JOSEPH N. AKROTIRIANAKIS (Bar No. 197971)
jakro@kslaw.com
KING & SPALDING LLP
633 West Fifth Street, Suite 1600
Los Angeles, CA 90071
Telephone: 213-443-4355
Facsimile: 213-443-4310

CHRISTOPHER C. CAMPBELL (*application for pro hac vice admission to be filed*)
ccampbell@kslaw.com
KING & SPALDING LLP
1700 Pennsylvania Avenue, NW, Suite 200
Washington, DC 20006-4707
Telephone: 202-626-5578
Facsimile: 202-626-3737

BRITTON F. DAVIS (*application for pro hac vice admission to be filed*)
bfdavis@kslaw.com
BRIAN EUTERMOSER (*application for pro hac vice admission to be filed*)
beutermoser@kslaw.com
PETER SAUER (*application for pro hac vice admission to be filed*)
psauer@kslaw.com
KING & SPALDING LLP
1401 Lawrence Street
Suite 1900
Denver, CO 80202
Telephone: 720-535-2300
Facsimile: 720-535-2400

Attorneys for Plaintiffs OPEN TEXT CORPORATION, OPEN TEXT SA ULC, and OPEN TEXT HOLDINGS INC.

UNITED STATES DISTRICT COURT

CENTRAL DISTRICT OF CALIFORNIA

SOUTHERN DIVISION

OPEN TEXT CORPORATION, OPEN TEXT SA ULC, and OPEN TEXT HOLDINGS INC., Plaintiffs, v. HYLAND SOFTWARE, INC., Defendant.	Case No. ________ **COMPLAINT FOR PATENT INFRINGEMENT** JURY TRIAL DEMANDED

Plaintiffs Open Text Corporation, Open Text SA ULC, and Open Text Holdings Inc., (collectively "Plaintiffs") allege against Defendant Hyland Software, Inc. ("Hyland" or "Defendant") as follows:

1. OpenText Corporation provides information management solutions that allow companies to organize and manage content, operate more efficiently, increase engagement with customers, collaborate with business partners, and address regulatory and business requirements.

2. OpenText Corporation provides such solutions by distributing software products and providing customer support and professional services through a number of subsidiaries, including Open Text, Inc., which sells OpenText software and services in the United States.

3. The OpenText family of companies (collectively "OpenText") has approximately 15,000 employees, more than 74,000 customers, and over $3.11 billion in annual revenues. OpenText invested approximately $1 billion on research and development over the three years ending June 30, 2020.

4. Gartner's Magic Quadrant report for 2019, published October 30, 2019, named OpenText a "Leader" in Content Services Platforms. And Gartner's 2019 Market Share Analysis, published July 24, 2020, ranked OpenText one of the "Top Five Content Services Providers, Worldwide" in 2019; in particular, OpenText was ranked first for "Content Services Platforms."

5. OpenText currently maintains three offices in the State of California, one of which is located in this judicial district, including the Pasadena office at 1055 E. Colorado Blvd., Pasadena, California 91106-2375.

6. OpenText tracks its business through four revenue streams: license, customer support, cloud services, and professional services. (Exhibit A at 9-10 (Aug. 6, 2020 10-K).) OpenText receives license revenue from its software products; customer support revenue from renewable support and maintenance OpenText provides to customers who have purchased its products; cloud services revenue from certain

"managed hosting" services arrangements; and professional services revenue from consulting fees OpenText collects for providing implementation, training, and integration services related to OpenText's product offerings.

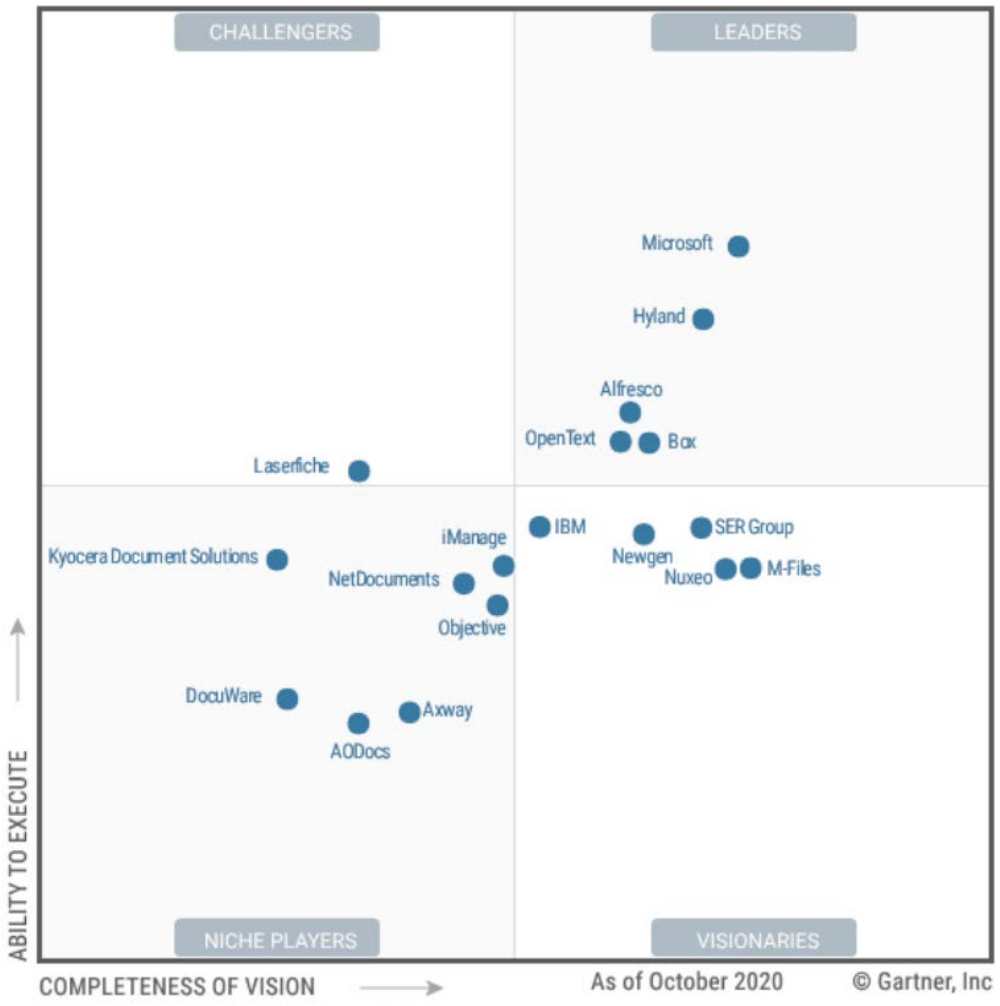
7. On or about September 9, 2020, Hyland entered into an agreement to acquire another of OpenText's competitors, Alfresco. (Exhibit B (2020.09.09 - Hyland enters definitive agreement to acquire Alfresco, hyland.com).) According to the press release, Alfresco is "the leading open source content services and solutions provider for information-rich enterprises with huge volumes of unstructured content." (Exhibit B (2020.09.09 - Hyland enters definitive agreement to acquire Alfresco, hyland.com).)

8. On or about October 22, 2020, Hyland's acquisition of Alfresco was completed. (Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com).) According to the Alfresco press release, "[t]he acquisition [of Alfresco] furthers Hyland's vision to become the world's leading content services provider, expanding its global footprint with additional customers, partners and employees with extensive industry experience." (Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com).) Further, the press release states that "[t]he addition of Alfresco's solutions augments Hyland's range of content services offerings and provides new opportunities to engage with the open-source community for product innovation." (Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com).)

9. On December 2, 2020, Hyland announced that "Hyland and its new acquisition, Alfresco were both named Leaders in the Gartner 2020 Magic Quadrant for Content Services Platforms." (Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com).) As shown in the Gartner 2020 Magic Quadrant for Content Services Platforms report, both Alfresco and Hyland compete directly with OpenText and the combination of Alfresco and Hyland represents a clear and emergent competitive threat to OpenText's business, perpetuated by infringement of OpenText's intellectual property by Alfresco and Hyland:

Magic Quadrant

Figure 1: Magic Quadrant for Content Services Platforms

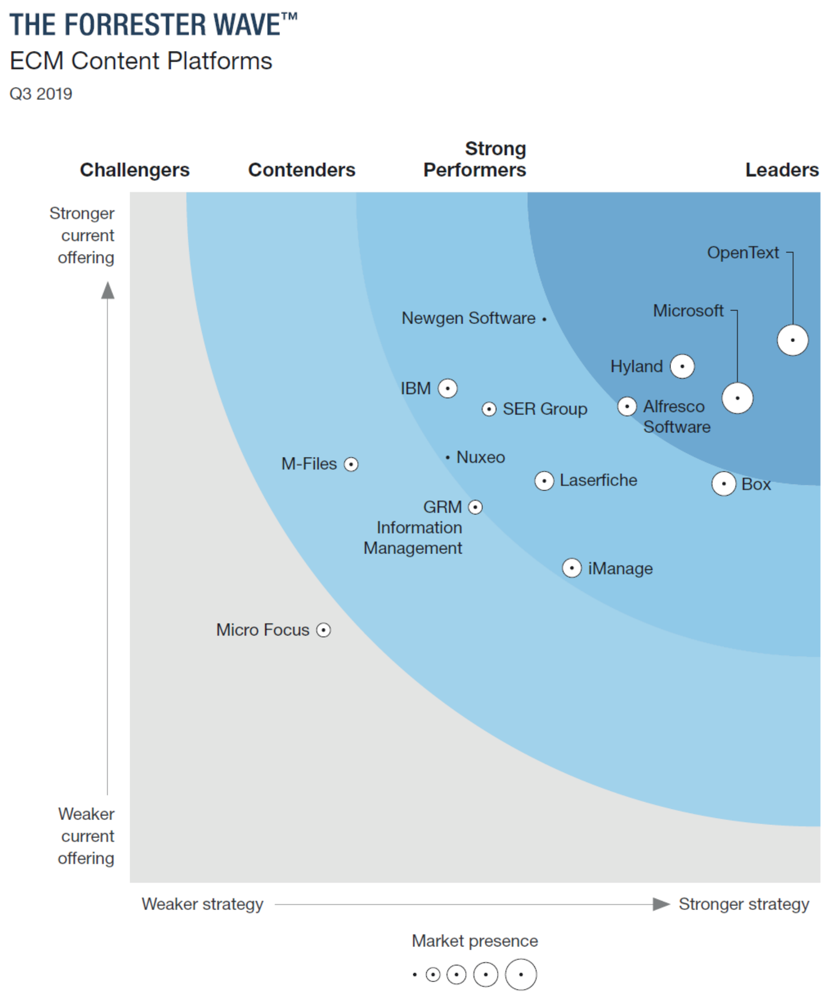


Source: Gartner (November 2020)

(Exhibit E at 3 (2020.11.16 - Gartner Content Services Report 2020).)

10. Hyland competes directly with OpenText in the Enterprise Content Management (ECM) and Enterprise Information Management (EIM) markets, as well as related products and services, by its manufacture, use, sale, and offer for sale of the Alfresco ECM platform (such as the Alfresco Content Services) and integrated

applications and features such as Alfresco Transformation Services and Application Development Framework, which infringe OpenText's intellectual property rights. (Exhibit F [Forrester Wave™- ECM Content Platforms, Q3 2019_24july2019] at 4.)



11. Plaintiffs bring this lawsuit to protect its intellectual property investments and to hold Hyland accountable for its infringement. As a result of Hyland's unlawful competition in this judicial district and elsewhere in the United States, Plaintiffs have lost sales and profits and suffered irreparable harm, including lost market share and goodwill.

NATURE OF THE CASE

12. Plaintiffs bring claims under the patent laws of the United States, 35 U.S.C. § 1, *et seq.*, for the infringement of United States Patent Nos. 9,047,146;

8,380,830; 9,813,381; 9,170,786; 10,540,150; and 9,189,761 (collectively, the "Patents-in-Suit").

THE PARTIES

13. Plaintiff OpenText Corporation is a Canadian corporation with its principal place of business at 275 Frank Tompa Drive, Waterloo, Ontario, Canada.

14. Plaintiff Open Text SA ULC is a Canadian corporation with its principal place of business at 1959 Upper Water St., Halifax, Nova Scotia, Canada.

15. Plaintiff Open Text Holdings Inc., is a Delaware corporation with its principal place of business at 275 Frank Tompa Drive, Waterloo, Ontario, Canada.

16. Defendant Hyland Software, Inc. is a corporation with its global headquarters at 28500 Clemens Road, Westlake, Ohio 44145, with multiple other offices within the U.S. and elsewhere, including an office in this District in Irvine, California, located at 2355 Main Street, Suite 100, Irvine, California 92614.

JURISDICTION & VENUE

17. This action arises under the Patent Laws of the United States, 35 U.S.C. § 1 *et seq*. The Court has subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

18. This Court has personal jurisdiction over Hyland because it regularly conducts business in the State of California and in this district, including operating systems and/or providing services in California and in this district that infringe one or more claims of the Patents-in-Suit in this forum. Hyland has, either directly or through intermediaries, purposefully and voluntarily placed its infringing products and/or services into the stream of commerce with the intention and expectation that they will be purchased and used by customers in this District, as detailed below.

