (default is 5.1.0).

4. Update spring surf API dependency

In the same POM file update the Surf dependency so it looks like:

```
<dependency>
    <groupId>org.alfresco.surf</groupId>
    <artifactId>spring-surf-api</artifactId>
    <version>${dependency.surf.version}</version>
    <scope>provided</scope>
</dependency>
```

Surf is no longer a Spring Framework project but instead an Alfresco managed project.

5. Clean metadata and content

   When upgrading the Alfresco Share AMP it is not necessary to clean out a database or clean content because these are related to the Alfresco Repository application (alfresco.war) and not the Share Application (share.war).

6. Run

   When running we need to specify what Surf version that should be used: `mvn clean install -Ddependency.surf.version=6.3 -Pamp-to-war,enterprise` Alfresco Surf is now released independently from Alfresco Share.

Upgrading Alfresco version for an All-in-One (AIO) project.

7. Set new version.

   In the IDE, open up the All-in-One project that you are working on. Then open the project file for it, for example `alfresco-extensions/all-in-one/pom.xml`. Scroll down so you see the `properties` section:

```
<properties>
    <!-- The following are default values for data location, Alfresco
 Enterprise version, and Records Management Module version.
         Uncomment if you need to change (Note. current default version
 for Enterprise edition is 5.1) -->
    <alfresco.version>5.1.1</alfresco.version>
    <alfresco.rm.version>2.4</alfresco.rm.version>
    <!-- <alfresco.data.location>/absolute/path/to/alf_data_dev</
alfresco.data.location> -->
...
```

   What you need to do here is uncomment the `alfresco.version` property, and then update the version to desired latest version (e.g. 5.1.0.5, 5.1.1). In this case we are upgrading to a newer Enterprise edition (default is 5.1.0). Note also that in this case I'm using the Records Management module and I am updating the version for it at the same time by uncommenting the `alfresco.rm.version` property and setting new version (for information about how to enable RM in an All-In-One project see this article).

8. Update spring surf API dependency in Share AMP

   In the `alfresco-extensions/all-in-one/share-amp/pom.xml` update the dependency it looks like:

```
<dependency>
    <groupId>org.alfresco.surf</groupId>
    <artifactId>spring-surf-api</artifactId>
    <version>${dependency.surf.version}</version>
    <scope>provided</scope>
</dependency>
```

   Surf is no longer a Spring Framework project but instead an Alfresco managed project.

9. Clean metadata and content

After setting a newer Alfresco version you will need to clean out current database (with metadata), content files, and indexes. It currently does not work to do an incremental upgrade with the SDK and the H2 database. You can clean the DB and content files by running the following command: `alfresco-extensions/all-in-one/mvn clean -Ppurge`

10. Run

    When running we need to specify what Surf version that should be used: `mvn clean install -Ddependency.surf.version=6.3 -Prun,enterprise` Alfresco Surf is now released independently from Alfresco Share.

You have now seen how to upgrade the Alfresco Enterprise version to 5.1.1 (or for example 5.1.0.5) for the different types of SDK projects and how to clean the database, index, and content before starting with the new version.

Upgrading SDK version for an extension project

This section describes how to upgrade the SDK version that is used by your extension project.

These instructions include information about how to upgrade projects generated from each one of the artifacts. Make sure you are following upgrade instructions for the correct project type and "From version -> To version".

Make sure you have made a complete backup of your project before you start the upgrade process!

What changes are allowed in an SDK release?

The following describes the kind of changes you can expect (are allowed) in major, minor, and patch releases.

A 3 digit versioning scheme is used, **major.minor.patch** (e.g. 2.1.0). The following is a list of changes that can go into each one of these releases:

1. **major**
    a. *Backward incompatible changes* (e.g. changes in the archetype project structure, functional changes in archetypes POMs, functional changes in existing profiles)
    b. *Changes in the artifact naming*
2. **minor** Cannot change existing behaviors (e.g. existing profiles semantics, build lifecycle, archetype structure).
    a. *New features* (e.g. new alfresco-sdk-parent, new archetype profiles, new properties)
    b. *New artifacts*
3. **patch** Ideally no changes to the code of the archetypes.
    a. *Bug Fixes*
    b. *Limited changes to SDK parent and Alfresco Plugin*

Note that in addition to this there can be beta releases to give early access to features.

Upgrading SDK version from 2.0.0 to 2.1.0

This section contains instructions for how to upgrade an extension project from using SDK version 2.0.0 to using SDK version 2.1.0.

These instructions include information about how to upgrade projects generated from each one of the Maven artifacts. Make sure you are following upgrade instructions for the correct project type. Default Alfresco versions for SDK 2.0.0 is Community 5.0.c and Enterprise 5.0. After upgrading to SDK 2.1.0 the default Alfresco versions will be Community 5.0.d and Enterprise 5.0.1.

Make sure you have made a complete backup of your project before you start the upgrade process!

Upgrading a Repository AMP project from SDK 2.0.0 to 2.1.0

These instructions will walk through what is needed when upgrading a Repository AMP project from using SDK version 2.0.0 to using SDK version 2.1.0.

There are multiple ways to go about an SDK upgrade. These instructions assume that you have a Repository AMP project where the source code is managed by a Software Configuration Management (SCM) system such as Git or Subversion. And you cannot just through away the history of this project, you need to upgrade "in-place". On the other hand, if your project is small, and you don't mind starting with a new project in the SCM, it might be easier to just generate a new project from the Repository AMP 2.1.0 SDK archetype and move the code and other changes over to it from the SDK 2.0.0 project, but this method is not covered in this article.

In the following instructions the `REPO_AMP_PROJECT_PATH` variable denotes the path to where you have your Repository AMP project folder. So, for example, if your Repository AMP project was generated in the `C:\alfresco-extensions\acme-repo-amp` directory, then this directory path is the value of this variable.

Make sure you have made a complete backup of your project before you start the upgrade process!

1. Setting the SDK Version to 2.1.0.

   In the IDE, open up the `{REPO_AMP_PROJECT_PATH}/pom.xml` project file. Scroll down so you see the `parent` section. Then update it to look as follows:

   ```
   <parent>
       <groupId>org.alfresco.maven</groupId>
       <artifactId>alfresco-sdk-parent</artifactId>
       <version>2.1.0</version>
   </parent>
   ```

2. Remove the property used to specify the webapp path for Alfresco Repository web application.

   In the project file `{REPO_AMP_PROJECT_PATH}/pom.xml` scroll down to the `properties` section. Then **remove** the property called `alfresco.client.contextPath`. This property is already set to `/alfresco` in the SDK parent POM so no need to set it here.

3. Add the `amp-to-war` profile with rad dependency.

   In the IDE, open up the `{REPO_AMP_PROJECT_PATH}/pom.xml` project file. Scroll down so you see the `profiles` section. Then add the following profile to it:

   ```
       <!--
           If the 'amp-to-war' profile is enabled then make sure to bring
    in the alfresco-rad module,
           which has the H2 scripts and other RAD features.
           -->
       <profile>
           <id>amp-to-war</id>
           <dependencies>
               <dependency>
                   <groupId>org.alfresco.maven</groupId>
                   <artifactId>alfresco-rad</artifactId>
                   <version>${maven.alfresco.version}</version>
               </dependency>
           </dependencies>
       </profile>
   ```

4. Update the Tomcat virtual webapp context file.

   Open the `{REPO_AMP_PROJECT_PATH}/tomcat/context.xml` file. Change it to look like this for best RAD experience:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 ================================================================================
    This context file is used only in a development IDE for rapid
 development,
    it is never released with the Alfresco.war

 ================================================================================
>

<!-- Setup docBase to something like repo-amp/target/repo-amp-war
     and path to /alfresco
     The Alfresco.war 5.0 does not have a webapp (it used to have
 Alfresco Explorer but not anymore)
     that we will access, so this docBase might not be needed
-->
<Context docBase="${app.amp.client.war.folder}"
 path="${alfresco.client.contextPath}">

  <Resources className="org.apache.naming.resources.VirtualDirContext"
             extraResourcePaths="/=${project.build.directory}/
${project.build.finalName}/web" />

  <!-- Setup the virtual class path like this:
       1) target/classes
       2) target/${project.build.finalName}/config
       3) target/test-classes

       This way mvn compile can be invoked and all changes will be
 picked up
  -->
  <Loader searchVirtualFirst="true"
          className="org.apache.catalina.loader.VirtualWebappLoader"
          virtualClasspath="${project.build.outputDirectory};
${project.build.directory}/${project.build.finalName}/config;
${project.build.testOutputDirectory}" />


  <!-- This enables hot reloading of web resources from uncompressed
 jars (while in prod they would be loaded from  WEB-INF/lib/{\*.jar}/
META-INF/resources -->
  <JarScanner scanAllDirectories="true" />
</Context>
```

5. Replace run scripts.

   Version 2.1.0 of the SDK have changes to the Linux run scripts and have new run scripts for Windows. So it make sense to take the new scripts from a newly generated 2.1.0 Repository AMP project and replace the 2.0.0 scripts with them. So follow these instructions to generate a Repository AMP project based on the 2.1.0 archetype. Then just copy over the `{newly generated 2.1.0 Repo AMP}/run.*` scripts to the `{REPO_AMP_PROJECT_PATH}` directory, overwriting the `run.sh` script.

Your Repository AMP project should now be fully updated to use the 2.1.0 version of the SDK.

Upgrading a Share AMP project from SDK 2.0.0 to 2.1.0

These instructions will walk through what is needed when upgrading a Share AMP project from using SDK version 2.0.0 to using SDK version 2.1.0.

There are multiple ways to go about an SDK upgrade. These instructions assume that you have a Share AMP project where the source code is managed by a Software Configuration Management (SCM) system such as Git or Subversion. And you cannot just through away the history of this project, you need to upgrade "in-place". On the other hand, if your project is small, and you don't mind starting with a new project in the SCM, it might be easier to just generate a new project from the Share AMP 2.1.0 SDK archetype and move the code and other changes over to it from the SDK 2.0.0 project, but this method is not covered in this article.

In the following instructions the `SHARE_AMP_PROJECT_PATH` variable denotes the path to where you have your Share AMP project folder. So, for example, if your Share AMP project was generated in the `C:\alfresco-extensions\acme-share-amp` directory, then this directory path is the value of this variable.

Make sure you have made a complete backup of your project before you start the upgrade process!

1. Setting the SDK Version to 2.1.0.

   In the IDE, open up the `{SHARE_AMP_PROJECT_PATH}/pom.xml` project file. Scroll down so you see the `parent` section. Then update it to look as follows:

```
<parent>
    <groupId>org.alfresco.maven</groupId>
    <artifactId>alfresco-sdk-parent</artifactId>
    <version>2.1.0</version>
</parent>
```

2. Remove the property used to specify the webapp path for the Alfresco Share web application.

   In the project file `{SHARE_AMP_PROJECT_PATH}/pom.xml` scroll down to the `properties` section. Then **remove** the property called `alfresco.client.contextPath`. This property is now called `share.client.contextPath`, and it is already set to `/share` in the SDK parent POM, so no need to set it here.

3. Update the name of the property specifying Share Webapp aritifact ID.

   In the project file `{SHARE_AMP_PROJECT_PATH}/pom.xml` scroll down to the `properties` section. Then change the name of the property called `alfresco.client.war` to `app.amp.client.war.artifactId`.

4. Add a `build` section to enable the JS Compression plugin.

   In the same project file `{SHARE_AMP_PROJECT_PATH}/pom.xml` scroll down to the `dependencies` end tag. Then add the following `build` section after it with the `yuicompressor-maven-plugin` to enable JS compression:

```
. . .
</dependencies>

<build>
    <plugins>
        <!-- Compress JavaScript files and store as *-min.js -->
        <plugin>
            <groupId>net.alchim31.maven</groupId>
            <artifactId>yuicompressor-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

5. Update `share-config-custom.xml` to enable better RAD.

Open the `{SHARE_AMP_PROJECT_PATH}/src/test/resources/alfresco/web-extension/share-config-custom.xml` file and update the `web-framework` configuration so it looks like this:

```
<web-framework>
    <autowire>
        <!-- Changing this to 'development' currently breaks the Admin
 Console.
             Instead we make a POST to clear Share dependency caches,
 see 'clear-caches-refresh-ws' profile. -->
        <mode>production</mode> <!-- not really need in the long run,
 used for YUI - deprecate -->
    </autowire>

    <!--
         We don't need to do this when we have the new refresh mojos in
 the Alfresco plug-in.

         If resource caching has been disabled then all the dependency
 caches will be cleared
         before processing the Aikau jsonModel request...
            (i.e. this.dojoDependencyHandler.clearCaches() )

         For more information see the Aikau source code: https://
github.com/Alfresco/Aikau
    -->
    <disable-resource-caching>false</disable-resource-caching>
</web-framework>
```

6. Update the Tomcat virtual webapp context file.

   Open the `{SHARE_AMP_PROJECT_PATH}/tomcat/context.xml` file. Change it to look like this for best RAD experience:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 ================================================================================
    This context file is used only in a development IDE for rapid
 development,
    it is never released with the Alfresco.war

 ================================================================================
>

<!-- Setup docBase to something like share-amp/target/share-amp-war
     and path to /share -->
<Context docBase="${app.amp.client.war.folder}"
 path="${share.client.contextPath}">

  <Resources className="org.apache.naming.resources.VirtualDirContext"
    extraResourcePaths="/=${project.build.directory}/
${project.build.finalName}/web" />

  <!-- Configure where the Share (share.war) web application can load
 classes, test classes, and config -->
  <!-- Setup the virtual class path like this:
       1) target/classes
       2) target/${project.build.finalName}/config
       3) target/test-classes

       This way mvn compile can be invoked and all changes will be
 picked up
  -->
  <Loader searchVirtualFirst="true"
```

```
          className="org.apache.catalina.loader.VirtualWebappLoader"
          virtualClasspath="${project.build.outputDirectory};
${project.build.directory}/${project.build.finalName}/config;
${project.build.testOutputDirectory}" />

  <!-- This enables hot reloading of web resources from uncompressed
 jars (while in prod they would be loaded from  WEB-INF/lib/{\*.jar}/
META-INF/resources -->
  <JarScanner scanAllDirectories="true" />

</Context>
```

7. Replace run scripts.

   Version 2.1.0 of the SDK have changes to the Linux run scripts and have new run scripts for Windows. So it make sense to take the new scripts from a newly generated 2.1.0 Share AMP project and replace the 2.0.0 scripts with them. So follow these instructions to generate a Share AMP project based on the 2.1.0 archetype. Then just copy over the `{newly generated 2.1.0 Share AMP}/run.*` scripts to the `{SHARE_AMP_PROJECT_PATH}` directory, overwriting the `run.sh` script.

Your Share AMP project should now be fully updated to use the 2.1.0 version of the SDK.

Upgrading an All-in-One (AIO) project from SDK 2.0.0 to 2.1.0

These instructions will walk through what is needed when upgrading an AIO project from using SDK version 2.0.0 to using SDK version 2.1.0.

There are multiple ways to go about an SDK upgrade. These instructions assume that you have an All-in-One project where the source code is managed by a Software Configuration Management (SCM) system such as Git or Subversion. And you cannot just through away the history of this project, you need to upgrade "in-place". On the other hand, if your project is small, and you don't mind starting with a new project in the SCM, it might be easier to just generate a new project from the AIO 2.1.0 SDK archetype and move the code and other changes over to it from the SDK 2.0.0 project, but this method is not covered in this article.

In the following instructions the `AIO_PROJECT_PATH` variable denotes the path to where you have your All-in-One top project folder. So, for example, if your All-in-One project was generated in the `C:\alfresco-extensions\acme-poc` directory, then this directory path is the value of this variable.

Make sure you have made a complete backup of your project before you start the upgrade process!

Update the top AIO project file.

1. Setting the SDK Version to 2.1.0.

   In the IDE, open up the `{AIO_PROJECT_PATH}/pom.xml` project file. Scroll down so you see the `parent` section. Then update it to look as follows:

```
<parent>
    <groupId>org.alfresco.maven</groupId>
    <artifactId>alfresco-sdk-parent</artifactId>
    <version>2.1.0</version>
</parent>
```

2. Add a new property for the Alfresco Share Webapp location.

   In the same project file `{AIO_PROJECT_PATH}/pom.xml` scroll down to the `properties` section. Then update it with this extra property:

```
<properties>
```

```
. . .
    <!-- The Alfresco Share web application is accessible via this URL
 -->
    <share.client.url>http://localhost:8080/share</share.client.url>
```

3. Add a new property for the Alfresco RM Module version (**OPTIONAL**).

   If the Records Management (RM) module is used then add a property specifying the RM version that should be used. In the `properties` section add this extra property:

   ```
   <properties>
   . . .
       <alfresco.rm.version>2.3</alfresco.rm.version>
   ```

4. Add a `build` section to enable some plugins.

   In the same project file `{AIO_PROJECT_PATH}/pom.xml` scroll down to the `dependencyManagement` end tag. Then add the following `build` section after it with the `yuicompressor-maven-plugin` to enable JS compression and the `alfresco-maven-plugin` to enable webapp RAD development:

   ```
   . . .
   </dependencyManagement>

   <build>
       <plugins>
           <!-- Compress JavaScript files and store as *-min.js -->
           <plugin>
               <groupId>net.alchim31.maven</groupId>
               <artifactId>yuicompressor-maven-plugin</artifactId>
           </plugin>
           <plugin>
               <groupId>org.alfresco.maven.plugin</groupId>
               <artifactId>alfresco-maven-plugin</artifactId>
           </plugin>
       </plugins>
   </build>
   ```

5. Remove the Records Management (RM) profile.

   In the same project file `{AIO_PROJECT_PATH}/pom.xml` scroll down to the `profiles` section. Then **remove** the profile with the `rm` id (identifier). The RM module is added via the `{AIO_PROJECT_PATH}/repo/pom.xml` and the `{AIO_PROJECT_PATH}/share/pom.xml` project files. See further down in these instructions for more information.

6. Update the name of the Solr module.

   SDK version 2.1.0 comes with support for Solr 4, which is deployed directly from the maven artefact. The maven module just contains Solr 4 configuration information and because of this has changed name from `solr` to `solr-config`, so we need to update to the new name. In the same project file `{AIO_PROJECT_PATH}/pom.xml` scroll down to the `modules` section. Update it so it looks like this:

   ```
   . . .
   <modules>
       <module>repo-amp</module>
       <module>share-amp</module>
       <module>repo</module>
       <module>solr-config</module>
       <module>share</module>
       <module>runner</module>
   ```

```
</modules>
```

Note. You might have added extra modules that are not part of the AIO artefact, don't remove these modules from the definition.

Update the Repository Webapp (alfresco.war) Project file

7. Remove the Records Management (RM) profile.

   In the project file `{AIO_PROJECT_PATH}/repo/pom.xml` scroll down to the `profiles` section. Then **remove** the profile with the `rm` id (identifier). The RM module is now instead added permanently as a dependency and overlay.

8. Add a Records Management (RM) Module Dependency to Repository WAR (**OPTIONAL**)

   If the RM module is used, then it is now added permanently to the project instead of via profile activation. Add a dependency for it as follows. In the IDE, open up the `{AIO_PROJECT_PATH}/repo/pom.xml` project file. Scroll down so you see the `dependencies` section. Then add the following dependency:

```
<dependencies>
. . .
    <dependency>
        <groupId>${alfresco.groupId}</groupId>
        <artifactId>alfresco-rm</artifactId>
        <version>${alfresco.rm.version}</version>
        <type>amp</type>
    </dependency>
```

9. Add a Records Management (RM) Module Overlay Repository WAR (**OPTIONAL**)

   If the RM module is used, then it is now added permanently to the project instead of via profile activation. Add an overlay configuration as follows. In the `{AIO_PROJECT_PATH}/repo/pom.xml` project file scroll down so you see the `overlays` section. Then add the following overlay at the end:

```
<overlays>
. . .
    <overlay>
        <groupId>${alfresco.groupId}</groupId>
        <artifactId>alfresco-rm</artifactId>
        <type>amp</type>
    </overlay>
</overlays>
```

10. Make sure the Repository is using the Solr 4 subsystem

    When running all the web applictions during testing the repository webapp (alfresco.war) is reading its configuration from the `{AIO_PROJECT_PATH}/repo/src/main/properties/local/alfresco-global.properties` file. We need to update it so it uses Solr 4, the following properties should be changed:

```
index.subsystem.name=solr4
solr.backup.alfresco.remoteBackupLocation=${dir.root}/solr4Backup/
alfresco
solr.backup.archive.remoteBackupLocation=${dir.root}/solr4Backup/
archive
```

Update the Repository AMP Project file

11. Remove the property used to specify the artifact ID for the Alfresco WAR.

In the project file `{AIO_PROJECT_PATH}/repo-amp/pom.xml` scroll down to the `properties` section. Then **remove** the property called `alfresco.client.war`. This property is now called `app.amp.client.war.artifactId` and defaults to `alfresco`, so no need to set it in the repo-amp project file. This property is used when you run with the `-Pamp-to-war` profile.

12. Add a Records Management (RM) classes Dependency (**OPTIONAL**)

    If the RM module is used, then it is now added permanently to the project instead of via profile activation. So to get access to the RM classes add a dependency as follows. In the `{AIO_PROJECT_PATH}/repo-amp/pom.xml` project file scroll down so you see the `dependencies` section. Then add the following dependency:

```
<dependencies>
. . .
    <dependency>
        <groupId>${alfresco.groupId}</groupId>
        <artifactId>alfresco-rm</artifactId>
        <version>${alfresco.rm.version}</version>
        <classifier>classes</classifier>
    </dependency>
```

13. Remove the Records Management (RM) profile.

    In the project file `{AIO_PROJECT_PATH}/repo-amp/pom.xml` scroll down to the `profiles` section. Then **remove** the profile with the `rm` id (identifier). The RM classes are now instead added permanently as a dependency.

Update the Share Webapp (share.war) Project file

14. Add a properties section with a new property for the Alfresco Repository location.

    In the project file `{AIO_PROJECT_PATH}/share/pom.xml` add the following `properties` section just after the `parent` section:

```
. . .
</parent>

<properties>
   <!-- Used in share-config-custom.xml when testing.
        By default points to standard location (local) of Alfresco
 Repository -->
   <alfresco.repo.url>http://localhost:8080/alfresco</
alfresco.repo.url>
</properties>
```

15. Move `share-config-custom.xml` from `share-amp` to `share`.

    The share configuration file has moved from the share AMP sub project to the share WAR project. This is because it contains generic configuration such as where the Repository is running and RAD related configuration. Move the `{AIO_PROJECT_PATH}/share-amp/src/test/resources/alfresco/web-extension/share-config-custom.xml` file to the `{AIO_PROJECT_PATH}/share/src/main/resources/alfresco/web-extension` location. Then update the `web-framework` configuration so it looks like this:

```
<web-framework>
    <autowire>
        <!-- Changing this to 'development' currently breaks the Admin
 Console.
        Instead we make a POST to clear Share dependency caches, see
 'clear-caches-refresh-ws' profile. -->
```

```
        <mode>production</mode> <!-- not really need in the long run,
 used for YUI - deprecate -->
    </autowire>

    <!--
        We don't need to do this when we have the new refresh mojos in
 the Alfresco plug-in.

