Alfresco Technology Certification

™

Technical Validation Report

## Solution Overview

<solution name>

<solution summary – use case etc.>

“<solution tagline & descriptive text from website>”

Available from <solution URL>.

## Document Control

**Author:** Peter Monks ([pmonks@alfresco.com](mailto:pmonks@alfresco.com))

**Date:** <date>

**Version:** <version>

## Summary of Findings

Of the 57 technical validation criteria, XX criteria are met, YY are unmet and ZZ have yet to be validated.

Must fix:

1. xxxx

Fix, priority 2:

1. xxxx

Fix, priority 3:

1. xxxx

Fix opportunistically:

1. xxxx

Technical Validation Results

## Technical Summary

|  |  |
| --- | --- |
| **Certification type:** <Certification | Recertification>  **Version(s) reviewed:** <version>  **Compatible with:** <set of Alfresco editions>  **Min Alfresco version:** <Alfresco version>  **Max Alfresco version:** <Alfresco version>  **Build:** <build tool(s)>  **Packaging:** <AMP, ZIP, etc.>  **Repo:** <yes/no>  **Share:** <yes/no> | **Java:** XX files / YY LOC[[1]](#footnote-1)  **Javascript:** XX files / YY LOC  **Freemarker:** XX files / YY LOC  **Content models:** XX  **Spring app contexts:** XX  **Web Scripts:** XX  **Actions:** XX  **Behaviours:** XX  **Quartz jobs:** XX |

## API Usage

### API01 – Only the Public Alfresco Java APIs may be used

**Description:** Only the public [Alfresco Java APIs](http://docs.alfresco.com/4.2/concepts/java-public-api-list.html)[[2]](#footnote-2) may be used in a certified extension.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### API02 – Use Alfresco browser Javascript APIs whenever possible

**Description:** Alfresco provides a [browser-side Javascript API](http://sharextras.org/jsdoc/share/community-head/)[[3]](#footnote-3) that should be used whenever possible. Extending Share with additional Javascript libraries is error prone and risks upgrade issues later on.

**Applies to:** Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### API03 – Extend Alfresco.component.Base, Alfresco.ConsoleTool, Alfresco.ConsolePanelHandler, etc.

**Description:** Alfresco’s base classes provide useful fields and methods for custom browser-side Javascript classes.

**Applies to:** Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### API04 – Extend Alfresco browser components using YAHOO.extend(), YAHOO.lang.augmentProto() or YAHOO.lang.augmentObject()

**Description:** This is the standard extension approach for custom browser-side Share components. These methods should be called within the constructor of the custom class.

**Applies to:** Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### API05 – Inject ServiceRegistry, not individual \*Service beans

**Description:** There are a number of reasons why custom extension code should inject the ServiceRegistry in preference to individual \*Service beans. First of all, only a small subset of the Spring beans defined by Alfresco are Public i.e. available for use by custom 3rd party code. Unfortunately Spring doesn’t provide a way to differentiate between those that are available for use / extension and those that aren’t, and the public-services-context.xml file that Alfresco provides is incomplete (not all public services are defined in that file). The ServiceRegistry, on the other hand, contains getter methods for most of the Public services, and most (though not all) of the services available from the ServiceRegistry are Public and may be used by extensions – this will improve over time e.g. via the use of a new “AlfrescoPublicApi” annotation that will be introduced in Alfresco v4.3.

Another reason for using the ServiceRegistry is that customers are able to wire in an alternative implementation of the ServiceRegistry that disables security and auditing (amongst other things), and if your extension uses the ServiceRegistry bean exclusively it will be automatically compatible with this mode of operation. Injecting individual beans makes it impractical for customers to run in this mode if they’ve installed your extension.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 3**

**Evidence:** <Supporting evidence>

**Notes:**

### API06 – Prefer dependency injection over the service locator pattern

**Description:** In Alfresco extensions, the dependency injection pattern is preferred to service locator pattern (see [this article](http://martinfowler.com/articles/injection.html)[[4]](#footnote-4) for an overview of both mechanisms). This is because the set of Spring beans defined by Alfresco that are Public and hence available for use in extensions is not documented, so it is difficult to ensure your extension isn’t using service location to access a private bean.  
Note that the different types of dependency injection (constructor vs setter) are not significant from a certification perspective – this is mostly a matter of style and preference, and therefore left to the implementers’ discretion.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 3**