19. Venue is proper in this judicial district pursuant to 28 U.S.C. §§ 1391(b) and (c) and 28 U.S.C. § 1400(b) because, upon information and belief, Hyland regularly conducts business within this District, has a regular and established place of business in this District, and has committed acts of infringement within this District. In addition,

on information and belief, as a foreign corporation with sufficient contacts with this District, venue is proper against Hyland in this District.

20. Hyland Software, Inc. is a registered business in California. (Exhibit N.)

21. On information and belief, Hyland has a regular and established place of business at 2355 Main Street, Suite 100, Irvine, California 92614. (Exhibit O.)

22. On information and belief, Hyland has employees in this district, including at least 60 employees at Hyland's Irvine, California location.

23. Hyland sells and/or offers for sale its infringing ECM platform, software and services, including Alfresco Content Services, Alfresco Transformation Services, and Alfresco Development Framework, as well as related products and modules (hereinafter "Accused Products"), through its websites - https://docs.alfresco.com/content-services/6.1/, https://docs.alfresco.com/transform-service/latest/, and https://www.alfresco.com/ecm-software/application-development-framework, which may be accessed by customers within this district.

24. On information and belief, Hyland sells and/or offers for sale the Accused Products and related software and services to customers located in this district.

25. As further detailed below, Hyland's use, provision of, installation, configuration, offer for sale, sales, and advertising of the Accused Products within this judicial district infringe the Patents-in-Suit. Hyland's customers infringe the Patents-in-Suit at least by using the Accused Products within this judicial district.

26. Because Hyland actively targets customers served by OpenText and the OpenText office in Pasadena, California, Hyland's infringement adversely impacts the over 187 OpenText employees who live and work in and around this judicial district.

THE PATENTS-IN-SUIT

U.S. Patent Nos. 9,047,146 ('146 Patent) and 8,380,830 ('830 Patent)

27. The '146 and '830 Patents are part of the same patent family and are generally directed to "system[s] and method[s] for processing an input data stream in a first data format of a plurality of first data formats to an output data stream in a second

data format of a plurality of second data formats." ('146 Patent, Abstract.) Plaintiff Open Text SA ULC owns by assignment the entire right, title, and interest in and to the '146 and '830 Patents.

28. The '146 Patent is entitled "Method and system for transforming input data streams," was filed on January 18, 2013, claims priority to an application filed on June 28, 2002, and was duly and legally issued by the USPTO on June 2, 2015. A true and correct copy of the '146 Patent is attached as Exhibit G.

29. The '830 Patent, also entitled "Method and system for transforming input data streams," was filed on April 22, 2011, claims priority to an application filed on June 28, 2002, and was duly and legally issued by the USPTO on February 19, 2013. A true and correct copy of the '830 Patent is attached as Exhibit H.

30. At the time of the priority date of the '146 and '830 patents, businesses communications were becoming more complex, with different companies and business partners communicating using different means and different document formats, including electronic documents (such as .pdfs, .text, .html). ('146 Patent, 30-45.) The inventors recognized the problems associated with having multiple different document formats that needed to be compatible with different software programs and had to be compiled into usable formats for various purposes. However, given the volume of data and the various sources of the documents, such transformation could be time and resource intensive. Thus, the inventors discovered specific technical improvements to improve the operation and efficiency of the transformation process.

31. For example, in some embodiments, an agent can scan the input stream corresponding to a file from a business application and identify fields in the input data stream. (*See* '146 Patent, 4:51-5:45 and 7:47-58.) Fields from the event can be placed in a message structured according to a message tree built using the fields, blocks and variables. (*See* '146 Patent, 4:51-5:45). This can be repeated for each event identified in the input data stream. A process corresponding to the event can be applied to transform the messages containing the text from the events and structured according to

the generic data structure to produce an output. (*See* ’146 Patent, 4:51-5:45 and 7:47-58.) For example, a process can be applied to transform messages containing text from the events into “a document for printing, faxing, .pdf, web, etc.” (*See* ’146 Patent, 6:61-7:3.)

32. The inventors found that this event driven approach, using a processing thread, enabled the efficient detection of patterns in the input files. For example, the inventors found that by identifying events, and then creating messages using a generic structure for each event, each thread could process part of the process to transform the input document into a different format. Moreover, each thread can be processed in parallel with other threads. In this way, embodiments disclosed and claimed provided advantages over conventional techniques, including increase in performance and providing support for parallel job execution. This system architecture also offers better scalability for multi-processor systems. All threads are connected to queues and/or connectors, enabling extremely flexible configuration. Several job threads can serve one or several queues and several input connectors can use one or several queues and job threads. (’146 Patent, 3:32-38.) Accordingly, the specific technical solution described and claimed in the ‘830 and ‘146 Patents also improves the functionality of existing computer systems.

U.S. Patent No. 9,813,381

33. The ’381 Patent is entitled “Flexible and secure transformation of data using stream pipes,” was filed on May 1, 2015, claims priority to a provisional application filed on June 18, 2014, and was duly and legally issued by the USPTO on November 7, 2017. Plaintiff Open Text SA ULC owns by assignment the entire right, title, and interest in and to the ’381 Patent. A true and correct copy of the ’381 Patent is attached as Exhibit I.

34. The ’381 patent is generally directed to systems and methods that provide “a transformation pipeline [that] may be created to efficiently transform file data one unit at a time in memory.” (’381, Abstract.) In some embodiments, a process uses read

and write methods to move the unit of data into and out of processing streams and calls the appropriate transformation engine(s), and transformation. For example, the write method may move a unit of data, for instance, from a memory buffer into an associated stream. The read method may read the unit of data from the stream, call an associated transformation, and pass the unit of data thus transformed to the next stream or a destination. This process is repeated until all desired and/ or required transformations such as compression, encryption, tamper protection, conversion, etc. are applied to the unit of data. ('381, Abstract.)

35. The '381 Patent describes and claims inventive and patentable subject matter that significantly improves on traditional data management and data processing systems. The '381 patent was developed in the context that "an increasing amount of data is stored and communicated in electronic format" and "in many cases, data may exist only in electronic form, making access and security considerations for such data important—inasmuch as the data may not be readily accessed or protected in any other manner." ('381 patent, 1:28-32) Electronic data tends to have the following characteristics, including volume, variety, velocity, variability, veracity, and complexity. ('381 patent, 2:1-22.) The inventors recognized that "[i]n view of the growth trend toward increasingly large and complex data sets, conventional data management and data processing systems and methods are strained and, in some cases, unequal to the task. Challenges include analysis, capture, curation, search, sharing, security, storage, transfer, visualization, and information privacy." ('381 patent, 1:63-2:1.)

36. The '381 Patent provided technical improvements over conventional data management and data processing systems and methods by solving technical problems experienced by these systems and methods. For example, embodiments of the claimed invention in the '381 patent can achieve flexible and secure transformations of streamed data without requiring streamed data to be written to interim persistent storage. ('381 patent, 1:17-24)

37. In some embodiments, in response to a request to read or write a file, a transformation pipeline is created that "allows read or write methods to be called on each stream such that the results of each read operation from each stream class is passed as input to the next stream class in the transformation pipeline." ('381 patent, 8:46-65.) The transformation pipeline can be created by, e.g., "instantiating a stream object for each stream class of the multiple stream classes," and "[e]ach stream object may include a write method for moving a unit of data into the associated stream and a read method for retrieving the unit of data from the associated stream, calling an associated transformation function (e.g., compression, encryption, tamper protection, conversion, encoding, transcoding, etc.), and providing the unit of data thus transformed within the associated stream to the next stream or, if no more transformation streams in the transformation pipeline, to a destination device." *Id.*

38. One technical benefit of the specific technical solutions described and claimed in the '381 patent is "the ability to engage flexible and secure data stream processing with substantially reduced persistent data storage requirements." ('381 patent, 10:30-38) In modern computing devices, "accessing persistent storage is typically the most time-intensive operation." Therefore, technical solutions in the '381 Patent can realize "a substantial reduction in the amount of time required to engage, process and manage secure data communication" compared with conventional systems. *Id.*

39. Thus, the '381 Patent describes and claims systems and methods that provide technical advantages and improvements over traditional conventional data management and data processing systems and methods, including the ability to achieve flexible and secure transformations of streamed data without requiring streamed data to be written to interim persistent storage.

U.S. Patent Nos. 9,170,786 and 10,540,150

40. The '786 and '150 Patents are part of the same patent family and are generally directed to "developer-composed context menus, e.g., composed by a

developer in connection with use of a software development tool to create an application." Plaintiff OpenText Corporation owns by assignment the entire right, title, and interest in and to the '786 and '150 Patents.

41. The '786 Patent is entitled "Composable Context Menus," was filed on December 20, 2013, and was duly and legally issued by the USPTO on October 27, 2015. A true and correct copy of the '786 Patent is attached as Exhibit J.

42. The '150 Patent, also entitled "Composable Context Menus," was filed on September 1, 2015, and was duly and legally issued by the USPTO on January 21, 2020. The '150 Patent claims priority to the '786 Patent. A true and correct copy of the '150 Patent is attached as Exhibit K.

43. The '150 and '786 Patents describe and claim inventive and patentable subject matter that significantly improves on traditional application development tools used to build context menus for software applications, as well as the graphical user interface ("UI") provided to a user. "'Context' or "'contextual menus' enable a user-selectable set of contextually-relevant options to be displayed in an application or other user interface." ('786 Patent, 1:11-13.) For example, "if a user enters a 'right click' or other prescribed input while a mouse of other cursor is "hovering over an object displayed on an application page, a context menu comprising a list of actions considered to be potentially desired to be performed by the user with respect to the hovered-over object may be displayed. The set of options may be determined at least in part by application context data." ('786 Patent, 1:13-121.)

44. The '150 and '786 Patents provided technical improvements over conventional application development tools by solving technical problems experienced by application UI developers and improving the development tools as well as the UI. Unlike conventional application development tools, which limited developers' ability to "define context menus to a predefined set," the '150 and '786 Patents describe and claim a development framework that enables developers to create dynamic context menus whose visual features, display data, responsive actions, etc. can be updated or

changed during application execution. (Exhibit J, '786 Patent, 1:22-28, 2:21-45; Exhibit K, '150 Patent, 1:30-36, 2:27-54.). Rather than embedding pre-fabricated and static context menus into their applications, the application development framework enabled by these patents permits developers to create dynamic context menus whose features can be filled in, and swapped out, at runtime, i.e. during the execution of the application.

45. The inventors found that dynamic context menus provided a number of technical advantages. For example, enabling the context menu to update during execution of the application page enabled more flexibility in providing context specific displays and actions, thereby providing additional functionality to a user and performance of the application. Whereas fixed context menus would only be able to provide a set of functions that would have to be applicable across any application, thereby limiting the functionality to generic and widely applicable options, dynamic context menus provided flexibility to customize the displayed data and corresponding actions to the particular context of not only the application, but the specific portion of the application the user was interacting with at the time. This greatly improved the functionality of the interface.

46. Embodiments also disclose and claim the use of "invisible objects," such as an invisible "container" for the context menu. As explained above, the inventors found that these invisible objects enable the menu to be dynamically "updated during execution of the application page" and have both visual features and display data that can be changed "at runtime, *e.g*. at context menu display time," on the basis of what the application is doing or what the user does. (*See* Exhibit J, '786 Patent, 3:58-4:9; Exhibit K, '150 Patent, 3:66-4:17). Moreover, these invisible objects provide additional potential advantages (including an improved UI), such as providing more efficient UI (e.g., by not obscuring other content on the interface for the application), containing the data and actions in an efficient container, and providing customizable and/or dynamic content menu options.

47. Thus, the ’786 and ’150 Patents describe and claim systems and methods that provide technical advantages and improvements over traditional application development tools, including the ability to generate highly dynamic context menus whose features and display data can be swapped in and out during application execution through the use of “invisible” objects.

48. Applicant further explained during prosecution that the creation of an “invisible object” that “provides, to the context menu, information with which the context menu is updated during execution of the application page” is an unconventional step in the area of application development frameworks and entirely absent from the prior art. (See Exhibit M, March 16, 2015 Applicant Remarks, at 7 (emphasis added)).

49. In response to that argument, the Examiner withdrew a rejection based on 35 U.S.C. §101 and §102 and allowed the patent to issue. As recognized by the USPTO Examiner, the claimed inventions of the ’786 and ’150 Patents provide a technical solution to the technical problem of generating dynamic context menus whose features can be swapped in, and swapped out, during application execution.

U.S. Patent No. 9,189,761

50. The ’381 Patent is entitled “Action flow client framework,” was filed on May 17, 2012, and was duly and legally issued by the USPTO on November 17, 2015. Plaintiff Open Text Corporation owns by assignment the entire right, title, and interest in and to the ’761 Patent. A true and correct copy of the ’761 Patent is attached as Exhibit L.