        If resource caching has been disabled then all the dependency
 caches will be cleared
        before processing the Aikau jsonModel request...
        (i.e. this.dojoDependencyHandler.clearCaches() )

        For more information see the Aikau source code: https://
github.com/Alfresco/Aikau
    -->
    <disable-resource-caching>false</disable-resource-caching>
</web-framework>
```

16. Remove the Records Management (RM) profile.

    In the project file `{AIO_PROJECT_PATH}/share/pom.xml` scroll down to the `profiles` section. Then **remove** the profile with the `rm` id (identifier). The RM module is now instead added permanently as a dependency and overlay.

17. Add a Records Management (RM) Module Dependency to Share WAR (**OPTIONAL**)

    If the RM module is used, then it is now added permanently to the project instead of via profile activation. Add a dependency for it as follows. In the IDE, open up the `{AIO_PROJECT_PATH}/share/pom.xml` project file. Scroll down so you see the `dependencies` section. Then add the following dependency:

```

<dependencies>
. . .
    <dependency>
        <groupId>${alfresco.groupId}</groupId>
        <artifactId>alfresco-rm-share</artifactId>
        <version>${alfresco.rm.version}</version>
        <type>amp</type>
    </dependency>
```

18. Add a Records Management (RM) Module Overlay to Share WAR (**OPTIONAL**)

    If the RM module is used, then it is now added permanently to the project instead of via profile activation. Add an overlay configuration as follows. In the `{AIO_PROJECT_PATH}/share/pom.xml` project file scroll down so you see the `overlays` section. Then add the following overlay at the end:

```

<overlays>
. . .
    <overlay>
        <groupId>${alfresco.groupId}</groupId>
        <artifactId>alfresco-rm-share</artifactId>
        <type>amp</type>
    </overlay>
</overlays>
```

Update the Share AMP Project file

19. Change the name of the property used to specify the artifact ID for the Share WAR.

    In the project file `{AIO_PROJECT_PATH}/share-amp/pom.xml` scroll down to the `properties` section. Then change the name of property called `alfresco.client.war` to its new name `app.amp.client.war.artifactId`. It defaults to `alfresco` so we need to override it here with the value `share`. This property is used when you run with the `-Pamp-to-war` profile.

20. Remove the property used to specify the location of the Alfresco Repository Webapp.

    In the project file `{AIO_PROJECT_PATH}/share-amp/pom.xml` scroll down to the `properties` section. Then **remove** the property called `alfresco.repo.url`. This property is only used by the `{AIO_PROJECT_PATH}/share/pom.xml` project in an All-in-One extension project.

21. Remove the property used to specify the port number for embedded Tomcat.

    In the project file `{AIO_PROJECT_PATH}/share-amp/pom.xml` scroll down to the `properties` section. Then **remove** the property called `maven.tomcat.port`. This property is only used by the `{AIO_PROJECT_PATH}/runner/pom.xml` project when starting an embeeded Tomcat instance. Default port number is configured to 8080 in the parent SDK pom.

22. Add dependencies for TestNG and Share Page Object classes.

    In version 2.1.0 of the SDK there are two new profiles called `regression-testing` and `functional-testing` that uses Page Objects (PO) to do functional testing of the Share Web application. We need to add all dependencies needed for these tests. In the `{AIO_PROJECT_PATH}/share-amp/pom.xml` project file scroll down so you see the `dependencies` section. Then add the following dependencies:

```
<dependencies>
. . .
     <!--
================================================================
        The following dependencies are needed to be able to compile the
        custom functional tests that are based on Page Objects (PO)

 ================================================================-->

    <!-- Bring in the Share Page Objects (PO) used in our functional
 tests.
         It contains page objects such as LoginPage -->
    <dependency>
        <groupId>${alfresco.groupId}</groupId>
        <artifactId>share-po</artifactId>
        <version>${alfresco.version}</version>
        <scope>test</scope>
    </dependency>
    <!-- Bring in the Share Page Object (PO) Tests that comes with
 Alfresco. It has
         the org.alfresco.po.share.AbstractTest class that our custom
 tests extend. -->
    <dependency>
        <groupId>${alfresco.groupId}</groupId>
        <artifactId>share-po</artifactId>
        <version>${alfresco.version}</version>
        <classifier>tests</classifier>
        <scope>test</scope>

        <!-- Exclude version 2.39.0 of selenium that does not work with
 latest FF browsers, we include
           version 2.45 later on here -->
        <exclusions>
            <exclusion>
                <groupId>org.seleniumhq.selenium</groupId>
                <artifactId>selenium-java</artifactId>
            </exclusion>
            <exclusion>
                <groupId>org.seleniumhq.selenium</groupId>
                <artifactId>selenium-server</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <!-- Bring in newer selenium version -->
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>2.45.0-alfresco</version>
    </dependency>
    <!-- Test NG is defined with test scope in share-po, so need it
 here too -->
    <!-- Alfresco code creates a wrapper around Test NG -->
    <dependency>
        <groupId>org.alfresco.test</groupId>
        <artifactId>alfresco-testng</artifactId>
        <version>1.1</version>
        <scope>test</scope>
        <exclusions>
            <exclusion>
                <groupId>org.hamcrest</groupId>
                <artifactId>hamcrest-core</artifactId>
            </exclusion>
        </exclusions>
    </dependency>

</dependencies>
```

Replace Modules and Scripts

23. Replace the `runner` module.

    The project configuration for the `runner` module has changed quite a bit in version 2.1.0 of the SDK. And there should not be much custom configuration done to it. So it make sense to take the `runner` module from a newly generated 2.1.0 AIO project and replace the 2.0.0 `runner` module with it. So follow these instructions to generate an AIO project based on the 2.1.0 archetype. Then delete the `{AIO_PROJECT_PATH}/runner` module/directory from the All-in-One project. Now copy the `{newly generated 2.1.0 AIO}/runner` module into the `{AIO_PROJECT_PATH}/runner` location.

    The new `runner` module probably does not have the same parent configuration as your {AIO_PROJECT_PATH} project. So open up the `{AIO_PROJECT_PATH}/runner/pom.xml` file and make sure the `parent` section is correct.

    If you have made changes to the virtual web application context because you have added more AMPs to the AIO project, then see these instructions for how to update the 2.1.0 runner context.

24. Replace the `solr` module.

    The project configuration for the `solr` module has changed completely from bringing in the complete Solr 1.4 web application to just bringing in the Solr 4 configuration. So it make sense to take the `solr-config` module from a newly generated 2.1.0 AIO project and replace the 2.0.0 `solr` module with it. So follow these instructions to generate an AIO project based on the 2.1.0 archetype (if you have not already done it). Then delete the `{AIO_PROJECT_PATH}/solr` module/directory from the All-in-One project. Now copy the `{newly generated 2.1.0 AIO}/solr-config` module into the `{AIO_PROJECT_PATH}/solr-config` location.

    The new `solr-config` module probably does not have the same parent configuration as your {AIO_PROJECT_PATH} project. So open up the `{AIO_PROJECT_PATH}/solr-config/pom.xml` file and make sure the `parent` section is correct.

If you have made changes to the Solr configuration, such as adding a synonyms list, then you will have to update the `solr-config` project with these changes.

25. Replace run scripts.

    Version 2.1.0 of the SDK have changes to the Linux run scripts and have new run scripts for Windows. So it make sense to take the new scripts from a newly generated 2.1.0 AIO project and replace the 2.0.0 scripts with them. So follow these instructions to generate an AIO project based on the 2.1.0 archetype (if you have not already done it). Then just copy over the `{newly generated 2.1.0 AIO}/run.*` scripts to the `{AIO_PROJECT_PATH}` directory, overwriting the `run.sh` script.

26. Remove the `alf_data_dev` directory.

    It is not possible to do an incremental H2 database schema update. The complete `alf_data_dev` directory needs to be deleted before you run the application again.

Your All-in-One project should now be fully updated to use the 2.1.0 version of the SDK.

Upgrading SDK version from 2.1.0 to 2.1.1

This section contains instructions for how to upgrade an extension project from using SDK version 2.1.0 to using SDK version 2.1.1.

These instructions include information about how to upgrade projects generated from each one of the Maven artifacts. Make sure you are following upgrade instructions for the correct project type. Default Alfresco versions for SDK 2.1.0 is Community 5.0.d and Enterprise 5.0.1. After upgrading to SDK 2.1.1 the default Alfresco versions will stay the same.

Make sure you have made a complete backup of your project before you start the upgrade process!

Upgrading a Repository AMP project from SDK 2.1.0 to 2.1.1

These instructions will walk through what is needed when upgrading a Repository AMP project from using SDK version 2.1.0 to using SDK version 2.1.1.

There are multiple ways to go about an SDK upgrade. These instructions assume that you have a Repository AMP project where the source code is managed by a Software Configuration Management (SCM) system such as Git or Subversion. And you cannot just through away the history of this project, you need to upgrade "in-place". On the other hand, if your project is small, and you don't mind starting with a new project in the SCM, it might be easier to just generate a new project from the Repository AMP 2.1.1 SDK archetype and move the code and other changes over to it from the SDK 2.1.0 project, but this method is not covered in this article.

In the following instructions the `REPO_AMP_PROJECT_PATH` variable denotes the path to where you have your Repository AMP project folder. So, for example, if your Repository AMP project was generated in the `C:\alfresco-extensions\acme-repo-amp` directory, then this directory path is the value of this variable.

Make sure you have made a complete backup of your project before you start the upgrade process!

1. Setting the SDK Version to 2.1.1.

   In the IDE, open up the `{REPO_AMP_PROJECT_PATH}/pom.xml` project file. Scroll down so you see the `parent` section. Then update it to look as follows:

```
<parent>
    <groupId>org.alfresco.maven</groupId>
    <artifactId>alfresco-sdk-parent</artifactId>
    <version>2.1.1</version>
</parent>
```

2. Update the commented out Alf Data location value.

   In the same project file update the property `alfresco.data.location` as follows:

   ```
   <properties>
       <!-- The following are default values for data location, Alfresco
    Community version.
            Uncomment if you need to change (Note. current default version
    for Enterprise edition is 5.0.1)
         <alfresco.version>5.0.d</alfresco.version>
         <alfresco.data.location>/absolute/path/to/alf_data_dev</
   alfresco.data.location> -->
   ```

3. Update version numbers and add a comment to the demo component bean definition.

   In the `{REPO_AMP_PROJECT_PATH}/src/main/amp/config/alfresco/module/repo-amp/context/service-context.xml` project file update as follows:

   ```
     <!-- A simple module component that will be executed once.
           Note. this module component will only be executed once, and
    then there will be an entry for it in the Repo.
           So doing for example $ mvn clean install -Prun twice will only
    execute this component the first time.
           You need to remove /alf_data_dev for it to be executed again.
    -->
       <bean ...
           <property name="sinceVersion" value="1.0" />
           <property name="appliesFromVersion" value="1.0" />
           ...
       </bean>
   ```

Your Repository AMP project should now be fully updated to use the 2.1.1 version of the SDK.

Upgrading a Share AMP project from SDK 2.1.0 to 2.1.1

These instructions will walk through what is needed when upgrading a Share AMP project from using SDK version 2.1.0 to using SDK version 2.1.1.

There are multiple ways to go about an SDK upgrade. These instructions assume that you have a Share AMP project where the source code is managed by a Software Configuration Management (SCM) system such as Git or Subversion. And you cannot just through away the history of this project, you need to upgrade "in-place". On the other hand, if your project is small, and you don't mind starting with a new project in the SCM, it might be easier to just generate a new project from the Share AMP 2.1.1 SDK archetype and move the code and other changes over to it from the SDK 2.1.0 project, but this method is not covered in this article.

In the following instructions the `SHARE_AMP_PROJECT_PATH` variable denotes the path to where you have your Share AMP project folder. So, for example, if your Share AMP project was generated in the `C:\alfresco-extensions\acme-share-amp` directory, then this directory path is the value of this variable.

Make sure you have made a complete backup of your project before you start the upgrade process!

Setting the SDK Version to 2.1.1.

In the IDE, open up the `{SHARE_AMP_PROJECT_PATH}/pom.xml` project file. Scroll down so you see the `parent` section. Then update it to look as follows:

```

<parent>
    <groupId>org.alfresco.maven</groupId>
    <artifactId>alfresco-sdk-parent</artifactId>
    <version>2.1.1</version>
</parent>
```

Your Share AMP project should now be fully updated to use the 2.1.1 version of the SDK.

Upgrading an All-in-One (AIO) project from SDK 2.1.0 to 2.1.1

These instructions will walk through what is needed when upgrading an AIO project from using SDK version 2.1.0 to using SDK version 2.1.1.

There are multiple ways to go about an SDK upgrade. These instructions assume that you have an All-in-One project where the source code is managed by a Software Configuration Management (SCM) system such as Git or Subversion. And you cannot just through away the history of this project, you need to upgrade "in-place". On the other hand, if your project is small, and you don't mind starting with a new project in the SCM, it might be easier to just generate a new project from the AIO 2.1.1 SDK archetype and move the code and other changes over to it from the SDK 2.1.0 project, but this method is not covered in this article.

In the following instructions the `AIO_PROJECT_PATH` variable denotes the path to where you have your All-in-One top project folder. So, for example, if your All-in-One project was generated in the `C:\alfresco-extensions\acme-poc` directory, then this directory path is the value of this variable.

Make sure you have made a complete backup of your project before you start the upgrade process!

1. Setting the SDK Version to 2.1.1.

   In the IDE, open up the `{AIO_PROJECT_PATH}/pom.xml` project file. Scroll down so you see the `parent` section. Then update it to look as follows:

   ```
   <parent>
       <groupId>org.alfresco.maven</groupId>
       <artifactId>alfresco-sdk-parent</artifactId>
       <version>2.1.1</version>
   </parent>
   ```

2. Update the commented out Alf Data location value.

   In the same parent project file update the property `alfresco.data.location` as follows:

   ```
   <properties>
       <!-- The following are default values for data location, Alfresco
    Community version, and Records Management Module version.
            Uncomment if you need to change (Note. current default version
    for Enterprise edition is 5.0.1)
         <alfresco.version>5.0.d</alfresco.version>
         <alfresco.rm.version>2.3</alfresco.rm.version>
         <alfresco.data.location>/absolute/path/to/alf_data_dev</
   alfresco.data.location> -->
   ```

3. Add test scope for Selenium dependency in Share AMP.

   In the `{AIO_PROJECT_PATH}/share-amp/pom.xml` project file scroll down to the following `dependency`. Then add `<scope>test</scope>`:

   ```
   <dependency>
      <groupId>org.seleniumhq.selenium</groupId>
      <artifactId>selenium-java</artifactId>
      <version>2.45.0-alfresco</version>
      <scope>test</scope>
   </dependency>
   ```

4. Update version numbers and add a comment to the demo component bean definition in the Repo AMP.

In the `{AIO_PROJECT_PATH}/repo-amp/src/main/amp/config/alfresco/module/repo-amp/context/service-context.xml` project file update as follows:

```
 <!-- A simple module component that will be executed once.
        Note. this module component will only be executed once, and
 then there will be an entry for it in the Repo.
        So doing for example $ mvn clean install -Prun twice will only
 execute this component the first time.
        You need to remove /alf_data_dev for it to be executed again.
 -->
    <bean ...
        <property name="sinceVersion" value="1.0" />
        <property name="appliesFromVersion" value="1.0" />
        ...
    </bean>
```

5. Add property for module log level to repo project.

   In the `{AIO_PROJECT_PATH}/repo/pom.xml` project file add the following `properties` section:

```
<properties>
        <!-- During development we set log root level to Debug,
            this will be applicable to the log configuration in
            repo/src/main/resources/alfresco/extension/dev-
log4j.properties,
            such as DemoComponent logging. -->
        <app.log.root.level>DEBUG</app.log.root.level>
    </properties>
```

6. Configure module logging in repo project.

   In the `{AIO_PROJECT_PATH}/repo/src/main/resources/alfresco/extension/dev-log4j.properties` log configuration file add the following line:

```
log4j.logger.org.alfresco.demoamp.DemoComponent=${app.log.root.level}
```

7. In the runner project update the properties section.

   Update and add properties as follows in the `{AIO_PROJECT_PATH}/runner/pom.xml` project file :

```
<properties>
    <alfresco.solr.dir>${alfresco.data.location}/solr4</
alfresco.solr.dir>
    <alfresco.solr.home.dir>${alfresco.solr.dir}/config</
alfresco.solr.home.dir>
    <alfresco.solr.data.dir>${alfresco.solr.dir}/data</
alfresco.solr.data.dir>
</properties>
```

8. In the Repo tomcat context file add a comment about resource loading

   In the `{AIO_PROJECT_PATH}/runner/tomcat/context-repo.xml` file add comments as follows:

```
   <Resources className="org.apache.naming.resources.VirtualDirContext"
               extraResourcePaths="/=${project.parent.basedir}/repo-
amp/target/repo-amp/web" />
    <!-- IMPORTANT! The extraResourcePaths string need to be on one
 continues line, so if we add another Repo AMP,
                  it would look something like this:
    <Resources
 className="org.apache.naming.resources.VirtualDirContext"
```

```
                extraResourcePaths="/=${project.parent.basedir}/repo-
amp/target/repo-amp/web,/=${project.parent.basedir}/component-a-repo/
target/component-a-repo/web" />
    -->
```

9. In the Share tomcat context file add a comment about resource loading

   In the `{AIO_PROJECT_PATH}/runner/tomcat/context-share.xml` file add comments as follows:

```
    <Resources className="org.apache.naming.resources.VirtualDirContext"
                extraResourcePaths="/=${project.parent.basedir}/share-
amp/target/share-amp/web" />
    <!-- IMPORTANT! The extraResourcePaths string need to be on one
 continues line, so if we add another Share AMP,
                     it would look something like this:
    <Resources
 className="org.apache.naming.resources.VirtualDirContext"
                extraResourcePaths="/=${project.parent.basedir}/share-
amp/target/share-amp/web,/=${project.parent.basedir}/component-a-share/
target/component-a-share/web" />
                -->
```

10. In the Solr tomcat context file update all paths

    In the `{AIO_PROJECT_PATH}/runner/tomcat/context-solr.xml` file update the environment property values as follows:

```
<Context>
    <Environment name="solr/home"        type="java.lang.String"
 value="${alfresco.solr.home.dir}/" override="true"/>
    <Environment name="solr/model/dir"   type="java.lang.String"
 value="${alfresco.solr.home.dir}/alfrescoModels/" override="true"/>
    <Environment name="solr/content/dir" type="java.lang.String"
 value="${alfresco.solr.data.dir}/content/" override="true"/>
```

11. In the Solr Configuration project update the properties section

    Open up the `{AIO_PROJECT_PATH}/solr-config/pom.xml` project file. Update the `properties` sections as follows:

```
<properties>
        <alfresco.solr.dir>${alfresco.data.location}/solr4</
alfresco.solr.dir>
        <alfresco.solr.home.dir>${alfresco.solr.dir}/config</
alfresco.solr.home.dir>
        <alfresco.solr.data.dir>${alfresco.solr.dir}/data</
alfresco.solr.data.dir>
    </properties>
```

Your All-in-One project should now be fully updated to use the 2.1.1 version of the SDK.

Upgrading SDK version from 2.1.1 to 2.2.0

This section contains instructions for how to upgrade an extension project from using SDK version 2.1.1 to using SDK version 2.2.0.

These instructions include information about how to upgrade projects generated from each one of the Maven artifacts. Make sure you are following upgrade instructions for the correct project type. Default Alfresco versions for SDK 2.1.1 is Community 5.0.d and Enterprise 5.0.1. After upgrading to SDK 2.2.0 the default Alfresco versions will be Community 5.1.d and Enterprise 5.1.0.

Make sure you have made a complete backup of your project before you start the upgrade process!

Upgrading a Repository AMP project from SDK 2.1.1 to 2.2.0

These instructions will walk through what is needed when upgrading a Repository AMP project from using SDK version 2.1.1 to using SDK version 2.2.0.

There are multiple ways to go about an SDK upgrade. These instructions assume that you have a Repository AMP project where the source code is managed by a Software Configuration Management (SCM) system such as Git or Subversion. And you cannot just through away the history of this project, you need to upgrade "in-place". On the other hand, if your project is small, and you don't mind starting with a new project in the SCM, it might be easier to just generate a new project from the Repository AMP 2.2.0 SDK archetype and move the code and other changes over to it from the SDK 2.1.1 project, but this method is not covered in this article.

In the following instructions the `REPO_AMP_PROJECT_PATH` variable denotes the path to where you have your Repository AMP project folder. So, for example, if your Repository AMP project was generated in the `C:\alfresco-extensions\acme-repo-amp` directory, then this directory path is the value of this variable.

Make sure you have made a complete backup of your project before you start the upgrade process!

1. Setting the SDK Version to 2.2.0.

   In the IDE, open up the `{REPO_AMP_PROJECT_PATH}/pom.xml` project file. Scroll down so you see the `parent` section. Then update it to look as follows:

```
<parent>
    <groupId>org.alfresco.maven</groupId>
    <artifactId>alfresco-sdk-parent</artifactId>
    <version>2.2.0</version>
</parent>
```

2. Add dependency for H2 database scripts.

   In the same project file add the following dependency:

```
<!-- If we are running tests then make the H2 Scripts available
         Note. tests are skipped when you are running -Pamp-to-war -->
    <dependency>
        <groupId>${alfresco.groupId}</groupId>
        <artifactId>alfresco-repository</artifactId>
        <version>${alfresco.version}</version>
        <classifier>h2scripts</classifier>
        <scope>test</scope>
        <exclusions>
            <exclusion>
                <groupId>*</groupId>
                <artifactId>*</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
```

3. Remove `alfresco-rad` dependency.

   This artifact previously contained the H2 database scripts but they are now available separately. In the same project file **remove** the following profile and dependency:

```
<!--
        If the 'amp-to-war' profile is enabled then make sure to bring
 in the alfresco-rad module,
        which has the H2 scripts and other RAD features.