**Evidence:** <Supporting evidence>

**Notes:**

## Change Management

### CM01 – Use AMPs for final packaging

**Description:** AMPs are the only deployment mechanism that is well supported across Alfresco’s entire supported platform matrix. Other deployment mechanisms (uncontrolled modification of the WAR file, use of tomcat/shared/lib, installation of files into the exploded webapp) are fragile and unsupported, and most don’t work on anything but Tomcat.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

## Development

### DEV01 – Artifacts repository is used for builds

**Description:** Alfresco has deprecated the legacy “SDK Download” in favour of the [Alfresco Maven artifacts repository](https://artifacts.alfresco.com/)[[5]](#footnote-5). At some point in the not-too-distant future the SDK will no longer be available, and any extensions that use it will be forced to move over to the artifacts repository. Extension developers are strongly encouraged to move across to the artifacts repository now, while the option of a smooth, unforced migration exists.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix opportunistically**

**Evidence:** <Supporting evidence>

**Notes:**

### DEV02 – Prefer Javascript to Java for Web Script controller logic

**Description:** Alfresco is able to “sandbox” server-side Javascript logic to a far greater extent than is possible with Java code. For this reason, as much logic as feasible should be implemented in Javascript, to minimise the risks should an extension behave unexpectedly.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix opportunistically**

**Evidence:** <Supporting evidence>

**Notes:**

## Compatibility

### COM01 – Use unique Java packages (i.e. based on a domain name)

**Description:** When Java classnames collide the behaviour at runtime can be unpredictable (which version of two colliding classes is loaded depends on the JVM, app server, and a number of other factors). For compatibility (both with Alfresco and other extensions), an extension must use one or more unique packages, ideally based on the domain name of the developing organisation.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### COM02 – Use namespaces for custom browser-side JS objects

**Description:** Similar to COM01, the same issue can surface in the browser.

**Applies to:** Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### COM03 – Module identifiers must be namespaced and immutable

**Description:** Module ids are used by Alfresco to uniquely and unambiguously identify modules, including multiple different versions of the same module. For this reason module ids must be namespaced (to minimise the chance of collisions) and immutable – they must not change over time. In particular this means that module identifiers must not include build numbers or platform or environmental information, as this breaks Alfresco’s ability to uniquely and unambiguously identify the module.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 3**

**Evidence:** <Supporting evidence>

**Notes:**

### COM04 – Prefer the use of repository actions for implementing functionality

**Description:** [Repository actions](http://wiki.alfresco.com/wiki/Custom_Actions)[[6]](#footnote-6) (Alfresco’s implementation of the [GoF Command Pattern](http://en.wikipedia.org/wiki/Command_pattern)[[7]](#footnote-7)) are automatically enhanced to support invocation from rules, from server-side Javascript scripts and from within the workflow engine (amongst others). It is therefore advantageous to implement as much extension logic as possible as repository actions, so that that logic can be readily reused by customers or other implementers who’ve installed the extension.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix opportunistically**

**Evidence:** <Supporting evidence>

**Notes:**

### COM05 – Use custom <config> section inside a share-config-custom.xml for Share-side configuration settings. Don't use .properties files.

**Description:** The Share paradigm is to store configuration in custom <config> blocks within a share-config-custom.xml file, rather than in properties files. If this is done, a lot of the work of loading that configuration and making it available to controller logic is handled for you automatically by Share. With properties files, you’re on your own.

**Applies to:** Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix opportunistically**

**Evidence:** <Supporting evidence>

**Notes:**

### COM06 – Code is compiled for Java 1.6+ (<= Ent 4.1) or Java 1.7+ (>= Ent 4.2)

**Description:** Alfresco has not supported Java 1.5 for some time. No extension logic should be compiled for that version of Java any more.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 3**

**Evidence:** <Supporting evidence>

**Notes:**

### COM07 – Extension is compatible with Alfresco Supported Platforms Matrix (SPM)

**Description:** Extensions should be compatible with the entire [Alfresco SPM](http://www.alfresco.com/services/subscription/supported-platforms)[[8]](#footnote-8) – those that aren’t run the risk of behaving unpredictably and/or damaging some customer installations of Alfresco.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### COM08 – Ensure content model namespace prefixes are globally unique

**Description:** Unlike most other namespace aliasing mechanisms (e.g. XML), Alfresco content model namespace prefixes (“d”, “sys”, “cm”, etc.) are global – they are defined once and exist in that form throughout an Alfresco installation. As a result, collisions between namespaces prefixes have a high probability if they’re not thought through properly. The recommendation is to resist the temptation to use short (2 or 3 letter) prefixes, and instead use a more lengthy prefix that combines a few letters that are unique to your organisation and a few letters that are unique to the content model.