51. The ’381 patent is generally directed to systems and methods to implementing interface control(s) associated with declaratively defining an action flow are provided; the action flow includes a desired outcome of an action flow. Information associated with a user interface page is received; this information includes a state during which the user interface page is displayed. Information associated with a business service associated with a content management server is received; that information includes a state during which the business service is performed on the content

management server. In the action flow definition, a first association between the user interface page and the state during which the user interface page is displayed and a second association between the business service and the state during which the business service is invoked on the content management server are recorded.

52. The ’761 Patent describes and claims inventive and patentable subject matter that significantly improves on traditional content management applications. The Inventors of the ‘761 Patent recognized that it is desirable for content management applications of a company “to have channels by which information can be exchanged with people who are not employees of [the] company, for example customers of a bank who want to apply for a loan.” (‘761 patent, 1:24-34) The inventors also recognized that it is further desirable for content management applications to allow customers to create an action flow, such as a web-based loan application, which “offers advantages during the design phase (e.g., when a loan application is created or updated) and/or at run time (e.g., when an applicant accesses a loan application).” *Id.*

53. The ‘761 Patent provided specific technical solutions to the above technical problems by employing action flows that are agnostic with respect to user interface technology, thereby permitting “a variety of technologies to be installed on client device,” such as “Sencha Ext JS, jQuery, and/or YUI.” (‘761 patent, 5:20-27.) Thus, embodiments provide for improved user interfaces as well as improved design tools for providing that interface.

54. For example, embodiments of the ‘761 Patent employ declaratively-defined action flows to realize an improved graphical user interface for customers, such as loan applicants. A declaratively-defined action flow includes “a desired outcome of an action flow but does not (for example) include an executable step associated with achieving the desired outcome.” (‘761 patent, 4:43-67.) Instead, the “desired progression or sequence of states and/or actions in an action flow may be defined without limiting it or tying it to a specific underlying set of instructions, or being hard coded to a specific programming language or technology.” *Id.* As compared with

conventional content management systems, this approach "permits non-technical users with industry-specific or company-specific expertise to construct an action flow without requiring extensive technical knowledge." ('761 patent, 4:43-67; 10:45-61.) In addition, "update of and/or modification to an existing action flow is made easier" as compared with systems that "have actions flows that are 'hard coded' or tightly coupled to a specific implementation." Id.

55. Embodiments of the invention described and claimed in the '761 patent also allow the client device to perform at least part of the action flow, which in turn can produce "better user experience (e.g., not having to reload an entire page as a result of having to ask a server for instruction) and/or have better performance (e.g., not affected by a slow network connection and/or an overloaded server). ('761 patent, 3:1-6; 5:1-19).

56. Thus, the '381 Patent describes and claims systems and methods that provide technical advantages and improvements over traditional content management systems and methods, including the ability to declaratively define action flows that are agnostic with respect to user interface technology.

ACCUSED PRODUCTS

57. As set forth in more detail below, Hyland's ECM Platform, and related software and services, including Alfresco Content Services, Alfresco Transformation Services, and Alfresco Development Framework, (https://docs.alfresco.com/content-services/6.1/, https://docs.alfresco.com/transform-service/latest/, and https://www.alfresco.com/ecm-software/application-development-framework) provide platforms for enterprises and their users to store, manage, capture, and access content.

58. The Accused Products also include, without limitation, systems and software, and components thereof, that may operate at least in on-premise, mobile device, or cloud environments.

59. On information and belief, each of these implementations, whether accessed via computer or mobile device, operate similarly for purposes of determining infringement.

60. Plaintiffs informed Defendant of their infringement by letter dated September 2, 2022, but they continued to make, use, sell, offer to sell, and/or import into the United States the Counterclaim Accused Products, and to induce others to do so.

FIRST CAUSE OF ACTION

(INFRINGEMENT OF THE '146 PATENT)

61. Plaintiffs reallege and incorporate the preceding paragraphs of this complaint.

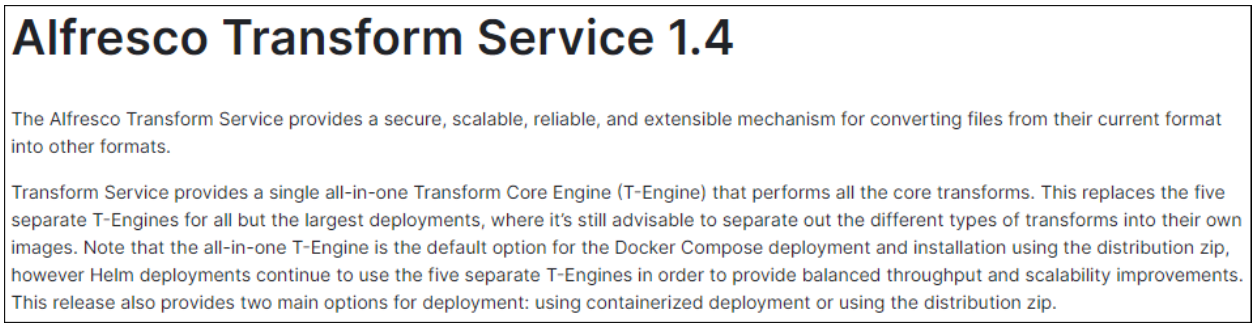
62. Defendants have infringed and continue to infringe one or more claims of the '146 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including features of the Alfresco Enterprise Content Management System (ECM), such as Alfresco Transform Service, as well as any other products that utilize of interface with Alfresco Transform Service, at least when used for their ordinary and customary purposes, practice each element of at least claim 15 of the '146 Patent as described below.

63. For example, claim 15 of the '146 patent recites:

> 15. A method for processing a data stream in a network environment, comprising:
>
> receiving an electronic input data stream of file data at a physical input over a network, wherein input data in the input data stream is of a first document format;
>
> in a same thread:

> detecting patterns in an input file of the input data stream to identify events;
>
> creating a message for each identified event containing text from the event according to a generic data structure corresponding to the event;
>
> executing a process configured to create output data of a second format from the messages, the output data created from a processed message containing text from the processed message, the output data in a different format from the first document format; and
>
> sending an output data stream to a destination, the output data stream comprising the output data.

64. The Accused Products perform at least the method of claim 15 of the '146 Patent. To the extent the preamble is construed to be limiting, the Accused Products perform *a method for processing a data stream in a network environment*, as further explained below. For example, the Alfresco Transform Service converts files "from their current format into other formats" in a network environment.


Alfresco Transform Service 1.4

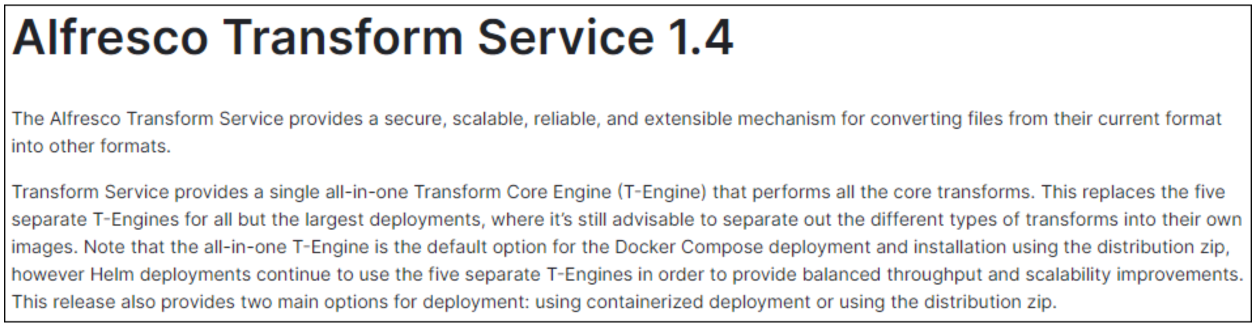
The Alfresco Transform Service provides a secure, scalable, reliable, and extensible mechanism for converting files from their current format into other formats.

Transform Service provides a single all-in-one Transform Core Engine (T-Engine) that performs all the core transforms. This replaces the five separate T-Engines for all but the largest deployments, where it's still advisable to separate out the different types of transforms into their own images. Note that the all-in-one T-Engine is the default option for the Docker Compose deployment and installation using the distribution zip, however Helm deployments continue to use the five separate T-Engines in order to provide balanced throughput and scalability improvements. This release also provides two main options for deployment: using containerized deployment or using the distribution zip.

(*See* https://docs.alfresco.com/transform-service/1.4/.)

65. The Accused Products perform a method that further includes *receiving an electronic input data stream of file data at a physical input over a network, wherein input data in the input data stream is of a first document format*. For example, the Alfresco Transform Service includes T-Engine/Transform Engines which "transforms files referenced by the repository and retrieved from the shared file store." The shared file store "is used as temporary storage for the original source files (stored by the

repository).”

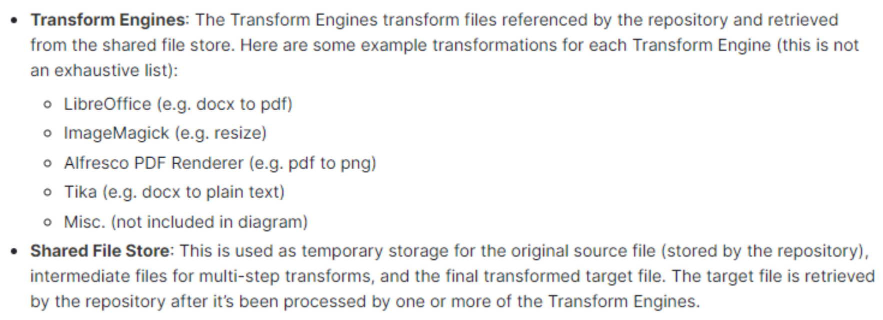


Alfresco Transform Service 1.4

The Alfresco Transform Service provides a secure, scalable, reliable, and extensible mechanism for converting files from their current format into other formats.

Transform Service provides a single all-in-one Transform Core Engine (T-Engine) that performs all the core transforms. This replaces the five separate T-Engines for all but the largest deployments, where it's still advisable to separate out the different types of transforms into their own images. Note that the all-in-one T-Engine is the default option for the Docker Compose deployment and installation using the distribution zip, however Helm deployments continue to use the five separate T-Engines in order to provide balanced throughput and scalability improvements. This release also provides two main options for deployment: using containerized deployment or using the distribution zip.

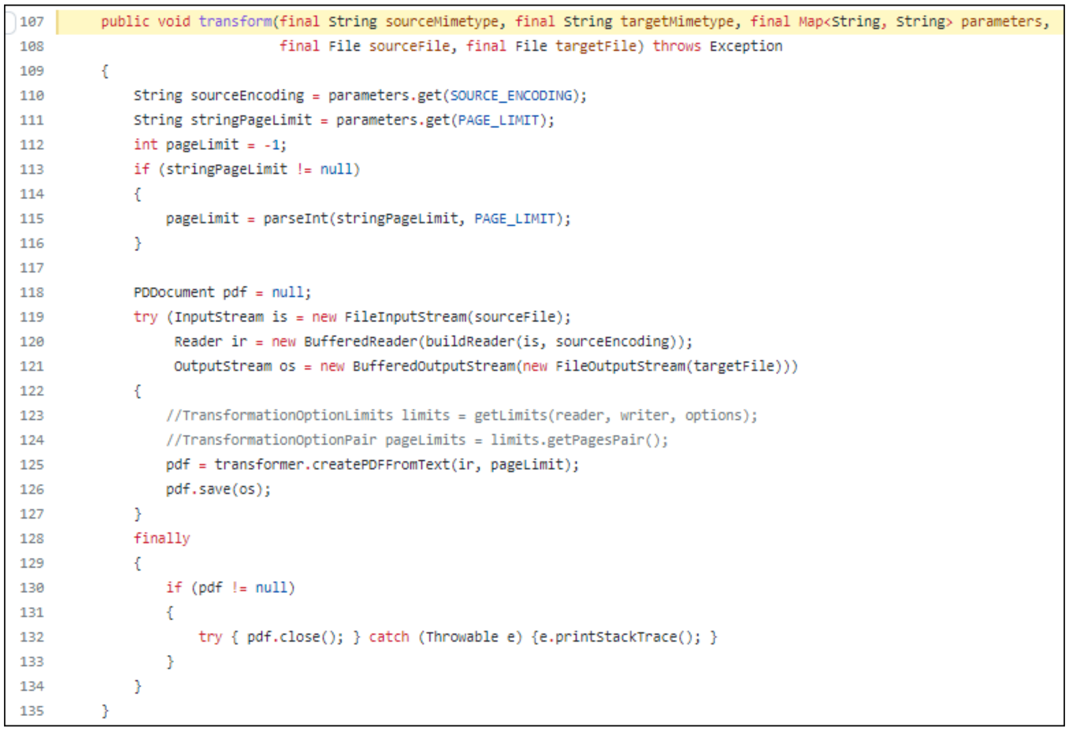
(*See* https://docs.alfresco.com/transform-service/1.4/.)