        TODO: TO INVESTIGATE: This dependency is already defined in the
 parent SDK pom in the 'amp-to-war' profile
              but this does not work, it is not include...
```

```
    <profile>
        <id>amp-to-war</id>
        <dependencies>
            <dependency>
                <groupId>org.alfresco.maven</groupId>
                <artifactId>alfresco-rad</artifactId>
                <version>${maven.alfresco.version}</version>
            </dependency>
        </dependencies>
    </profile> -->
```

4. Remove Spring Loaded configuration from run scripts.

   Spring Loaded currently blocks the Repository (Platform) from starting. Update the `{REPO_AMP_PROJECT_PATH}/run.sh` and `run.bat` so they don't use Spring Loaded, change the `MAVEN_OPTS` so it looks like this:

```
run.sh: MAVEN_OPTS="-Xms256m -Xmx2G" mvn integration-test -Pamp-to-war
run.bat: set MAVEN_OPTS=-Xms256m -Xmx2G
```

5. Update the Virtual Webapp Context for Repository (alfresco.war).

   Update the virtual webapp context to reflect new directory names and the change so resources can be overridden (it is used when running with `-Pamp-to-war`). Open the `{REPO_AMP_PROJECT_PATH}/tomcat/context.xml` file and update it so it looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 ================================================================================
    This context file is used only in a development IDE for rapid
 development,
    it is never released with the Alfresco.war

 ================================================================================
>

<!-- Setup a virtual context for the /alfresco webapp by specifying
 this as path for Context.
     The amp-to-war profile uses the tomcat7-maven-plugin to kick off
 the webapp.
     This profile is used for both the repo and share AMP archetypes,
 and has no config for path or resources,
     so we need to specify here both the context path and where the
 webapp resources can be found.

     The webapp resources are located in the {repo-amp-dir}/target/amp-
war directory, However, we
     cannot just set this up as the docBase attribute for the Context
 as it would always be read
     before any paths in the extraResourcePaths. So to allow for
 customizations to override
     stuff in the alfresco.war webapp, such as the /images/logo/
logo.png, we add the webapp resource
     path last in the extraResourcePaths.

     Note. Alfresco.war 5.0 does not have a webapp, just an index page,
 the Alfresco Explorer webapp is no longer available.
-->
<Context path="${alfresco.client.contextPath}">
    <Resources
 className="org.apache.naming.resources.VirtualDirContext"
               extraResourcePaths="/=${project.build.directory}/amp/
web,${app.amp.client.war.folder}" />

    <!-- Setup the virtual class path like this:
         1) target/classes
```

```
          2) target/amp/config
          3) target/test-classes

          This way mvn compile can be invoked and all changes will be
 picked up
    -->
    <Loader searchVirtualFirst="true"
            className="org.apache.catalina.loader.VirtualWebappLoader"
            virtualClasspath="${project.build.outputDirectory};
${project.build.directory}/amp/config;
${project.build.testOutputDirectory}" />


    <!-- This enables hot reloading of web resources from uncompressed
 jars (while in prod they would be loaded from  WEB-INF/lib/{\*.jar}/
META-INF/resources -->
    <JarScanner scanAllDirectories="true" />
</Context>
```

6. Update the AMP module Spring context load order.

   Open the `{REPO_AMP_PROJECT_PATH}/src/main/amp/config/alfresco/module/<module-id>/module-context.xml` file and update it so it looks like this:

```
<beans>
 <!-- This is filtered by Maven at build time, so that module name is
 single sourced. -->
 <!-- Note. The bootstrap-context.xml file has to be loaded first.
      Otherwise your custom models are not yet loaded when your service
 beans are instantiated and you
      cannot for example register policies on them. -->
        <import resource="classpath:alfresco/module/
${project.artifactId}/context/bootstrap-context.xml" />
        <import resource="classpath:alfresco/module/
${project.artifactId}/context/service-context.xml" />
        <import resource="classpath:alfresco/module/
${project.artifactId}/context/webscript-context.xml" />
</beans>
```

7. Update the AMP module version to align with Maven Artifact version.

   Open the `{REPO_AMP_PROJECT_PATH}/src/main/amp/module.properties` file and update the version property:

```
module.version=${project.version}
```

8. Finally remove current `alf_data_dev` directory with previous database.

   Remove the `{REPO_AMP_PROJECT_PATH}/alf_data_dev` directory. This is needed as the H2 script artifact does not currently contain upgrade scripts.

Your Repository AMP project should now be fully updated to use the 2.2.0 version of the SDK.

Upgrading a Share AMP project from SDK 2.1.1 to 2.2.0

These instructions will walk through what is needed when upgrading a Share AMP project from using SDK version 2.1.1 to using SDK version 2.2.0.

There are multiple ways to go about an SDK upgrade. These instructions assume that you have a Share AMP project where the source code is managed by a Software Configuration Management (SCM) system such as Git or Subversion. And you cannot just through away the history of this project, you need to upgrade "in-place". On the other hand, if your project is small, and you don't mind starting with a new project in the SCM, it might be easier to just generate a new project from the Share AMP 2.2.0 SDK archetype and move the code and other changes over to it from the SDK 2.1.1 project, but this method is not covered in this article.

In the following instructions the `SHARE_AMP_PROJECT_PATH` variable denotes the path to where you have your Share AMP project folder. So, for example, if your Share AMP project was generated in the `C:\alfresco-extensions\acme-share-amp` directory, then this directory path is the value of this variable.

Make sure you have made a complete backup of your project before you start the upgrade process!

1. Setting the SDK Version to 2.2.0.

   In the IDE, open up the `{SHARE_AMP_PROJECT_PATH}/pom.xml` project file. Scroll down so you see the `parent` section. Then update it to look as follows:

   ```
   <parent>
       <groupId>org.alfresco.maven</groupId>
       <artifactId>alfresco-sdk-parent</artifactId>
       <version>2.2.0</version>
   </parent>
   ```

2. Remove the Enterprise profile dependency.

   This is not needed any more. In the same project file **remove** the following profile and dependency:

   ```
   <profiles>
           <!--
               Brings in the extra Enterprise specific share classes,
               if the 'enterprise' profile has been activated, needs to be
    activated manually. -->
           <profile>
               <id>enterprise</id>
               <dependencies>
                   <dependency>
                       <groupId>${alfresco.groupId}</groupId>
                       <artifactId>share-enterprise</artifactId>
                       <version>${alfresco.version}</version>
                       <classifier>classes</classifier>
                       <scope>provided</scope>
                   </dependency>
               </dependencies>
           </profile>
       </profiles>
   ```

3. Update the Virtual Webapp Context for Share (share.war).

   Update the virtual webapp context to reflect new directory names and the change so resources can be overridden (it is used when running with -Pamp-to-war). Open the `{SHARE_AMP_PROJECT_PATH}/tomcat/context.xml` file and update it so it looks like this:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <!--
    ================================================================================
      This context file is used only in a development IDE for rapid
    development,
      it is never released with the Alfresco.war

    ================================================================================
   >

   <!-- Setup a virtual context for the /share webapp by specifying this
    as path for Context.
       The amp-to-war profile uses the tomcat7-maven-plugin to kick off
    the webapp.
       This profile is used for both the repo and share AMP archetypes,
    and has no config for path or resources,
   ```

```
     so we need to specify here both the context path and where the
 webapp resources can be found.

     The webapp resources are located in the {share-amp-dir}/target/
amp-war directory, However, we
     cannot just set this up as the docBase attribute for the Context
 as it would always be read
     before any paths in the extraResourcePaths. So to allow for
 customizations to override
     stuff in the share.war webapp, such as the /favicon.ico, we add
 the webapp resource
     path last in the extraResourcePaths.

     Note. most of the UI customizations for Share are done via custom
 themes.
-->
<Context path="${share.client.contextPath}">
  <Resources className="org.apache.naming.resources.VirtualDirContext"
    extraResourcePaths="/=${project.build.directory}/amp/web,
${app.amp.client.war.folder}" />

  <!-- Configure where the Share (share.war) web application can load
 classes, test classes, and config -->
  <!-- Setup the virtual class path like this:
       1) target/classes
       2) target/amp/config
       3) target/test-classes

       This way mvn compile can be invoked and all changes will be
 picked up
  -->
  <Loader searchVirtualFirst="true"
          className="org.apache.catalina.loader.VirtualWebappLoader"
          virtualClasspath="${project.build.outputDirectory};
${project.build.directory}/amp/config;
${project.build.testOutputDirectory}" />

  <!-- This enables hot reloading of web resources from uncompressed
 jars (while in prod they would be loaded from  WEB-INF/lib/{\*.jar}/
META-INF/resources -->
  <JarScanner scanAllDirectories="true" />