Note: content model prefixes are not persisted by the repository (they’re stored in memory only), so it is safe to change them at any time.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### COM09 – Prefer CMIS query language when using SearchService

**Description:** Alfresco’s SearchService API supports different “languages” (XPath, Lucene, SOLR and CMIS), which roughly correlate to the different underlying search implementations in Alfresco. Of these languages, only CMIS is fully abstracted away from the underlying implementation, and so is the only language that provides some guarantee of consistent behaviour, regardless of how a given Alfresco instance has been configured (SOLR vs Lucene, or MDQ, for example).  
Note however that SearchService doesn’t fully implement CMIS-QL – specifically, the “SELECT” clause in CMIS queries sent to the SearchService are not processed (they are silently ignored). SearchService, regardless of the query language used, always returns sets of matching NodeRefs.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix opportunistically**

**Evidence:** <Supporting evidence>

**Notes:**

### COM10 – Use a unique namespace prefix for Spring bean ids

**Description:** Spring bean ids are global, and must therefore be prefixed with some kind of unique prefix.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### COM11 – Use a unique namespace prefix for AttributeService keys

**Description:** AttributeService attributes created and read by an extension must be namespaced with some kind of unique prefix. The best practice for doing this involves using the 3 keys that the AttributeService supports in the following manner:

1. Key 1 should be a unique, global namespace for your organisation and technology, represented as a String (e.g. “com.acme.technologyname”).
2. Key 2 should be the name of a single attribute used in your technology, represented as a String (e.g. “myExternalSystemDocId”).
3. Key 3 should represent a single identifier (key) for that attribute, represented as any Serializable value (e.g. “abcd1234”).
4. The value should store that value for that key, represented as any Serializable value (e.g. “workspace://SpacesStore/f6184f65-b9a7-4ea4-7631-07cd73a988d6”). This is often a NodeRef, though it can be anything.

If you are working with composite keys, you should concatenate the individual key values with a delimiter (e.g. “key1value-key2value-key3value”), and store the result in key #3. You should pick a delimiter that works best for your key values, but having a delimiter of some kind is critical in order to avoid ambiguous concatenations (e.g. undelimited concatenation of “ab” & “cd” has the same result as undelimited concatenation of “a” & “bcd”).

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

## Performance

### PERF01 – Content policies / behaviours must be fast or asynchronous

**Description:** Content policies are wired in at a very low level in the repository and as a result can be called many hundreds of times a second in some cases (e.g. when content is being manipulated via CIFS). In addition they are executed synchronously within each Alfresco transaction. The result is that even the slightest poor performance in a custom content policy or behaviour can have a profound impact on Alfresco performance. For that reason it is critically important that custom content policies / behaviours are either fast (conduct minimal computation and only perform minimal I/O to the repository) or are made asynchronous.

Note that as at the time of writing, Alfresco doesn’t provide any direct support for asynchronous policies, so some kind of custom approach (e.g. using java.util.concurrent) is required.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### PERF02 – Be judicious in the use of indexed properties in content models

**Description:** Every additional indexed property slows down writes to the nodes that contain that property, so while there’s a temptation to naively configure every single property in a content model as indexed, “just in case”, this can have a significant impact on the performance of the repository and the size of the indexes on disk. The recommendation here is index just the bare minimum number of properties that are required for querying, also taking into account that the [transactional metadata query mechanism](http://docs.alfresco.com/4.2/index.jsp?topic=%2Fcom.alfresco.enterprise.doc%2Fconcepts%2Fintrans-metadata.html)[[9]](#footnote-9) (aka MDQ) doesn’t require that the properties being queries are marked as indexed.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 3**

**Evidence:** <Supporting evidence>

**Notes:**

### PERF03 – Don't store property values in search engine indexes

**Description:** The <stored>true</stored> setting for a property in an Alfresco content model does not provide any advantage to either Alfresco or an extension. Furthermore, it bloats the search engine indexes, resulting in less efficient indexing and querying operations. There is no reason to ever set this setting to true.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