- **Transform Engines**: The Transform Engines transform files referenced by the repository and retrieved from the shared file store. Here are some example transformations for each Transform Engine (this is not an exhaustive list):
  - LibreOffice (e.g. docx to pdf)
  - ImageMagick (e.g. resize)
  - Alfresco PDF Renderer (e.g. pdf to png)
  - Tika (e.g. docx to plain text)
  - Misc. (not included in diagram)
- **Shared File Store**: This is used as temporary storage for the original source file (stored by the repository), intermediate files for multi-step transforms, and the final transformed target file. The target file is retrieved by the repository after it's been processed by one or more of the Transform Engines.

(*See* https://docs.alfresco.com/transform-service/1.4/admin/)

66. As an example, the Alfresco Transform Service receives a text file “sourceFile” from “FileInputStream” and transforms the text file into a PDF file.

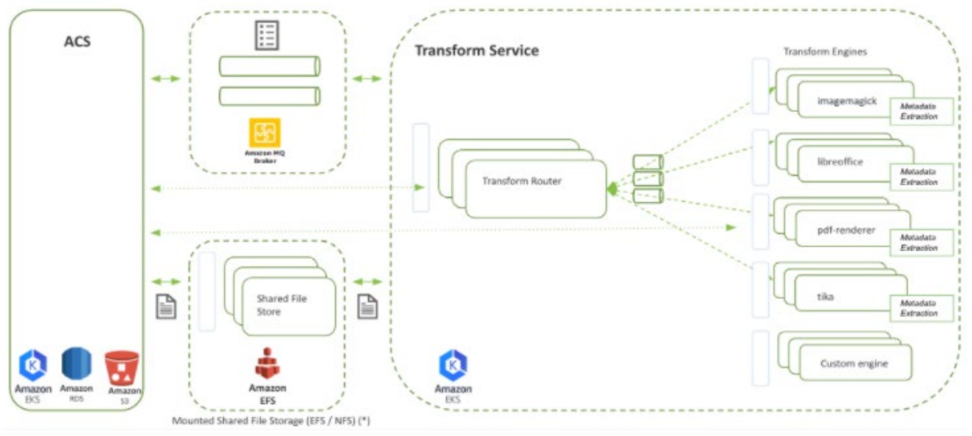


```
    public void transform(final String sourceMimetype, final String targetMimetype, final Map<String, String> parameters,
                          final File sourceFile, final File targetFile) throws Exception
    {
        String sourceEncoding = parameters.get(SOURCE_ENCODING);
        String stringPageLimit = parameters.get(PAGE_LIMIT);
        int pageLimit = -1;
        if (stringPageLimit != null)
        {
            pageLimit = parseInt(stringPageLimit, PAGE_LIMIT);
        }

        PDDocument pdf = null;
        try (InputStream is = new FileInputStream(sourceFile);
             Reader ir = new BufferedReader(buildReader(is, sourceEncoding));
             OutputStream os = new BufferedOutputStream(new FileOutputStream(targetFile)))
        {
            //TransformationOptionLimits limits = getLimits(reader, writer, options);
            //TransformationOptionPair pageLimits = limits.getPagesPair();
            pdf = transformer.createPDFFromText(ir, pageLimit);
            pdf.save(os);
        }
        finally
        {
            if (pdf != null)
            {
                try { pdf.close(); } catch (Throwable e) {e.printStackTrace(); }
            }
        }
    }
```

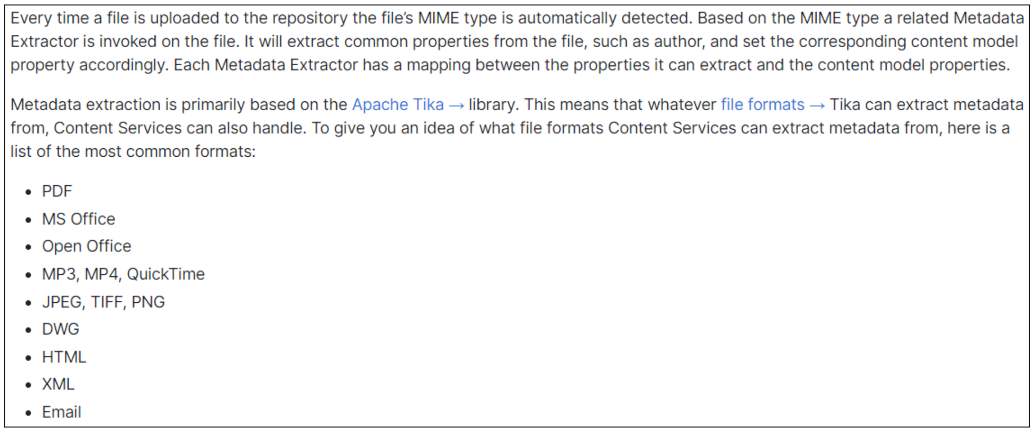
(*See* https://github.com/Alfresco/alfresco-transform-core/blob/e575ec943a5fa5dddca5593e6795a17a2bbb3cb6/alfresco-transform-misc/alfresco-transform-misc/src/main/java/org/alfresco/transformer/transformers/TextToPdfContentTransformer.java)

67. The Accused Products perform a method that further includes *in a same thread: detecting patterns in an input file of the input data stream to identify events*. The Alfresco Transform Service includes a transform router, transform engines and metadata extraction. This metadata extraction is performed in a Transform Engine (e.g., a "T-engine"). Further, a Metadata Extractor is invoked on a file (e.g., uploaded to the repository) to extract properties from the files (such as the author).

The following diagram shows a simple representation of the Transform Service components:



(*See* https://docs.alfresco.com/transform-service/latest/admin/)

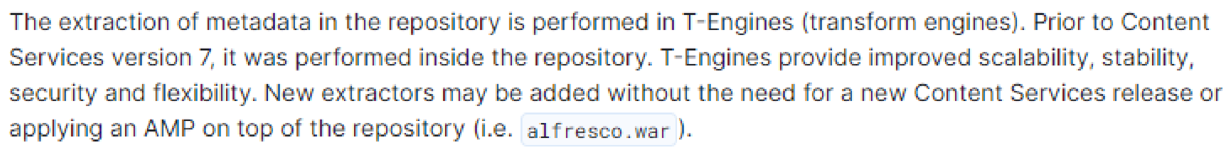


Every time a file is uploaded to the repository the file's MIME type is automatically detected. Based on the MIME type a related Metadata Extractor is invoked on the file. It will extract common properties from the file, such as author, and set the corresponding content model property accordingly. Each Metadata Extractor has a mapping between the properties it can extract and the content model properties.

Metadata extraction is primarily based on the Apache Tika → library. This means that whatever file formats → Tika can extract metadata from, Content Services can also handle. To give you an idea of what file formats Content Services can extract metadata from, here is a list of the most common formats:

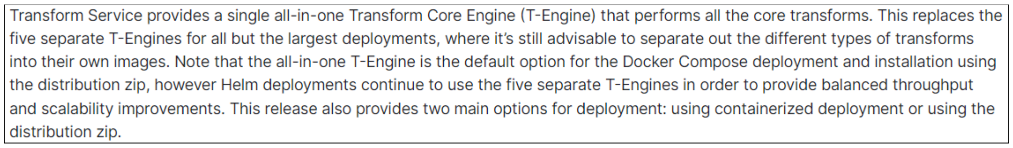
- PDF
- MS Office
- Open Office
- MP3, MP4, QuickTime
- JPEG, TIFF, PNG
- DWG
- HTML
- XML
- Email

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)



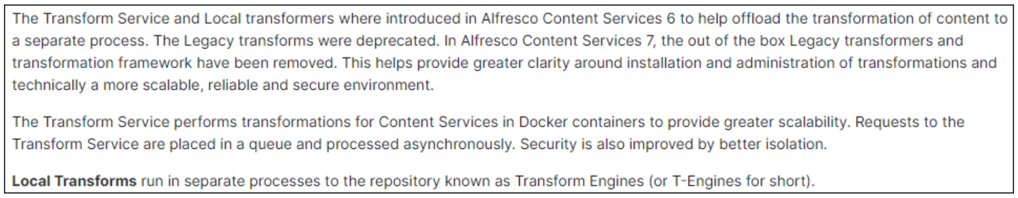
The extraction of metadata in the repository is performed in T-Engines (transform engines). Prior to Content Services version 7, it was performed inside the repository. T-Engines provide improved scalability, stability, security and flexibility. New extractors may be added without the need for a new Content Services release or applying an AMP on top of the repository (i.e. `alfresco.war`).

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)



Transform Service provides a single all-in-one Transform Core Engine (T-Engine) that performs all the core transforms. This replaces the five separate T-Engines for all but the largest deployments, where it's still advisable to separate out the different types of transforms into their own images. Note that the all-in-one T-Engine is the default option for the Docker Compose deployment and installation using the distribution zip, however Helm deployments continue to use the five separate T-Engines in order to provide balanced throughput and scalability improvements. This release also provides two main options for deployment: using containerized deployment or using the distribution zip.

(*See* https://docs.alfresco.com/transform-service/1.4/)



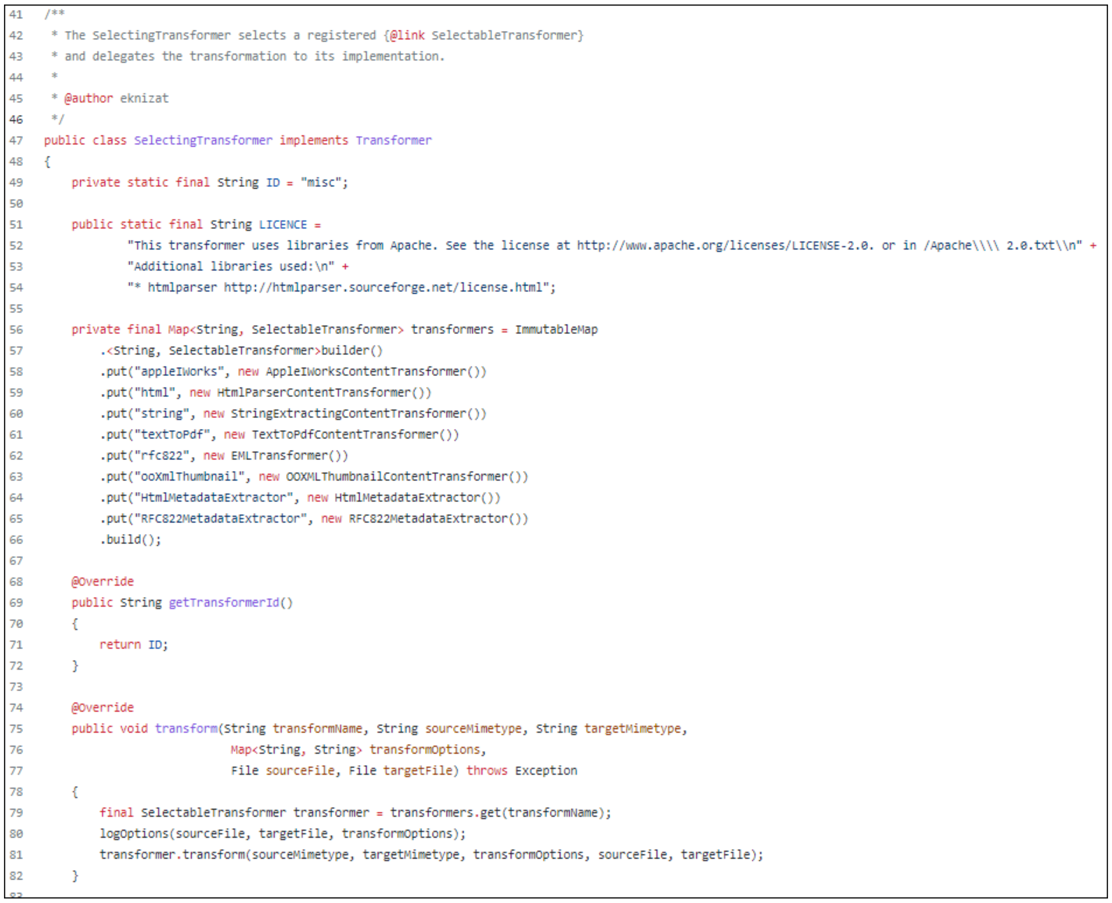
The Transform Service and Local transformers where introduced in Alfresco Content Services 6 to help offload the transformation of content to a separate process. The Legacy transforms were deprecated. In Alfresco Content Services 7, the out of the box Legacy transformers and transformation framework have been removed. This helps provide greater clarity around installation and administration of transformations and technically a more scalable, reliable and secure environment.

The Transform Service performs transformations for Content Services in Docker containers to provide greater scalability. Requests to the Transform Service are placed in a queue and processed asynchronously. Security is also improved by better isolation.

Local Transforms run in separate processes to the repository known as Transform Engines (or T-Engines for short).