</Context>
```

4. Update the AMP module version to align with Maven Artifact version.

   Open the `{SHARE_AMP_PROJECT_PATH}/src/main/amp/module.properties` file and update the version property:

   ```
   module.version=${project.version}
   ```

Your Share AMP project should now be fully updated to use the 2.2.0 version of the SDK.

Upgrading an All-in-One (AIO) project from SDK 2.1.1 to 2.2.0

These instructions will walk through what is needed when upgrading an AIO project from using SDK version 2.1.1 to using SDK version 2.2.0.

There are multiple ways to go about an SDK upgrade. These instructions assume that you have an All-in-One project where the source code is managed by a Software Configuration Management (SCM) system such as Git or Subversion. And you cannot just through away the history of this project, you need to upgrade "in-place". On the other hand, if your project is small, and you don't mind starting with a new project in the SCM, it might be easier to just generate a new project from the AIO 2.2.0 SDK archetype and move the code and other changes over to it from the SDK 2.1.1 project, but this method is not covered in this article.

In the following instructions the `AIO_PROJECT_PATH` variable denotes the path to where you have your All-in-One top project folder. So, for example, if your All-in-One project was generated in the `C:\alfresco-extensions\acme-poc` directory, then this directory path is the value of this variable.

Make sure you have made a complete backup of your project before you start the upgrade process!

1. Setting the SDK Version to 2.2.0.

   In the IDE, open up the `{AIO_PROJECT_PATH}/pom.xml` project file. Scroll down so you see the `parent` section. Then update it to look as follows:

   ```
   <parent>
       <groupId>org.alfresco.maven</groupId>
       <artifactId>alfresco-sdk-parent</artifactId>
       <version>2.2.0</version>
   </parent>
   ```

2. Add dependency for H2 database scripts.

   In the same project file add a new `dependencies` section with the following dependency:

   ```
   <dependencies>
           <!-- If we are running tests then make the H2 Scripts
    available.
                Note. tests are skipped when you are running -Prun -->
       <dependency>
           <groupId>${alfresco.groupId}</groupId>
           <artifactId>alfresco-repository</artifactId>
           <version>${alfresco.version}</version>
           <classifier>h2scripts</classifier>
           <scope>test</scope>
           <exclusions>
               <exclusion>
                   <groupId>*</groupId>
                   <artifactId>*</artifactId>
               </exclusion>
           </exclusions>
       </dependency>
   </dependencies>
   ```

3. Remove Spring Loaded configuration from run scripts.

   Spring Loaded currently blocks the Repository (Platform) from starting. Update the `{AIO_PROJECT_PATH}/run.sh` and `run.bat` so they don't use Spring Loaded, change the `MAVEN_OPTS` so it looks like this:

   ```
   run.sh: MAVEN_OPTS="-Xms256m -Xmx2G" mvn clean install -Prun
   run.bat: set MAVEN_OPTS=-Xms256m -Xmx2G
   ```

4. Update the Repo AMP module Spring context load order.

   Open the `{AIO_PROJECT_PATH}/<repo-amp-id>/src/main/amp/config/alfresco/module/<module-id>/module-context.xml` file and update it so it looks like this:

   ```
   <beans>
          <!-- This is filtered by Maven at build time, so that module
    name is single sourced. -->
    <!-- Note. The bootstrap-context.xml file has to be loaded first.
         Otherwise your custom models are not yet loaded when your service
    beans are instantiated and you
         cannot for example register policies on them. -->
           <import resource="classpath:alfresco/module/
   ${project.artifactId}/context/bootstrap-context.xml" />
   ```

```
        <import resource="classpath:alfresco/module/
${project.artifactId}/context/service-context.xml" />
        <import resource="classpath:alfresco/module/
${project.artifactId}/context/webscript-context.xml" />
</beans>
```

5. Update the Repo AMP module version to align with Maven Artifact version.

   Open the `{AIO_PROJECT_PATH}<repo-amp-id>/src/main/amp/module.properties` file and update the version property:

   ```
   module.version=${project.version}
   ```

6. Remove `selenium-java` dependency from the Share AMP project.

   Open the `{AIO_PROJECT_PATH}/<share-amp-id>/pom.xml` file and **remove** the following dependency:

   ```
   <!-- Bring in newer selenium version -->
       <dependency>
           <groupId>org.seleniumhq.selenium</groupId>
           <artifactId>selenium-java</artifactId>
           <version>2.45.0-alfresco</version>
           <scope>test</scope>
       </dependency>
   ```

7. Update the Share AMP module version to align with Maven Artifact version.

   Open the `{AIO_PROJECT_PATH}/<share-amp-id>/src/main/amp/module.properties` file and update the version property:

   ```
   module.version=${project.version}
   ```

8. Update the Share AMP Page Object (PO)

   The Share page object project has been updated a bit so the `{AIO_PROJECT_PATH}/<share-amp-id>/src/test/java/<package>/demoamp/po/DemoPage.java` class need to be updated a bit:

   ```
   import org.alfresco.po.share.SharePage;
   import org.alfresco.po.RenderTime;
   import org.openqa.selenium.NoSuchElementException;
   import org.openqa.selenium.WebElement;
   import org.openqa.selenium.support.FindBy;

   public class DemoPage extends SharePage {
       @FindBy(id="DEMO_SIMPLE_LOGO")
       WebElement logo;

       @FindBy(id="DEMO_SIMPLE_MSG")
       WebElement msg;

       @SuppressWarnings("unchecked")
       @Override
       public DemoPage render(RenderTime timer) {

           // Wait for logo and message to display, then consider page
    rendered
           while (true) {
               timer.start();
               try {
                   if (isSimpleLogoDisplayed() && isMessageDisplayed()) {
                       break;
                   }
               } catch (NoSuchElementException nse) {
               } finally {
   ```

```
                timer.end();
            }
        }

        return this;
    }

    public boolean isSimpleLogoDisplayed() {
        return isDisplayed(logo);
    }

    public boolean isMessageDisplayed() {
        return isDisplayed(msg);
    }

    public String getMessage() {
        return msg.getText();
    }
}
```

9. And update the Share AMP Page Object (PO) Test

   The Share page object project has been updated a bit so the `{AIO_PROJECT_PATH}/<share-amp-id>/src/test/java/<package>/demoamp/DemoPageTestIT.java` class need to be updated a bit:

```
import org.alfresco.tut.demoamp.po.DemoPage;
import org.alfresco.po.share.LoginPage;
import org.alfresco.po.AbstractTest;
import org.alfresco.po.share.PeopleFinderPage;
import org.testng.Assert;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;

public class DemoPageTestIT extends AbstractTest {
    DemoPage page;

    @BeforeClass(groups = {"alfresco-one"})
    public void prepare() throws Exception {
        // Navigate to share, which will redirect to Login page
        driver.navigate().to(shareUrl + "/page");

        // Resolve/Bind current page to LoginPage object
        LoginPage loginPage = resolvePage(driver).render();
        loginPage.loginAs(username, password);
    }

    @BeforeMethod
    public void loadPage() {
        // Goto demo page
        driver.navigate().to(shareUrl + "/page/hdp/ws/simple-page");

        // We need to instantiate the page like this as it is not yet
 in
        // the factory known list of pages
        page = factoryPage.instantiatePage(driver, DemoPage.class);
    }

    @Test
    public void findLogo() {
        Assert.assertTrue(page.isSimpleLogoDisplayed());
    }

    @Test
    public void messageIsDisplayed() {
        page.render();
        String msg = page.getMessage();
        Assert.assertNotNull(msg);
        Assert.assertEquals("Hello from i18n!", msg);
    }

    /**
     * Example of test reusing methods in abstract share page objects.
     */
    @Test
    public void titleDisplayed() {
        // Invoke render when ready to use page object.
        page.render();
        Assert.assertNotNull(page);
        Assert.assertTrue(page.getTitle().contains("This is a simple
 page"));
    }

    /**
     * Test that show how we are able to reuse share page objects
     * objects in particular the navigation object.
     */
    @Test
    public void navigate() {
        Assert.assertNotNull(page.getNav());
        PeopleFinderPage peopleFinderPage =
 page.getNav().selectPeople().render();
        Assert.assertNotNull(peopleFinderPage);
    }
}
```

10. Add a Spring version property to the Runner project

    We will need a newer Spring version than 3 to use annotations in the Share PO test classes. Add the following properties section in the `{AIO_PROJECT_PATH}/runner/pom.xml` file:

```
<properties>
        <!-- Bring in newer Spring with support for annotations, used
 for Page Object tests -->
        <spring.version>4.1.6.RELEASE</spring.version>
</properties>
```

11. Add H2 db script dependency to the `tomcat7-maven-plugin` in the Runner project

    These scripts come in a separate artifact now and not in the `alfresco-rad` artifact. In the same project file add a new `dependencies` section to the plug-in as follows (note. there are actually 2 dependencies to add):

```
<plugin>
    <groupId>org.apache.tomcat.maven</groupId>
    <artifactId>tomcat7-maven-plugin</artifactId>
    <dependencies>
        <!-- Bring in the H2 Database scripts needed when running
 embedded, they are now
             available from the standard generated artifacts, no longer
 needed to be picked
             up from the alfresco-rad project -->
        <dependency>
            <groupId>org.alfresco</groupId>
            <artifactId>alfresco-repository</artifactId>
            <version>${alfresco.version}</version>
            <classifier>h2scripts</classifier>
            <exclusions>
                <exclusion>
                    <groupId>*</groupId>
                    <artifactId>*</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <!-- Explicitly bring in the Plexus Archiver so assembly goes
 quicker -->
        <dependency>
            <groupId>org.codehaus.plexus</groupId>
            <artifactId>plexus-archiver</artifactId>
            <version>2.3</version>
        </dependency>
    </dependencies>
    <executions>
    . . .
```

12. Update dependencies section for the `regression-testing` profile in the Runner project

    We need to bring in Spring 4 for example, in the same project file, update the `profile dependencies` section so it looks like this:

```
...<dependencies>
        <!-- Bring in the Share Page Objects (PO) used in our
 functional tests.
             It contains page objects such as LoginPage and it also
 brings
             in selenium-grid and selenium. -->
        <dependency>
            <groupId>${alfresco.groupId}</groupId>
            <artifactId>share-po</artifactId>
            <version>${alfresco.version}</version>
            <scope>test</scope>
        </dependency>
        <!-- Bring in the Share Page Object (PO) Tests that comes with
 Alfresco. It has
             the org.alfresco.po.share.AbstractTest class that our
 custom tests extend. -->
        <dependency>
            <groupId>${alfresco.groupId}</groupId>
            <artifactId>share-po</artifactId>
            <version>${alfresco.version}</version>
            <classifier>tests</classifier>
            <scope>test</scope>

            <!-- Exclude selenium as it is already brought in by share-
po dependency above -->
            <exclusions>
                <exclusion>
                    <groupId>org.seleniumhq.selenium</groupId>
                    <artifactId>selenium-java</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>org.seleniumhq.selenium</groupId>
                    <artifactId>selenium-server</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <!-- Test NG is defined with test scope in share-po, so need it
 here too -->
        <!-- Alfresco code creates a wrapper around Test NG -->
        <dependency>
            <groupId>org.alfresco.test</groupId>
            <artifactId>alfresco-testng</artifactId>
            <version>1.1</version>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.hamcrest</groupId>
                    <artifactId>hamcrest-core</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <!-- Bring in newer selenium version if required
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>2.48.0</version>
            <scope>test</scope>
        </dependency>
        -->
        <!-- Need to bring in a newer Spring that supports annotations,
 Alfresco brings in older one -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
            <version>${spring.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aspects</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring.version}</version>
        </dependency>
    </dependencies>...
```

13. Update dependencies section for the `functional-testing` profile in the Runner project

    Same thing as for the regression testing, we need to bring in Spring 4, in the same project file, update the `profile dependencies` section so it looks like this:

```
...<dependencies>
        <!-- Bring in the Share Page Objects (PO) used in our
 functional tests.
             It contains page objects such as LoginPage and it also
 brings
             in selenium-grid and selenium. -->
        <dependency>
            <groupId>${alfresco.groupId}</groupId>
            <artifactId>share-po</artifactId>
            <version>${alfresco.version}</version>
            <scope>test</scope>
        </dependency>
        <!-- Bring in the Share Page Object (PO) Tests that comes with
 Alfresco. It has
             the org.alfresco.po.share.AbstractTest class that our
 custom tests extend. -->
        <dependency>
            <groupId>${alfresco.groupId}</groupId>
            <artifactId>share-po</artifactId>
            <version>${alfresco.version}</version>
            <classifier>tests</classifier>
            <scope>test</scope>

            <!-- Exclude selenium as it is already brought in by share-
po dependency above -->
            <exclusions>
                <exclusion>
                    <groupId>org.seleniumhq.selenium</groupId>
                    <artifactId>selenium-java</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>org.seleniumhq.selenium</groupId>
                    <artifactId>selenium-server</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <!-- Test NG is defined with test scope in share-po, so need it
 here too -->
        <!-- Alfresco code creates a wrapper around Test NG -->
        <dependency>
            <groupId>org.alfresco.test</groupId>
            <artifactId>alfresco-testng</artifactId>
            <version>1.1</version>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.hamcrest</groupId>
                    <artifactId>hamcrest-core</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <!-- Bring in newer selenium version if required
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>2.48.0</version>
            <scope>test</scope>
        </dependency>
        -->
        <!-- Need to bring in a newer Spring that supports annotations,
 Alfresco brings in older one -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
            <version>${spring.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aspects</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring.version}</version>
        </dependency>
    </dependencies>...
```

14. Update Virtual Webapp context for Repository (alfresco.war) in the Runner project

    Update directory paths for extra resource paths and AMP config, in the `{AIO_PROJECT_PATH}/runner/tomcat/context-repo.xml` file, update the `extraResourcePaths` and `virtualClasspath` as follows:

    ```
    ...
           extraResourcePaths="/=${project.parent.basedir}/repo-amp/target/
    amp/web" />
                   ...
           virtualClasspath="${project.parent.basedir}/repo-amp/target/
    classes;
                ${project.parent.basedir}/repo-amp/target/amp/config;
                ${project.parent.basedir}/repo-amp/target/test-classes"
                ...
    ```

    Note. if you got more Repo AMPs in your AIO project then you need to update the paths for them too.

15. Update Virtual Webapp context for Share (share.war) in the Runner project

    Update directory paths for extra resource paths and AMP config, in the `{AIO_PROJECT_PATH}/runner/tomcat/context-share.xml` file, update the `extraResourcePaths` and `virtualClasspath` as follows:

    ```
    ...
           extraResourcePaths="/=${project.parent.basedir}/share-amp/
    target/amp/web" />
                   ...
           virtualClasspath="${project.parent.basedir}/share-amp/target/
    classes;
                ${project.parent.basedir}/share-amp/target/amp/config;
                ${project.parent.basedir}/share-amp/target/test-classes;
                ${project.parent.basedir}/share/target/test-classes"
                ...
    ```

    Note. if you got more Repo AMPs in your AIO project then you need to update the paths for them too.

16. Add `maven-dependency-plugin` to the Share project

    We will need it to unpack the `MANIFEST.MF` file so we can save it and store it in the new custom WAR. Open the `{AIO_PROJECT_PATH}/share/pom.xml` file and add the plug-in just before the `maven-war-plugin`:

    ```
    <plugin>
            <!-- Bring in the Maven Dependency plugin so we can unpack and
     store the MANIFEST.MF file.
                 It will be used in the custom Share WAR that is produced
     by the WAR plugin,
                 it otherwise gets overwritten by the overlay process. -->
            <artifactId>maven-dependency-plugin</artifactId>
            <executions>
                <execution>
                    <id>unpack</id>
                    <phase>generate-sources</phase>
                    <goals>
                        <goal>unpack-dependencies</goal>
                    </goals>
                    <configuration>
                        <includeTypes>war</includeTypes>
                        <includeGroupIds>org.alfresco</includeGroupIds>
                        <includeArtifactIds>share</includeArtifactIds>
    ```

```
                    <includes>META-INF/MANIFEST.MF</includes>
                </configuration>
            </execution>
        </executions>
    </plugin>
```

17. Add `archive` section to the `maven-war-plugin` in the Share project

    Store the custom `MANIFEST.MF` file when we build the custom `share.war`. Open the `{AIO_PROJECT_PATH}/share/pom.xml` file and add the `archive` section to the `maven-war-plugin`:

```
<artifactId>maven-war-plugin</artifactId>
   <configuration>
    <!-- Bring in the MANIFEST.MF file from the original share.war, it
 contains version information
         that is needed for it to operate properly -->
    <archive>
        <addMavenDescriptor>false</addMavenDescriptor>
        <manifestFile>${project.build.directory}/dependency/META-INF/
MANIFEST.MF</manifestFile>
    </archive>
```

18. Finally remove current `alf_data_dev` directory with previous database.

    Remove the `{AIO_PROJECT_PATH}/alf_data_dev` directory. This is needed as the H2 script artifact does not currently contain upgrade scripts.

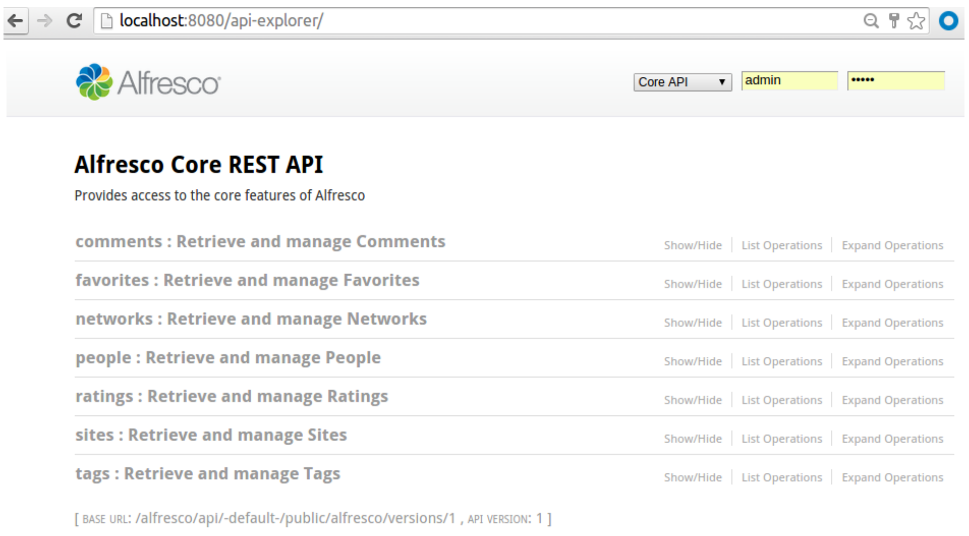
Your All-in-One project should now be fully updated to use the 2.2.0 version of the SDK.

Using the REST API Explorer

The Alfresco SDK comes with the Alfresco REST API Explorer web application built into the `run` profile for the All-In-One project.

The API Explorer is based on the OpenAPI Specification (Swagger) and provides an interactive (live) way of exploring the . When the All-In-One project is run the API Explorer is available via the `http://localhost:8080/api-explorer` URL to test the different APIs against the local running AIO instance.

The following screenshot shows an example of how the API Explorer home page looks like:



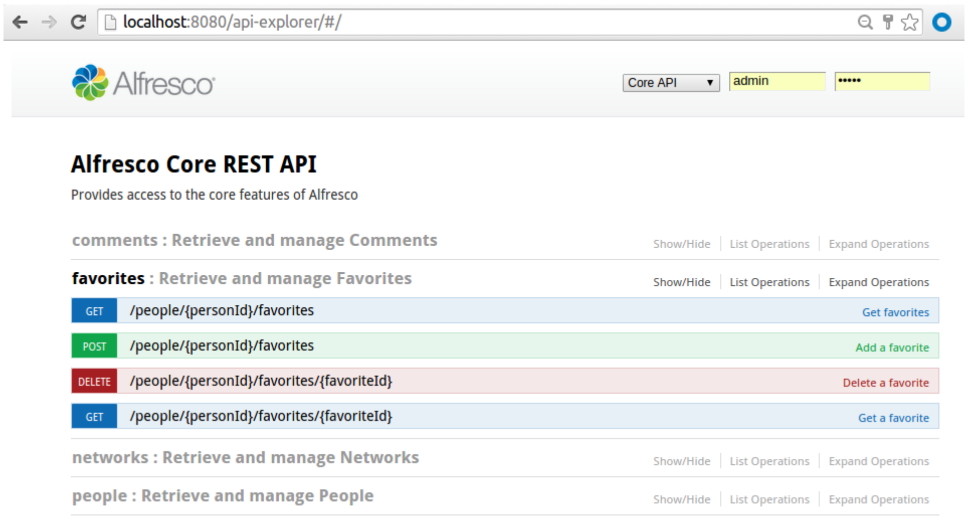
Alfresco Core REST API

Provides access to the core features of Alfresco

comments : Retrieve and manage Comments	Show/Hide	List Operations	Expand Operations
favorites : Retrieve and manage Favorites	Show/Hide	List Operations	Expand Operations
networks : Retrieve and manage Networks	Show/Hide	List Operations	Expand Operations
people : Retrieve and manage People	Show/Hide	List Operations	Expand Operations
ratings : Retrieve and manage Ratings	Show/Hide	List Operations	Expand Operations
sites : Retrieve and manage Sites	Show/Hide	List Operations	Expand Operations
tags : Retrieve and manage Tags	Show/Hide	List Operations	Expand Operations

[BASE URL: /alfresco/api/-default-/public/alfresco/versions/1 , API VERSION: 1]

To explore a particular group of APIs, such as **favorites**, click on it:



localhost:8080/api-explorer/#/

Core API | admin | •••••

Alfresco Core REST API

Provides access to the core features of Alfresco

comments : Retrieve and manage Comments — Show/Hide | List Operations | Expand Operations

favorites : Retrieve and manage Favorites — Show/Hide | List Operations | Expand Operations

GET	/people/{personId}/favorites	Get favorites
POST	/people/{personId}/favorites	Add a favorite
DELETE	/people/{personId}/favorites/{favoriteId}	Delete a favorite
GET	/people/{personId}/favorites/{favoriteId}	Get a favorite

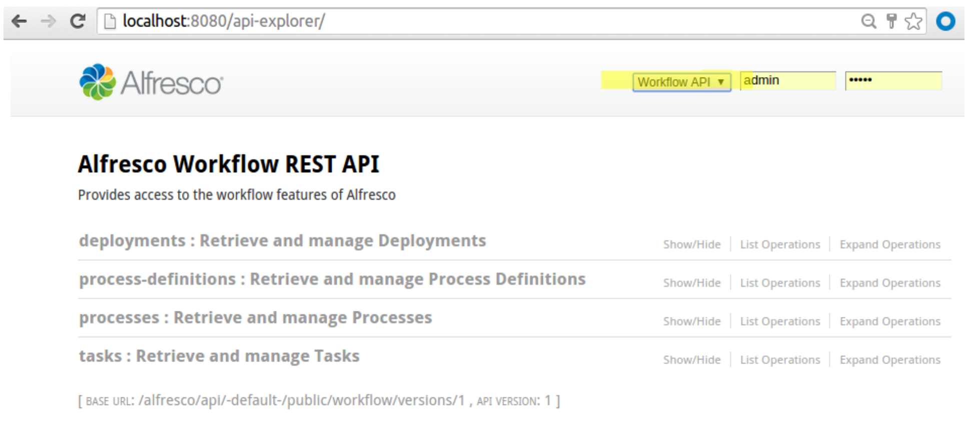
networks : Retrieve and manage Networks — Show/Hide | List Operations | Expand Operations

people : Retrieve and manage People — Show/Hide | List Operations | Expand Operations

From here you can click on each of the available operations and test them against the locally running Alfresco instance.

To run with another user than `admin` change the username and password in the upper right corner.

By default the explorer will show the Core API, if you are working with workflows click on the drop down box that says **Core API** and select the **Workflow API**, you should then see the following:



localhost:8080/api-explorer/

Workflow API | admin | •••••

Alfresco Workflow REST API

Provides access to the workflow features of Alfresco

deployments : Retrieve and manage Deployments — Show/Hide | List Operations | Expand Operations

process-definitions : Retrieve and manage Process Definitions — Show/Hide | List Operations | Expand Operations

processes : Retrieve and manage Processes — Show/Hide | List Operations | Expand Operations

tasks : Retrieve and manage Tasks — Show/Hide | List Operations | Expand Operations

[BASE URL: /alfresco/api/-default-/public/workflow/versions/1 , API VERSION: 1]

Using MySQL

The Alfresco SDK can be configured to use a MySQL database server rather than the default option of the H2 database engine.

By default, the Alfresco SDK uses H2 as its database, but it can be configured to use other databases such as MySQL, PostgreSQL, or DB2. MySQL is a commonly used open source database. The following tutorials looks at how the Alfresco SDK can be configured to use MySQL rather than H2 as the main database for Alfresco, for both Repository AMP projects and All-in-One projects.

Using MySQL with a repository AMP project

The Alfresco SDK can be configured to use MySQL, rather than the default database which is H2. The following shows how to configure a repository AMP project to use MySQL.

This tutorial assumes you have access to a suitable MySQL server, or a local installation of MySQL. Instructions on how to do this can be found in the MySQL documentation. It is also assumed that you have created a repository AMP project according to instructions found here.

You will see how to configure Alfresco SDK to use MySQL, rather than H2. This involves running a simple script in MySQL, to create the necessary database and user, and set privileges. You will also need to add some configuration to the AMP project `pom.xml` file.

1. Create a file `db_setup.sql` with the following contents:

   ```
   create database alfresco default character set utf8;
    grant all on alfresco.* to 'alfresco'@'localhost' identified by
    'alfresco' with grant option;
    grant all on alfresco.* to 'alfresco'@'localhost.localdomain'
    identified by 'alfresco' with grant option;
   ```

2. Log into your MySQL server as root using the MySQL client:

   ```
   mysql -u root -p
   ```

3. Run your script to set up the database for Alfresco:

   ```
   source db_setup.sql
   ```

   This will create the Alfresco database (alfresco) and user/pwd (alfresco/alfresco).
4. You now need to configure your project POM file. Change into your project directory and load `pom.xml` into your editor of choice.
5. Add a dependency for the MySQL JDBC driver at the `<project>` level of your `pom.xml` file:

   ```
   <dependencies>
           ...
           <dependency>
               <groupId>mysql</groupId>
               <artifactId>mysql-connector-java</artifactId>
               <version>5.1.32</version>
           </dependency>
   </dependencies>
   ```

6. Now add the configuration required for connecting to your MySQL server in the properties section of the POM:

   ```
   <!-- MySQL configuration -->
   <alfresco.db.name>alfresco</alfresco.db.name>
   <alfresco.db.username>alfresco</alfresco.db.username>
   <alfresco.db.password>alfresco</alfresco.db.password>
   <alfresco.db.host>localhost</alfresco.db.host>
   <alfresco.db.port>3306</alfresco.db.port>
   <alfresco.db.params></alfresco.db.params>
   <alfresco.db.url>jdbc:mysql://${alfresco.db.host}:${alfresco.db.port}/
   ${alfresco.db.name}</alfresco.db.url>
   <alfresco.db.datasource.class>org.gjt.mm.mysql.Driver</
   alfresco.db.datasource.class>
   ```

7. Save your changes to the `pom.xml` file.
8. Comment out the H2 Dialect configuration from `alfresco-global.properties`

   Open the `src/test/properties/local/alfresco-global.properties` configuration file. Then comment out the following line:

   ```
   #hibernate.dialect=org.hibernate.dialect.H2Dialect
   ```

9. Now in the project directory you can type:

   ```
   mvn clean install
   ```

10. Clean up any previous runs with H2:

    ```
    rm -rf alf_data_dev/
    ```

    This step is also very **important** as it will remove any content and indexes created when you started with the H2 database. When you switch over to run with MySQL the system thinks that it is the first time that you are running Alfresco, so it will create a new database, new content, and index again, which will clash with any previous starts with H2.

11. Once you have a successful build you can run up Alfresco using:

    ```
    mvn clean install -Pamp-to-war
    ```

    Alfresco will start up and use the MySQL database server that you configured. Track console messages to confirm such as:

    ```
    ...
    2014-09-15 15:47:52,552  INFO  [alfresco.repo.admin] [localhost-
    startStop-1] Using database URL 'jdbc:mysql://localhost:3306/alfresco'
     with user 'alfresco'.
    2014-09-15 15:47:52,987  INFO  [alfresco.repo.admin] [localhost-
    startStop-1] Connected to database MySQL version 5.6.11
    ...
    ```

12. Check for the message `INFO: Starting ProtocolHandler ["http-bio-8080"]`.
13. Point your web browser at `http://localhost:8080/alfresco`, and log in as `admin` with password `admin`.

You have configured the Alfresco SDK to use MySQL rather than H2.

Using MySQL with an All-in-One project

The Alfresco SDK can be configured to use MySQL, rather than the default database which is H2. The following shows how to configure an All-in-One project to use MySQL.

This tutorial assumes you have access to a suitable MySQL server, or a local installation of MySQL. Instructions on how to do this can be found in the MySQL documentation.

You will see how to configure Alfresco SDK to use MySQL, rather than H2. This involves running a simple script in MySQL, to create the necessary database and user, and set privileges. You will also need to add some configuration to the project `pom.xml` file.

1. Create a fresh All-in-One (AIO) project to work with. You can use the instructions contained in this tutorial as your guide.
2. Create a file `db_setup_aio.sql` with the following contents:

   ```
   create database alfrescoaio default character set utf8;
    grant all on alfrescoaio.* to 'alfresco'@'localhost' identified by
    'alfresco' with grant option;
    grant all on alfrescoaio.* to 'alfresco'@'localhost.localdomain'
    identified by 'alfresco' with grant option;
   ```

   ⚠ Note a different database has been specified here to avoid conflict with the previous tutorial.

3. Log into your MySQL server as root using the MySQL client:

   ```
   mysql -u root -p
   ```

4. Run your script to set up the database for Alfresco:

   ```
   source db_setup_aio.sql
   ```

   This will create the Alfresco database and user.

5. Add a dependency for the MySQL JDBC driver

   Open the `{AIO_PROJECT_ROOT}/runner/pom.xml` project file. Then add the following dependency at the end of the `tomcat7-maven-plugin` definition:

   ```
   <plugin>
       <groupId>org.apache.tomcat.maven</groupId>
       <artifactId>tomcat7-maven-plugin</artifactId>
   ...
       </configuration>
       <dependencies>
           <dependency>
               <groupId>mysql</groupId>
               <artifactId>mysql-connector-java</artifactId>
               <version>5.1.32</version>
           </dependency>
       </dependencies>
   </plugin>
   ```

6. Now add the configuration required for connecting to your MySQL server

   Open the `{AIO_PROJECT_ROOT}/pom.xml` project file. Then add the MySQL database connection properties as follows:

   ```
   <properties>
    ...
       <!-- MySQL configuration -->
       <alfresco.db.name>alfrescoaio</alfresco.db.name>
       <alfresco.db.username>alfresco</alfresco.db.username>
       <alfresco.db.password>alfresco</alfresco.db.password>
       <alfresco.db.host>localhost</alfresco.db.host>
       <alfresco.db.port>3306</alfresco.db.port>
       <alfresco.db.params></alfresco.db.params>
       <alfresco.db.url>jdbc:mysql://${alfresco.db.host}:
   ${alfresco.db.port}/${alfresco.db.name}</alfresco.db.url>
       <alfresco.db.datasource.class>org.gjt.mm.mysql.Driver</
   alfresco.db.datasource.class>
   </properties>
   ```

7. Comment out the H2 Dialect configuration from `alfresco-global.properties`

   Open the `{AIO_PROJECT_ROOT}/repo/src/main/properties/local/alfresco-global.properties` configuration file. Then comment out the following line:

   ```
   #hibernate.dialect=org.hibernate.dialect.H2Dialect
   ```

   Open the `{AIO_PROJECT_ROOT}/repo-amp/src/test/properties/local/alfresco-global.properties` configuration file. Then comment out the following line:

   ```
   #hibernate.dialect=org.hibernate.dialect.H2Dialect
   ```

8. Now in the project directory you can type:

   ```
   mvn clean install
   ```

This step is very **important** as it cleans up any previous configuration files from target/...

9. Clean up any previous runs with H2:

   ```
   rm -rf alf_data_dev/
   ```

   This step is also very **important** as it will remove any content and indexes created when you started with the H2 database. When you switch over to run with MySQL the system thinks that it is the first time that you are running Alfresco, so it will create a new database, new content, and index again, which will clash with any previous starts with H2.

10. Once you have a successful build the project you can start up Alfresco using:

    ```
    mvn clean install -Prun
    ```

    Alfresco will start up and use the MySQL database server that you configured. As before, scan the console for messages that confirm that Alfresco has connected to MySQL:

    ```
    2014-09-15 16:14:59,912  INFO  [domain.schema.SchemaBootstrap]
     [localhost-startStop-1] Connecting to database: jdbc:mysql://
    localhost:3306/alfrescoaio, UserName=alfresco@localhost, MySQL
     Connector Java
    2014-09-15 16:14:59,913  INFO  [domain.schema.SchemaBootstrap]
     [localhost-startStop-1] Schema managed by database dialect
     org.hibernate.dialect.MySQLInnoDBDialect.
    ```

11. Point your web browser at `http://localhost:8080/share`, and log in as `admin` with password `admin`.

You have configured the Alfresco SDK to use MySQL rather than H2.