## Security

### SEC01 – Don't use the "lower case" versions of the Alfresco service beans

**Description:** The “lower case” versions of the Alfresco service beans (i.e. those whose first name starts with a lowercase letter e.g. nodeService) are configured to bypass Alfresco’s security, transaction and auditing checks, with no recourse for the administrator to turn them back on. There have been persistent (but incorrect) rumours in the Alfresco community that these versions of the services perform significantly better than the official (“upper case”) versions, but that hasn’t been the case since at least Alfresco v2.x.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### SEC02 – Minimise manual control of authenticated session (e.g. AuthenticationUtil.runAs\*)

**Description:** Manual control of the authenticated session is effectively a controlled spoofing attack within Alfresco, and it’s difficult to prove that an extension that does this isn’t opening up a subtle security hole in Alfresco. For that reason use of these mechanisms must be minimised.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### SEC03 – Avoid "none" authentication in Web Scripts

**Description:** “None” authentication in a Web Script means that anyone with network access to the Alfresco server can invoke that Web Script, without having to provide any kind of authentication credentials whatsoever. Such Web Scripts are therefore inherently insecure, and if combined with masquerading APIs (see SEC02) can create serious security holes in Alfresco.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### SEC04 – Minimise use of Process.exec() / ProcessBuilder

**Description:** Forking external OS processes should not be undertaken lightly as doing so is both a resource intensive operation and a potential security hole (Alfresco has no ability to control what those processes do).

Note that this is legitimate in certain specific cases (e.g. when leveraging OS tools for document transformation or metadata extraction).

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### SEC05 – Don't use Javascript eval() (e.g. when handling JSON)

**Description:** Javascript’s “eval” function is a significant source of security holes in Javascript code, both on the server and in the browser. Alfresco provides both server- and client- side APIs for performing common operations (such as parsing JSON documents) that don’t require the use of eval.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

## Stability

### STB01 – Don't override or mask core files

**Description:** Overriding a core file immediately places the Alfresco instance into an unsupported configuration and creates compatibility and upgrade problems. This includes both explicit replacement (e.g. directly replacing a core file using an AMP) as well as “masking” (whereby a file inside a JAR inside an AMP “masks” a core Alfresco file).

There are rare cases (particularly in Surf/Share) where overriding a core file is the only way to achieve a particular requirement. In this case this should be considered an Alfresco bug, a support and/or JIRA ticket should be raised and the Alfresco Certification Team notified so that they may expedite it with Alfresco engineering.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### STB02 – Don't replace or add new versions of Alfresco provided JARs

**Description:** Alfresco has been QAed with a precise set of 3rd party JARs. Replacing or upgrading those JARs immediately places the Alfresco instance into an unsupported configuration.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### STB03 – Don't use servlets or servlet filters

**Description:** Adding a servlet or filter to Alfresco requires replacing the web.xml file, which (as a core file) violates STB01.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### STB04 – Don't access (for read or write) the Alfresco database directly

**Description:** Alfresco’s database schema and the SQL the product uses has been carefully tuned. Uncontrolled access to the same tables can interfere with Alfresco’s normal operation, impacting both performance and (in some cases) stability. Note that this includes reads (SELECTs), as this can block concurrent write operations in some circumstances.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### STB05 – Don't modify or extend the Alfresco DB schema - if you need your own tables, put them in your own schema

**Description:** Adding tables to the Alfresco schema violates STB04 and introduces upgrade risk. There are legitimate reasons for requiring custom database tables as part of an extension, but in this case they should be stored in a separate database schema.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### STB06 – Use RetryingTransactionHelper for all manually defined transactions, NOT TransactionService

**Description:** As the name implies, RetryingTransactionHelper contains retry logic for certain recoverable, expected database exceptions (deadlocks, basically). It also uses Spring’s “template” pattern to ensure a transaction is always completed (committed or rolled back) correctly, regardless of what happens in the logic inside the transaction.

The “raw” TransactionService provides neither of these benefits and for that reason is considered unsafe for use in extensions.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### STB07 – Close all resources (e.g. search results)

**Description:** Certain resources in Alfresco (specifically search result objects) are not cleaned up automatically by Alfresco and must therefore be cleaned up correctly by extension code (i.e. in a “finally” block). Failing to clean up such resources results in leaks, not only of memory but also, in some cases, “real” operating system resources (such as file handles) as well.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### STB08 – Do not synchronise on the Alfresco APIs - they are thread safe

**Description:** The Alfresco APIs are thread safe. Synchronising on them is not only unnecessary, it can also lead to performance problems and even deadlock.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### STB09 – Don't excessively synchronise in custom code

**Description:** Most extensions only mutate data in the repository (whose APIs are already thread safe – see STB08), which makes synchronisation unnecessary in many cases.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### STB10 – Don't create or manage threads

**Description:** It is generally safer and easier to let Alfresco manage execution on its own various thread pools. There are cases where threads make sense (e.g. PERF01), however they should be used judiciously.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### STB11 – Proper exception handling

**Description:** Exceptions should either be caught and recovered from, or allowed to flow up the call stack. It is almost never appropriate to “swallow” an exception (catch it and do nothing), and excessive wrapping of exceptions inside other exceptions should be minimised (it makes triage more difficult).