(*See* https://docs.alfresco.com/transform-service/1.4/config/)

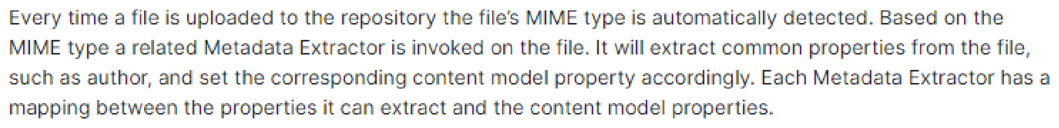
68. In addition, the Alfresco Transform Service defines a class called "SelectingTransformer" to detect input file formats and select a registered transform engine to implement the content transformation. Within the "SelectingTransformer" class, the Alfresco Transform Service also performs metadata extraction.



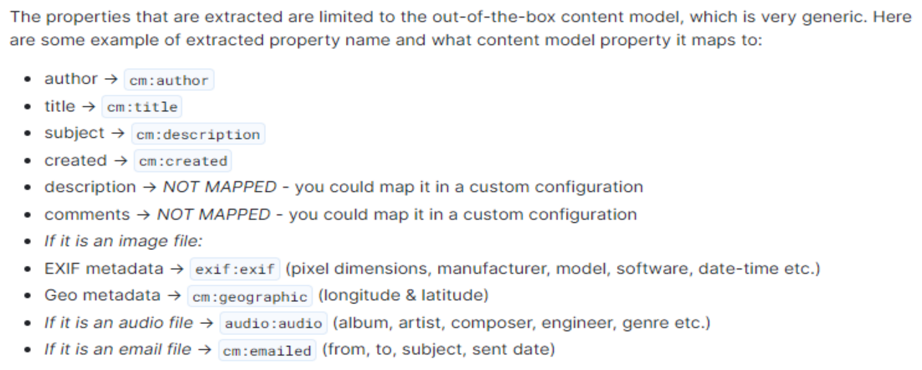
```java
/**
 * The SelectingTransformer selects a registered {@link SelectableTransformer}
 * and delegates the transformation to its implementation.
 *
 * @author eknizat
 */
public class SelectingTransformer implements Transformer
{
    private static final String ID = "misc";

    public static final String LICENCE =
            "This transformer uses libraries from Apache. See the license at http://www.apache.org/licenses/LICENSE-2.0. or in /Apache\\\\ 2.0.txt\\n" +
            "Additional libraries used:\n" +
            "* htmlparser http://htmlparser.sourceforge.net/license.html";

    private final Map<String, SelectableTransformer> transformers = ImmutableMap
        .<String, SelectableTransformer>builder()
        .put("appleIWorks", new AppleIWorksContentTransformer())
        .put("html", new HtmlParserContentTransformer())
        .put("string", new StringExtractingContentTransformer())
        .put("textToPdf", new TextToPdfContentTransformer())
        .put("rfc822", new EMLTransformer())
        .put("ooXmlThumbnail", new OOXMLThumbnailContentTransformer())
        .put("HtmlMetadataExtractor", new HtmlMetadataExtractor())
        .put("RFC822MetadataExtractor", new RFC822MetadataExtractor())
        .build();

    @Override
    public String getTransformerId()
    {
        return ID;
    }

    @Override
    public void transform(String transformName, String sourceMimetype, String targetMimetype,
                          Map<String, String> transformOptions,
                          File sourceFile, File targetFile) throws Exception
    {
        final SelectableTransformer transformer = transformers.get(transformName);
        logOptions(sourceFile, targetFile, transformOptions);
        transformer.transform(sourceMimetype, targetMimetype, transformOptions, sourceFile, targetFile);
    }
```

(*See* https://github.com/Alfresco/alfresco-transform-core/blob/e575ec943a5fa5dddca5593e6795a17a2bbb3cb6/alfresco-transform-misc/alfresco-transform-misc/src/main/java/org/alfresco/transformer/transformers/SelectingTransformer.java)

69. The Accused Products perform a method that includes *creating a message for each identified event containing text from the event according to a generic data structure corresponding to the event*. For example, the Alfresco Transform Service's Metadata Extractor identifies and extracts metadata from source files, where the extracted metadata stores common properties, such as author, title, subject, etc. in a generic data structure by mapping the common properties to content model properties as name value pairs.

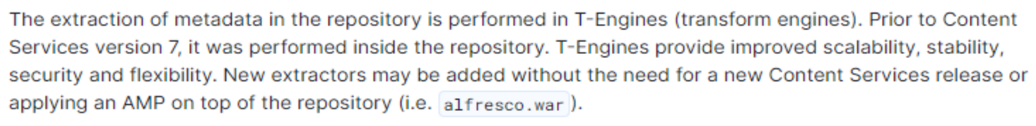

Every time a file is uploaded to the repository the file's MIME type is automatically detected. Based on the MIME type a related Metadata Extractor is invoked on the file. It will extract common properties from the file, such as author, and set the corresponding content model property accordingly. Each Metadata Extractor has a mapping between the properties it can extract and the content model properties.

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)

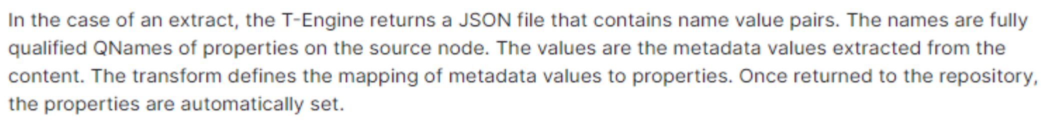

The properties that are extracted are limited to the out-of-the-box content model, which is very generic. Here are some example of extracted property name and what content model property it maps to:

- author → `cm:author`
- title → `cm:title`
- subject → `cm:description`
- created → `cm:created`
- description → *NOT MAPPED* - you could map it in a custom configuration
- comments → *NOT MAPPED* - you could map it in a custom configuration
- *If it is an image file:*
- EXIF metadata → `exif:exif` (pixel dimensions, manufacturer, model, software, date-time etc.)
- Geo metadata → `cm:geographic` (longitude & latitude)
- *If it is an audio file* → `audio:audio` (album, artist, composer, engineer, genre etc.)
- *If it is an email file* → `cm:emailed` (from, to, subject, sent date)

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)

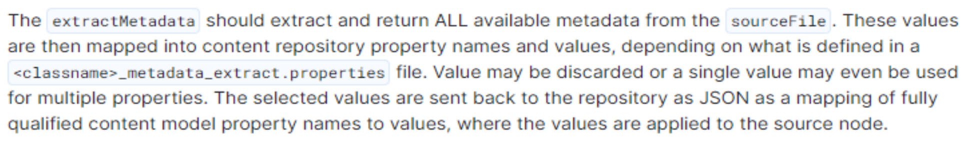

The extraction of metadata in the repository is performed in T-Engines (transform engines). Prior to Content Services version 7, it was performed inside the repository. T-Engines provide improved scalability, stability, security and flexibility. New extractors may be added without the need for a new Content Services release or applying an AMP on top of the repository (i.e. `alfresco.war`).

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)



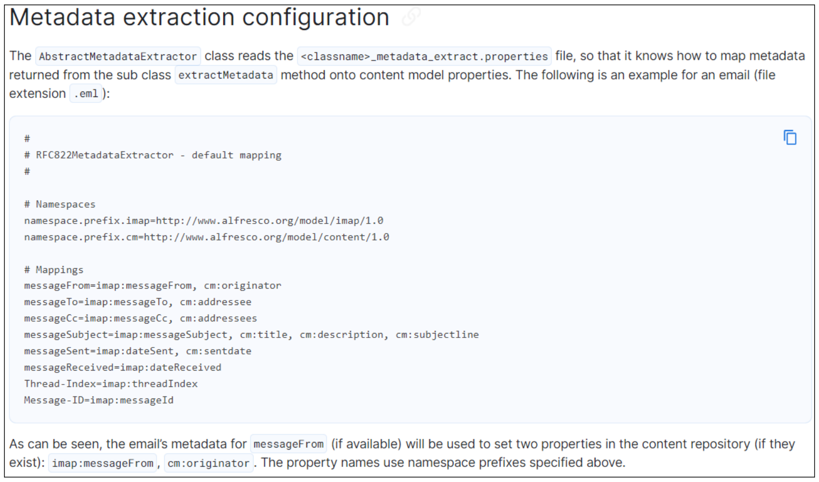
In the case of an extract, the T-Engine returns a JSON file that contains name value pairs. The names are fully qualified QNames of properties on the source node. The values are the metadata values extracted from the content. The transform defines the mapping of metadata values to properties. Once returned to the repository, the properties are automatically set.

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)



The `extractMetadata` should extract and return ALL available metadata from the `sourceFile`. These values are then mapped into content repository property names and values, depending on what is defined in a `<classname>_metadata_extract.properties` file. Value may be discarded or a single value may even be used for multiple properties. The selected values are sent back to the repository as JSON as a mapping of fully qualified content model property names to values, where the values are applied to the source node.

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)



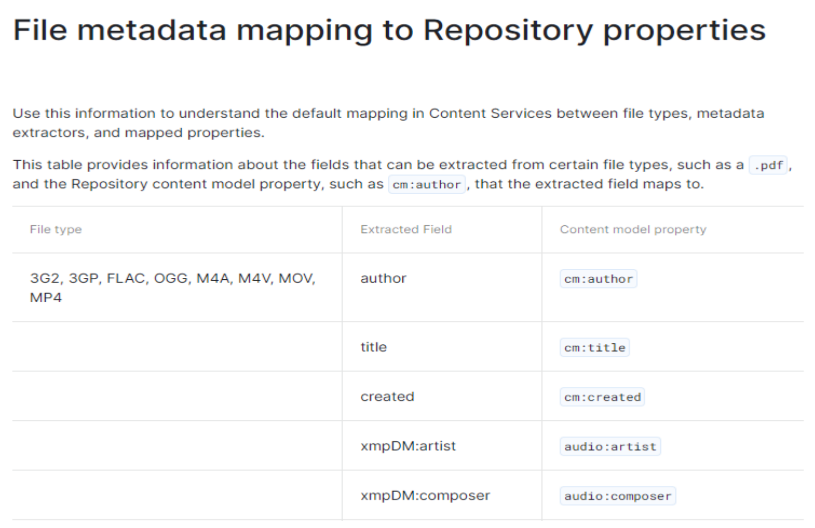
Metadata extraction configuration

The `AbstractMetadataExtractor` class reads the `<classname>_metadata_extract.properties` file, so that it knows how to map metadata returned from the sub class `extractMetadata` method onto content model properties. The following is an example for an email (file extension `.eml`):

```
#
# RFC822MetadataExtractor - default mapping
#

# Namespaces
namespace.prefix.imap=http://www.alfresco.org/model/imap/1.0
namespace.prefix.cm=http://www.alfresco.org/model/content/1.0

# Mappings
messageFrom=imap:messageFrom, cm:originator
messageTo=imap:messageTo, cm:addressee
messageCc=imap:messageCc, cm:addressees
messageSubject=imap:messageSubject, cm:title, cm:description, cm:subjectline
messageSent=imap:dateSent, cm:sentdate
messageReceived=imap:dateReceived
Thread-Index=imap:threadIndex
Message-ID=imap:messageId
```

As can be seen, the email's metadata for `messageFrom` (if available) will be used to set two properties in the content repository (if they exist): `imap:messageFrom`, `cm:originator`. The property names use namespace prefixes specified above.

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)



File metadata mapping to Repository properties

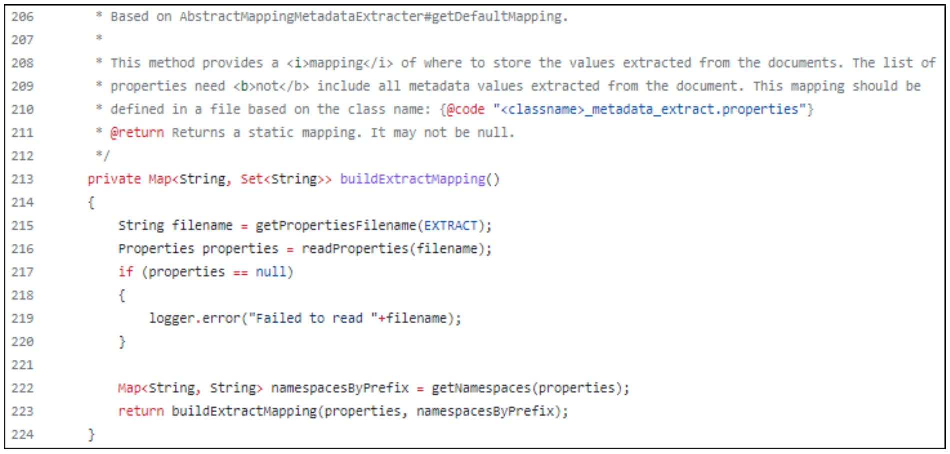
Use this information to understand the default mapping in Content Services between file types, metadata extractors, and mapped properties.

This table provides information about the fields that can be extracted from certain file types, such as a `.pdf`, and the Repository content model property, such as `cm:author`, that the extracted field maps to.

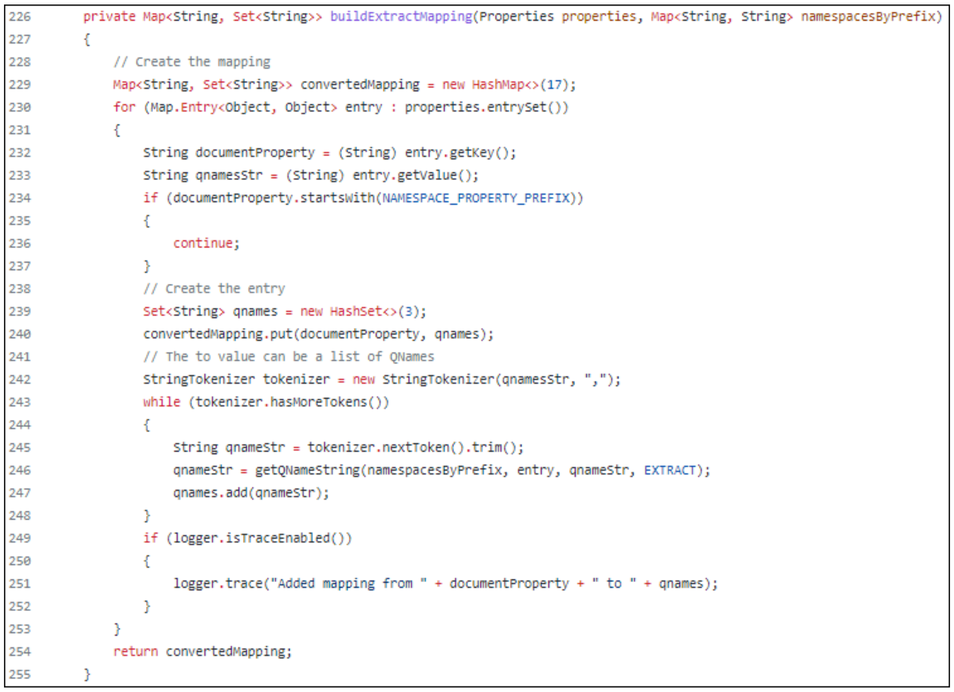
File type	Extracted Field	Content model property
3G2, 3GP, FLAC, OGG, M4A, M4V, MOV, MP4	author	`cm:author`
	title	`cm:title`
	created	`cm:created`
	xmpDM:artist	`audio:artist`
	xmpDM:composer	`audio:composer`

(*See* https://docs.alfresco.com/content-services/latest/admin/metadata-extraction/)

70. As another example, the Alfresco Transform Service implements "buildExtractMapping" method that identifies and extracts property values from an input file and store the values in "Map" data structure.

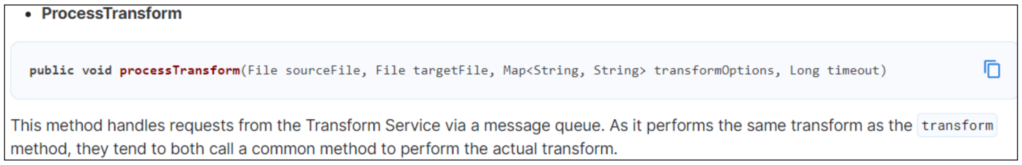


```
     * Based on AbstractMappingMetadataExtracter#getDefaultMapping.
     *
     * This method provides a <i>mapping</i> of where to store the values extracted from the documents. The list of
     * properties need <b>not</b> include all metadata values extracted from the document. This mapping should be
     * defined in a file based on the class name: {@code "<classname>_metadata_extract.properties"}
     * @return Returns a static mapping. It may not be null.
     */
    private Map<String, Set<String>> buildExtractMapping()
    {
        String filename = getPropertiesFilename(EXTRACT);
        Properties properties = readProperties(filename);
        if (properties == null)
        {
            logger.error("Failed to read "+filename);
        }

        Map<String, String> namespacesByPrefix = getNamespaces(properties);
        return buildExtractMapping(properties, namespacesByPrefix);
    }
```


```
    private Map<String, Set<String>> buildExtractMapping(Properties properties, Map<String, String> namespacesByPrefix)
    {
        // Create the mapping
        Map<String, Set<String>> convertedMapping = new HashMap<>(17);
        for (Map.Entry<Object, Object> entry : properties.entrySet())
        {
            String documentProperty = (String) entry.getKey();
            String qnamesStr = (String) entry.getValue();
            if (documentProperty.startsWith(NAMESPACE_PROPERTY_PREFIX))
            {
                continue;
            }
            // Create the entry
            Set<String> qnames = new HashSet<>(3);
            convertedMapping.put(documentProperty, qnames);
            // The to value can be a list of QNames
            StringTokenizer tokenizer = new StringTokenizer(qnamesStr, ",");
            while (tokenizer.hasMoreTokens())
            {
                String qnameStr = tokenizer.nextToken().trim();
                qnameStr = getQNameString(namespacesByPrefix, entry, qnameStr, EXTRACT);
                qnames.add(qnameStr);
            }
            if (logger.isTraceEnabled())
            {
                logger.trace("Added mapping from " + documentProperty + " to " + qnames);
            }
        }
        return convertedMapping;
    }
```

(*See* https://github.com/Alfresco/alfresco-transform-core/blob/e575ec943a5fa5dddca5593e6795a17a2bbb3cb6/alfresco-transformer-base/src/main/java/org/alfresco/transformer/metadataExtractors/AbstractMetadataExtractor.java)

71. In addition, the Alfresco Transform Service also implements "processTransform" method to "handle[] requests from the Transform Service via a message queue."


- **ProcessTransform**

```
public void processTransform(File sourceFile, File targetFile, Map<String, String> transformOptions, Long timeout)
```

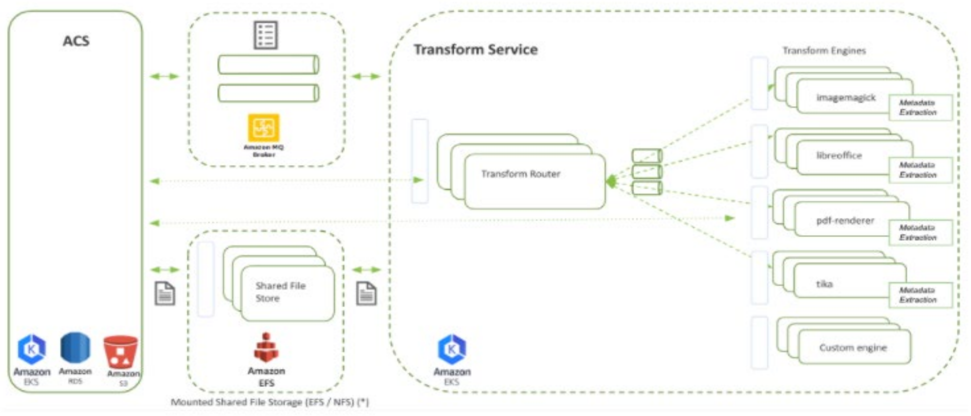
This method handles requests from the Transform Service via a message queue. As it performs the same transform as the `transform` method, they tend to both call a common method to perform the actual transform.

(*See* https://docs.alfresco.com/transform-service/1.4/config/engine/)

72. The Accused Products perform a method that includes *executing a process configured to create output data of a second format from the messages, the output data created from a processed message containing text from the processed message, the*

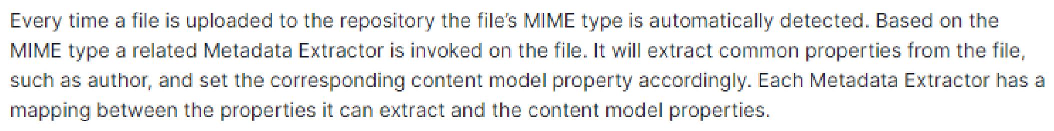
output data in a different format from the first document format. For example, the Alfresco Transform Service provides "a secure, scalable, reliable, and extensible mechanism for converting files from their current format into other formats." The Transform Engines perform transformation for conversion of files from their current format into other format, e.g., docx to pdf, pdf to png, and docx to plain text. (*See* https://docs.alfresco.com/transform-service/1.4/)

The following diagram shows a simple representation of the Transform Service components:



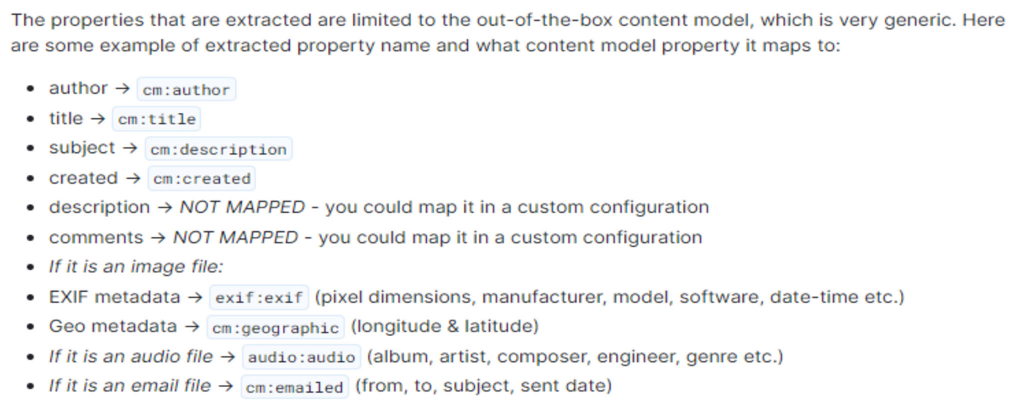
Note that from Transform Service version 1.3.2 the metadata extraction that usually takes part in the core repository legacy transform engines has now been lifted out into the separate transform engine processes. This enables scaling of the metadata extraction.

(*See* https://docs.alfresco.com/transform-service/1.4/admin/)



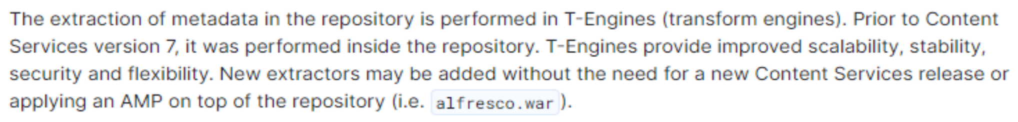
Every time a file is uploaded to the repository the file's MIME type is automatically detected. Based on the MIME type a related Metadata Extractor is invoked on the file. It will extract common properties from the file, such as author, and set the corresponding content model property accordingly. Each Metadata Extractor has a mapping between the properties it can extract and the content model properties.

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)


The properties that are extracted are limited to the out-of-the-box content model, which is very generic. Here are some example of extracted property name and what content model property it maps to:

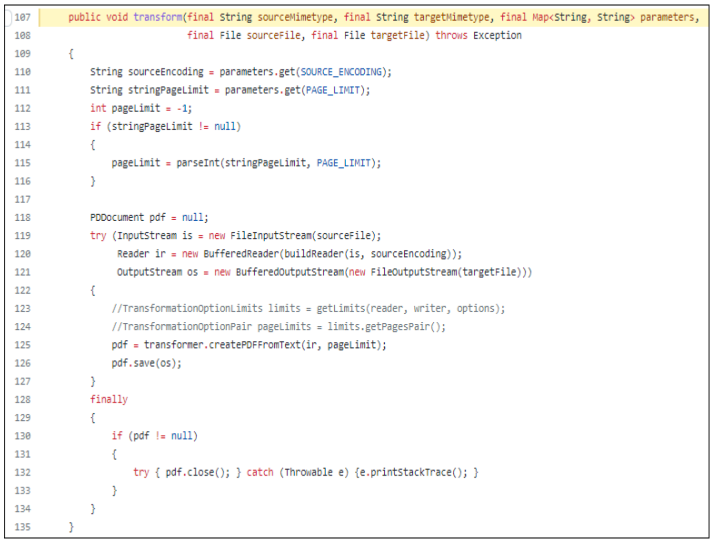
- author → `cm:author`
- title → `cm:title`
- subject → `cm:description`
- created → `cm:created`
- description → *NOT MAPPED* - you could map it in a custom configuration
- comments → *NOT MAPPED* - you could map it in a custom configuration
- *If it is an image file:*
- EXIF metadata → `exif:exif` (pixel dimensions, manufacturer, model, software, date-time etc.)
- Geo metadata → `cm:geographic` (longitude & latitude)
- *If it is an audio file* → `audio:audio` (album, artist, composer, engineer, genre etc.)
- *If it is an email file* → `cm:emailed` (from, to, subject, sent date)

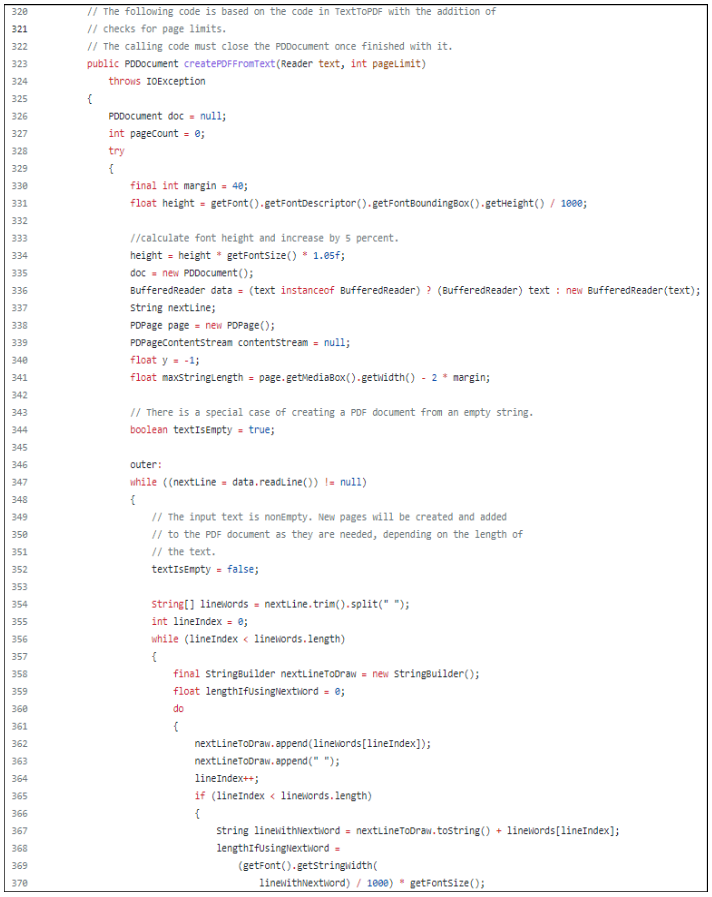
(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)


The extraction of metadata in the repository is performed in T-Engines (transform engines). Prior to Content Services version 7, it was performed inside the repository. T-Engines provide improved scalability, stability, security and flexibility. New extractors may be added without the need for a new Content Services release or applying an AMP on top of the repository (i.e. `alfresco.war`).