Catching or throwing java.lang.Error instances, and catching java.lang.Throwables is also inappropriate – these classes (java.lang.Error, specifically) indicate fatal JVM problems and therefore cannot be safely caught or thrown.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### STB12 – Use logging properly

**Description:** All output should be logged using Alfresco’s standard logging library (Apache Commons Logging, at the time of writing). Methods that emit output to stdout or stderr (including System.out, System.err, Throwable.printStrackTrace etc.) must not be used under any circumstances – in addition to making the administrator’s life more difficult, these methods are also synchronised on some platforms, which can cause performance problems.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### STB13 – Always have a timeout on all RPCs

**Description:** Remote procedure calls (including REST, SOAP, RMI, etc. calls) must always have a timeout, to handle cases where the call doesn’t complete in a “reasonable” amount of time. RPCs are blocking, and tie up resources in the Alfresco server – RPCs that take significant amounts of time can lead to resource pressure and/or starvation if not timed out appropriately.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### STB14 – Avoid executing searches in components that run during bootstrap

**Description:** The infrastructure behind the SearchService API is not yet initialised when custom code is bootstrapped. Use of the SearchService API at this time will result in unexpected behaviour that may prevent Alfresco from starting up.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### STB15 – Load content models via bootstrap, don't place models in the Data Dictionary

**Description:** The Data Dictionary’s models subdirectory is reserved for end-customer use. Custom extensions must not use it themselves.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### STB16 – Load web scripts via bootstrap, don't place webscripts in the Data Dictionary

**Description:** The Data Dictionary’s Web Script Extension subdirectory is reserved for end-customer use. Custom extensions must not use it themselves.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### STB17 – Test against latest version of Alfresco Enterprise

**Description:** Extensions being considered for certification must always be tested against the latest available version of Alfresco Enterprise, at a minimum.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### STB18 – Prefer Alfresco-managed transactions

**Description:** The repository manages transactions around many of Alfresco’s extension points, including actions & rules, Web Scripts etc. This transaction management code has been exhaustively QAed by Alfresco across the SPM and is preferable to manually managed transactions (i.e. RetryingTransactionHelper). Leverage this capability wherever possible.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 3**

**Evidence:** <Supporting evidence>

**Notes:**

### STB19 – Avoid “none” transaction setting in Web Scripts

**Description:** Turning off transactions for Web Scripts is never appropriate if that Web Script makes use of any Alfresco repository services. There are rare cases where this is appropriate, when used in combination with the RetryingTransactionHelper (see STB06).

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### STB20 – Leave Web Script transaction setting out of Web Script descriptors (i.e. at the default)

**Description:** The Web Script framework has a good set of defaults for the <transaction> setting. It is rare that the defaults for this setting need to be overridden.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix opportunistically**

**Evidence:** <Supporting evidence>

**Notes:**

### STB21 – Don’t redefine / augment Alfresco Spring beans

**Description:** The Spring beans that Alfresco defines are used internally by the Alfresco system itself. Redefining (e.g. with custom implementations) and/or augmenting them (e.g. via AspectJ interceptors, proxies etc.) can have unintended side effects throughout the system (not to mention conflict with other extensions).

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### STB22 – Minimise use of ThreadLocals

**Description:** It is generally safer and easier to let Alfresco manage execution on its own various thread pools. There are cases where threads make sense (e.g. PERF01), however they should be used judiciously.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

## Usability

### UX01 – Code must not assume search / query is transactionally consistent

**Description:** With the exception of the new (as of 4.2) [transactional metadata query mechanism](http://docs.alfresco.com/4.2/index.jsp?topic=%2Fcom.alfresco.enterprise.doc%2Fconcepts%2Fintrans-metadata.html)[[10]](#footnote-10), none of Alfresco’s search or query APIs can be assumed to be transactionally consistent. SOLR-based installations are always eventually consistent for both metadata and content, and Lucene-based installations may be configured into this mode of operation as well. Custom logic that assumes transactional consistency of query / search (i.e. the SearchService API) will not function correctly, typically manifesting as unexpected UI behaviour.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

## Upgrade

### UP01 – Don't extend the legacy Explorer UI

**Description:** The legacy Explorer UI is no longer being actively developed by Alfresco and will eventually be end-of-lifed. As a result, extensions that integrate with the Explorer UI are ineligible for certification.