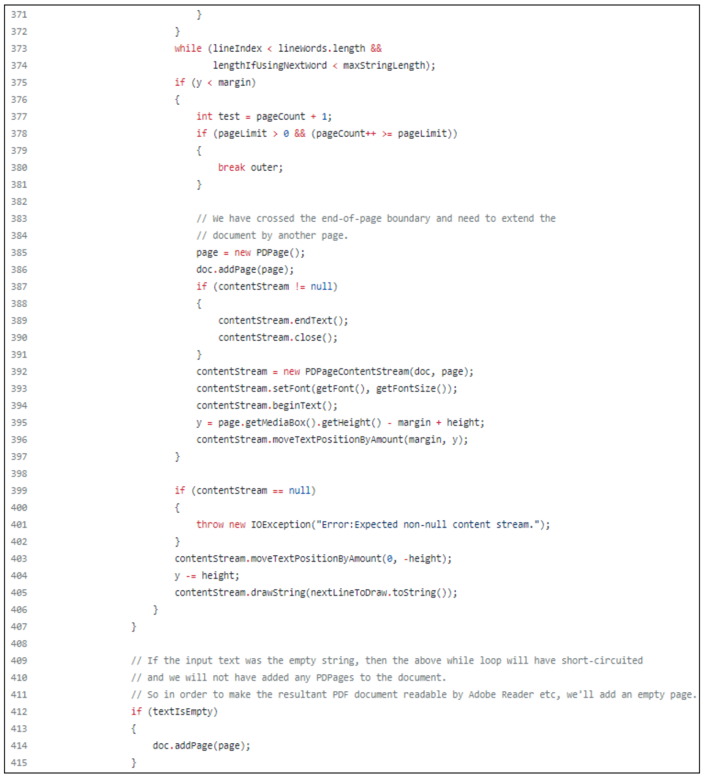
(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)

73. As an example, the Alfresco Transform Service implements a text to PDF content transformation that creates output data of a second format (e.g., PDF) from a processed message containing text from first document format (e.g., text).


```
    public void transform(final String sourceMimetype, final String targetMimetype, final Map<String, String> parameters,
                          final File sourceFile, final File targetFile) throws Exception
    {
        String sourceEncoding = parameters.get(SOURCE_ENCODING);
        String stringPageLimit = parameters.get(PAGE_LIMIT);
        int pageLimit = -1;
        if (stringPageLimit != null)
        {
            pageLimit = parseInt(stringPageLimit, PAGE_LIMIT);
        }

        PDDocument pdf = null;
        try (InputStream is = new FileInputStream(sourceFile);
             Reader ir = new BufferedReader(buildReader(is, sourceEncoding));
             OutputStream os = new BufferedOutputStream(new FileOutputStream(targetFile)))
        {
            //TransformationOptionLimits limits = getLimits(reader, writer, options);
            //TransformationOptionPair pageLimits = limits.getPagesPair();
            pdf = transformer.createPDFFromText(ir, pageLimit);
            pdf.save(os);
        }
        finally
        {
            if (pdf != null)
            {
                try { pdf.close(); } catch (Throwable e) {e.printStackTrace(); }
            }
        }
    }
```



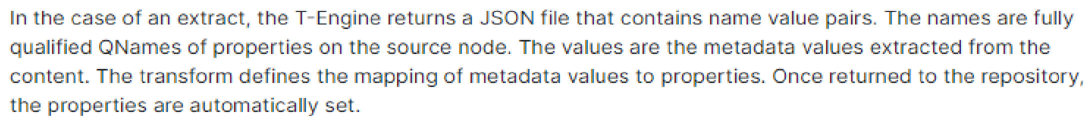
```java
    // The following code is based on the code in TextToPDF with the addition of
    // checks for page limits.
    // The calling code must close the PDDocument once finished with it.
    public PDDocument createPDFFromText(Reader text, int pageLimit)
        throws IOException
    {
        PDDocument doc = null;
        int pageCount = 0;
        try
        {
            final int margin = 40;
            float height = getFont().getFontDescriptor().getFontBoundingBox().getHeight() / 1000;

            //calculate font height and increase by 5 percent.
            height = height * getFontSize() * 1.05f;
            doc = new PDDocument();
            BufferedReader data = (text instanceof BufferedReader) ? (BufferedReader) text : new BufferedReader(text);
            String nextLine;
            PDPage page = new PDPage();
            PDPageContentStream contentStream = null;
            float y = -1;
            float maxStringLength = page.getMediaBox().getWidth() - 2 * margin;

            // There is a special case of creating a PDF document from an empty string.
            boolean textIsEmpty = true;

            outer:
            while ((nextLine = data.readLine()) != null)
            {
                // The input text is nonEmpty. New pages will be created and added
                // to the PDF document as they are needed, depending on the length of
                // the text.
                textIsEmpty = false;

                String[] lineWords = nextLine.trim().split(" ");
                int lineIndex = 0;
                while (lineIndex < lineWords.length)
                {
                    final StringBuilder nextLineToDraw = new StringBuilder();
                    float lengthIfUsingNextWord = 0;
                    do
                    {
                        nextLineToDraw.append(lineWords[lineIndex]);
                        nextLineToDraw.append(" ");
                        lineIndex++;
                        if (lineIndex < lineWords.length)
                        {
                            String lineWithNextWord = nextLineToDraw.toString() + lineWords[lineIndex];
                            lengthIfUsingNextWord =
                                (getFont().getStringWidth(
                                    lineWithNextWord) / 1000) * getFontSize();
```



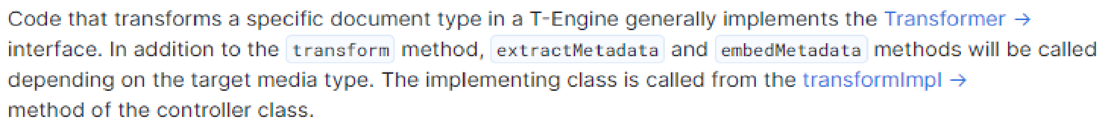
```java
                        }
                    }
                    while (lineIndex < lineWords.length &&
                            lengthIfUsingNextWord < maxStringLength);
                    if (y < margin)
                    {
                        int test = pageCount + 1;
                        if (pageLimit > 0 && (pageCount++ >= pageLimit))
                        {
                            break outer;
                        }

                        // We have crossed the end-of-page boundary and need to extend the
                        // document by another page.
                        page = new PDPage();
                        doc.addPage(page);
                        if (contentStream != null)
                        {
                            contentStream.endText();
                            contentStream.close();
                        }
                        contentStream = new PDPageContentStream(doc, page);
                        contentStream.setFont(getFont(), getFontSize());
                        contentStream.beginText();
                        y = page.getMediaBox().getHeight() - margin + height;
                        contentStream.moveTextPositionByAmount(margin, y);
                    }

                    if (contentStream == null)
                    {
                        throw new IOException("Error:Expected non-null content stream.");
                    }
                    contentStream.moveTextPositionByAmount(0, -height);
                    y -= height;
                    contentStream.drawString(nextLineToDraw.toString());
                }
            }

            // If the input text was the empty string, then the above while loop will have short-circuited
            // and we will not have added any PDPages to the document.
            // So in order to make the resultant PDF document readable by Adobe Reader etc, we'll add an empty page.
            if (textIsEmpty)
            {
                doc.addPage(page);
            }
```

(*See* https://github.com/Alfresco/alfresco-transform-core/blob/e575ec943a5fa5dddca5593e6795a17a2bbb3cb6/alfresco-transform-misc/alfresco-transform-misc/src/main/java/org/alfresco/transformer/transformers/TextToPdfContentTransformer.java)

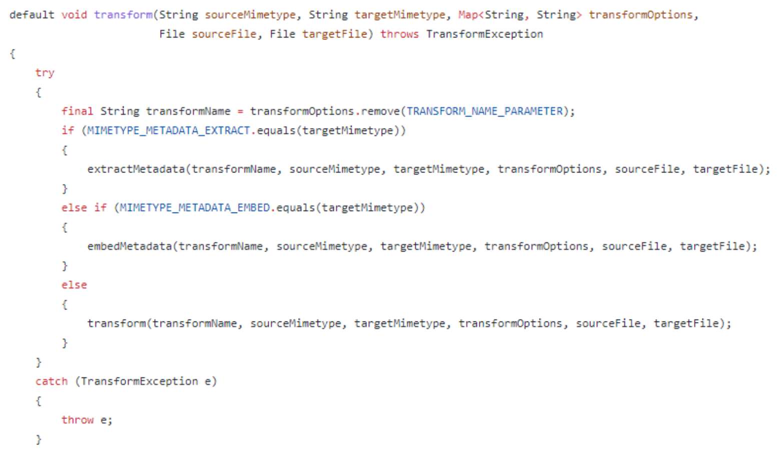
74. In addition, the Transform Engines (e.g., T-Engines) also perform metadata extraction by calling the "extractMetadata" method, where the "extractMetadata" method extracts and returns "All available metadata from the sourceFile." "In the case of an extract, the T-Engine returns a JSON file that contains name value pairs" of the extracted metadata and embeds the extracted metadata into the output file.


In the case of an extract, the T-Engine returns a JSON file that contains name value pairs. The names are fully qualified QNames of properties on the source node. The values are the metadata values extracted from the content. The transform defines the mapping of metadata values to properties. Once returned to the repository, the properties are automatically set.

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)


Code that transforms a specific document type in a T-Engine generally implements the Transformer → interface. In addition to the `transform` method, `extractMetadata` and `embedMetadata` methods will be called depending on the target media type. The implementing class is called from the transformImpl → method of the controller class.

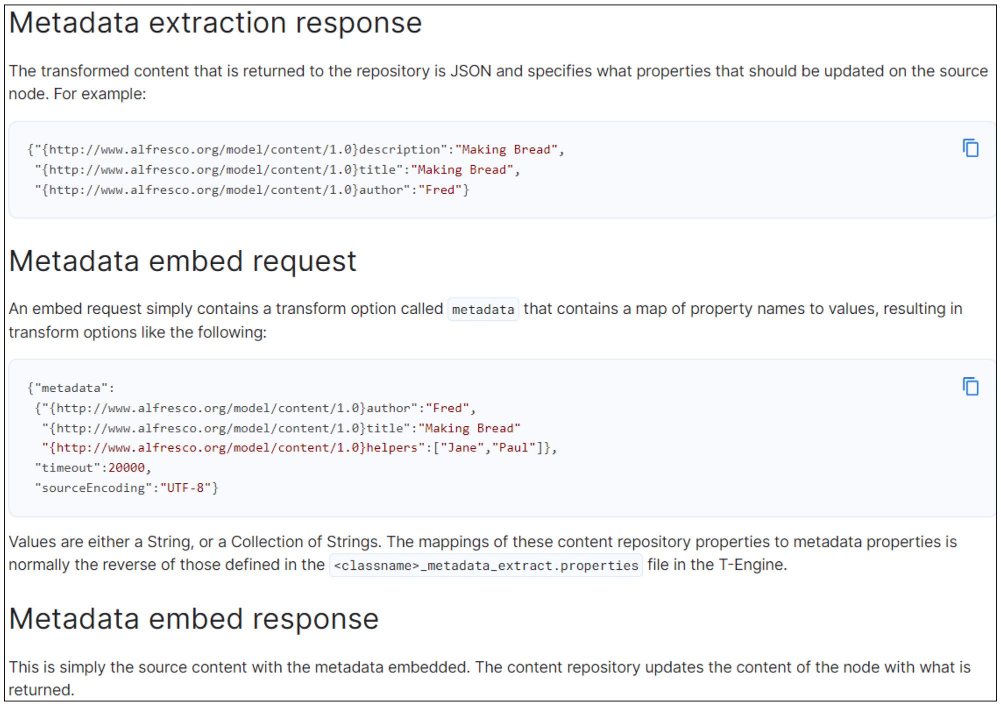
(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)


```
default void transform(String sourceMimetype, String targetMimetype, Map<String, String> transformOptions,
                       File sourceFile, File targetFile) throws TransformException
{
    try
    {
        final String transformName = transformOptions.remove(TRANSFORM_NAME_PARAMETER);
        if (MIMETYPE_METADATA_EXTRACT.equals(targetMimetype))
        {
            extractMetadata(transformName, sourceMimetype, targetMimetype, transformOptions, sourceFile, targetFile);
        }
        else if (MIMETYPE_METADATA_EMBED.equals(targetMimetype))
        {
            embedMetadata(transformName, sourceMimetype, targetMimetype, transformOptions, sourceFile, targetFile);
        }
        else
        {
            transform(transformName, sourceMimetype, targetMimetype, transformOptions, sourceFile, targetFile);
        }
    }
    catch (TransformException e)
    {
        throw e;
    }
```

(*See* https://github.com/Alfresco/alfresco-transform-core/blob/master/alfresco-transformer-base/src/main/java/org/alfresco/transformer/executors/Transformer.java)

> The `extractMetadata` should extract and return ALL available metadata from the `sourceFile`. These values are then mapped into content repository property names and values, depending on what is defined in a `<classname>_metadata_extract.properties` file. Value may be discarded or a single value may even be used for multiple properties. The selected values are sent back to the repository as JSON as a mapping of fully qualified content model property names to values, where the values are applied to the source node.