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### UP02 – Don't parse NodeRefs, StoreRefs, etc.

**Description:** Although NodeRefs, StoreRefs etc. currently have a substructure, this structure is an implementation detail that must not be parsed or processed by custom extensions – NodeRefs, StoreRefs etc. must be treated as an opaque value. Alfresco reserves the right to change the substructure of NodeRefs, StoreRefs etc. at any time, without warning.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### UP03 – Set module.repo.version.min and module.repo.version.max in module.properties

**Description:** Alfresco’s API compatibility policy only extends within a single minor version of Alfresco Enterprise i.e. we guarantee that APIs will not change in service packs or hotfixes. While Alfresco tries not to change APIs in minor or major versions of Alfresco Enterprise, such changes are sometimes necessary and Alfresco reserves the right to make such changes if no reasonable backwards-compatible alternative exists.

As a result of this policy, extensions (AMPs) should be tested on all new minor and major versions of Alfresco Enterprise, and must include the module.repo.version.min and module.repo.version.max properties set to those tested versions.

AMPs that don’t set these properties run are almost certain to break customers’ Alfresco installations during future upgrades.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix, priority 2**

**Evidence:** <Supporting evidence>

**Notes:**

### UP04 – Specify module.editions in module.properties

**Description:** AMP files can optionally specify which edition(s) of Alfresco they are compatible with. Extensions being considered for certification should set this property to “Enterprise” only (and not “Community”).

**Applies to:** Repo

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Fix opportunistically**

**Evidence:** <Supporting evidence>

**Notes:**

## Legal

### LGL01 – Avoid “unfriendly” licenses

**Description:** Alfresco is unable to certify technologies that are provided under an “unfriendly” license. This list includes the various so-called viral licenses (e.g. GPL, AGPL, Creative Commons) as well as licenses that include terms that Alfresco is unable to meet (e.g. the [ExtJS license](http://www.exttld.com/index.php?content=terms)[[11]](#footnote-11)).

Most traditional commercial licenses, as well as the “liberal” open source licenses (e.g. Apache, MIT, EPL) are   
suitable.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

### LGL02 – Don’t embed libraries or software that are provided under an “unfriendly” license

**Description:** Alfresco is unable to certify technologies that include 3rd party libraries or software that are provided under an “unfriendly” license. See [LGL01](#_LGL01_–_Avoid) for more details on what constitutes an “unfriendly” license.

**Applies to:** Repo, Share

**Result:** **Meets****Does not meet**

**Remedial Action:** **None****Must fix**

**Evidence:** <Supporting evidence>

**Notes:**

1. LOCs are calculated using the [Ohloh Line Count](https://github.com/blackducksw/ohcount) tool. It ignores whitespace and comments but is otherwise naïve. [↑](#footnote-ref-1)
2. <http://docs.alfresco.com/4.2/concepts/java-public-api-list.html> [↑](#footnote-ref-2)
3. <http://sharextras.org/jsdoc/share/community-head/> [↑](#footnote-ref-3)
4. <http://martinfowler.com/articles/injection.html> [↑](#footnote-ref-4)
5. <https://artifacts.alfresco.com/> [↑](#footnote-ref-5)
6. <http://wiki.alfresco.com/wiki/Custom_Actions> [↑](#footnote-ref-6)
7. <http://en.wikipedia.org/wiki/Command_pattern> [↑](#footnote-ref-7)
8. <http://www.alfresco.com/services/subscription/supported-platforms> [↑](#footnote-ref-8)
9. <http://docs.alfresco.com/4.2/index.jsp?topic=%2Fcom.alfresco.enterprise.doc%2Fconcepts%2Fintrans-metadata.html> [↑](#footnote-ref-9)
10. <http://docs.alfresco.com/4.2/index.jsp?topic=%2Fcom.alfresco.enterprise.doc%2Fconcepts%2Fintrans-metadata.html> [↑](#footnote-ref-10)
11. <http://www.exttld.com/index.php?content=terms> [↑](#footnote-ref-11)