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)



Metadata extraction response

The transformed content that is returned to the repository is JSON and specifies what properties that should be updated on the source node. For example:

```
{"{http://www.alfresco.org/model/content/1.0}description":"Making Bread",
 "{http://www.alfresco.org/model/content/1.0}title":"Making Bread",
 "{http://www.alfresco.org/model/content/1.0}author":"Fred"}
```

Metadata embed request

An embed request simply contains a transform option called `metadata` that contains a map of property names to values, resulting in transform options like the following:

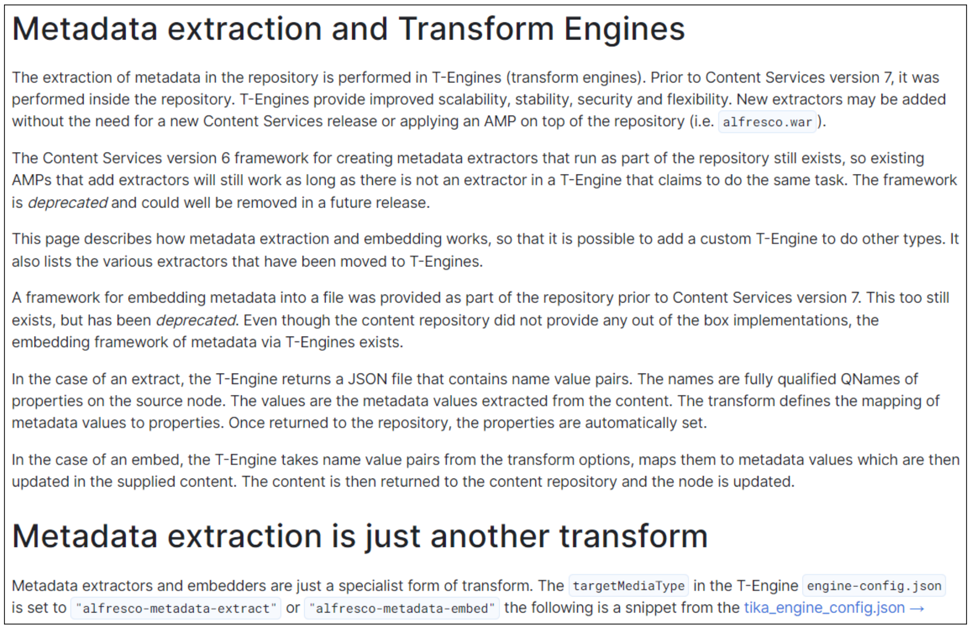
```
{"metadata":
 {"{http://www.alfresco.org/model/content/1.0}author":"Fred",
  "{http://www.alfresco.org/model/content/1.0}title":"Making Bread"
  "{http://www.alfresco.org/model/content/1.0}helpers":["Jane","Paul"]},
 "timeout":20000,
 "sourceEncoding":"UTF-8"}
```

Values are either a String, or a Collection of Strings. The mappings of these content repository properties to metadata properties is normally the reverse of those defined in the `<classname>_metadata_extract.properties` file in the T-Engine.

Metadata embed response

This is simply the source content with the metadata embedded. The content repository updates the content of the node with what is returned.

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)



Metadata extraction and Transform Engines

The extraction of metadata in the repository is performed in T-Engines (transform engines). Prior to Content Services version 7, it was performed inside the repository. T-Engines provide improved scalability, stability, security and flexibility. New extractors may be added without the need for a new Content Services release or applying an AMP on top of the repository (i.e. `alfresco.war`).

The Content Services version 6 framework for creating metadata extractors that run as part of the repository still exists, so existing AMPs that add extractors will still work as long as there is not an extractor in a T-Engine that claims to do the same task. The framework is *deprecated* and could well be removed in a future release.

This page describes how metadata extraction and embedding works, so that it is possible to add a custom T-Engine to do other types. It also lists the various extractors that have been moved to T-Engines.

A framework for embedding metadata into a file was provided as part of the repository prior to Content Services version 7. This too still exists, but has been *deprecated*. Even though the content repository did not provide any out of the box implementations, the embedding framework of metadata via T-Engines exists.

In the case of an extract, the T-Engine returns a JSON file that contains name value pairs. The names are fully qualified QNames of properties on the source node. The values are the metadata values extracted from the content. The transform defines the mapping of metadata values to properties. Once returned to the repository, the properties are automatically set.

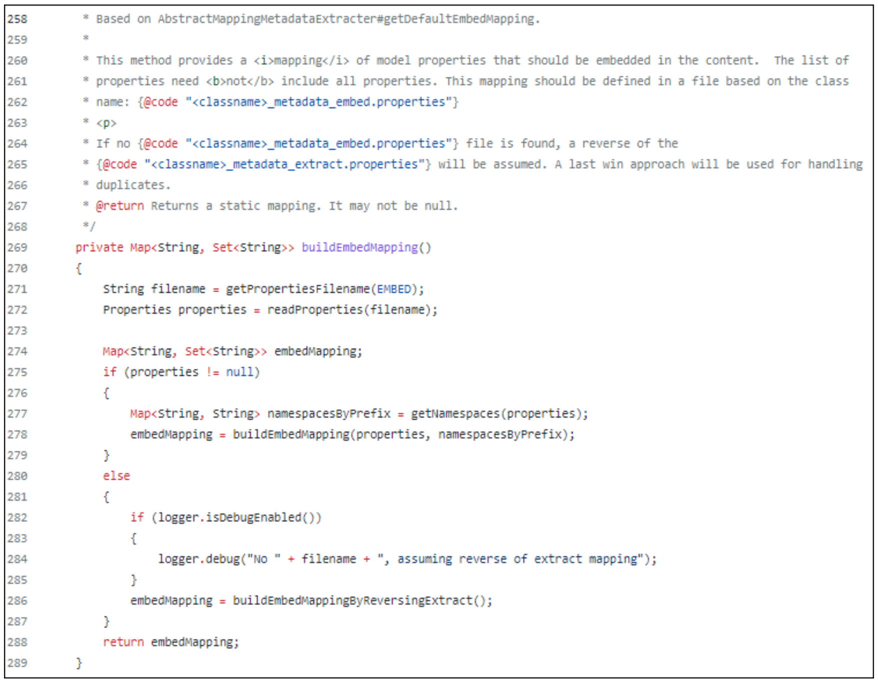
In the case of an embed, the T-Engine takes name value pairs from the transform options, maps them to metadata values which are then updated in the supplied content. The content is then returned to the content repository and the node is updated.

Metadata extraction is just another transform

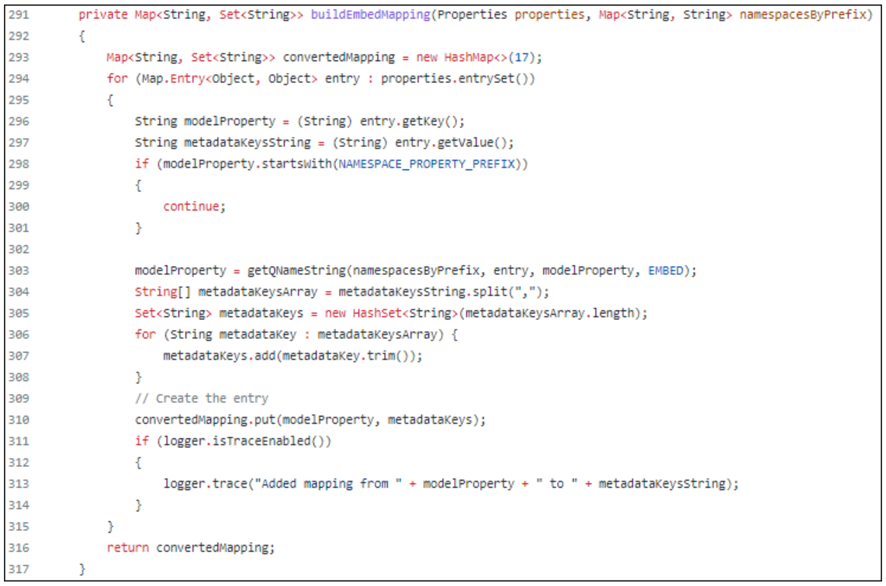
Metadata extractors and embedders are just a specialist form of transform. The `targetMediaType` in the T-Engine `engine-config.json` is set to `"alfresco-metadata-extract"` or `"alfresco-metadata-embed"` the following is a snippet from the tika_engine_config.json →

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/#metadata-extraction-is-just-another-transform)

75. The Alfresco Transform Service implements "buildEmbedMapping" method to provide mappings of model properties to metadata and then to embed the metadata into the content of a target file.

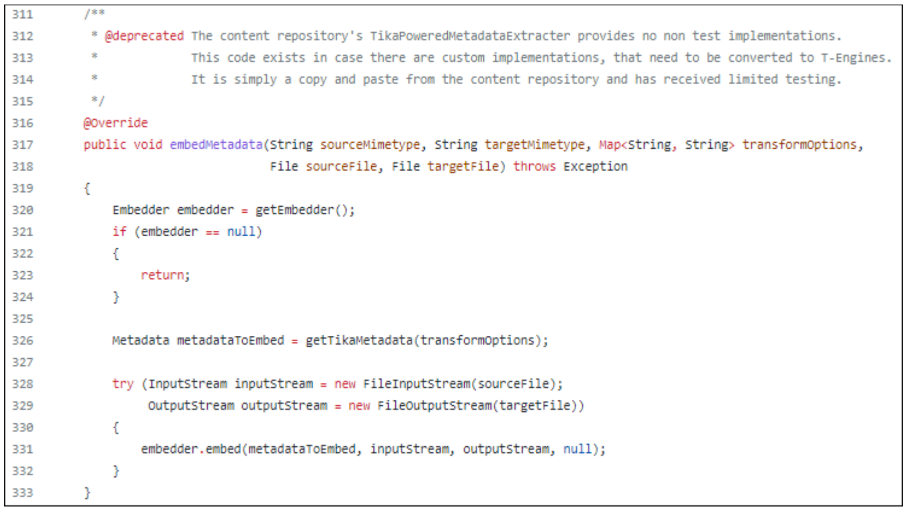


```
     * Based on AbstractMappingMetadataExtracter#getDefaultEmbedMapping.
     *
     * This method provides a <i>mapping</i> of model properties that should be embedded in the content.  The list of
     * properties need <b>not</b> include all properties. This mapping should be defined in a file based on the class
     * name: {@code "<classname>_metadata_embed.properties"}
     * <p>
     * If no {@code "<classname>_metadata_embed.properties"} file is found, a reverse of the
     * {@code "<classname>_metadata_extract.properties"} will be assumed. A last win approach will be used for handling
     * duplicates.
     * @return Returns a static mapping. It may not be null.
     */
    private Map<String, Set<String>> buildEmbedMapping()
    {
        String filename = getPropertiesFilename(EMBED);
        Properties properties = readProperties(filename);

        Map<String, Set<String>> embedMapping;
        if (properties != null)
        {
            Map<String, String> namespacesByPrefix = getNamespaces(properties);
            embedMapping = buildEmbedMapping(properties, namespacesByPrefix);
        }
        else
        {
            if (logger.isDebugEnabled())
            {
                logger.debug("No " + filename + ", assuming reverse of extract mapping");
            }
            embedMapping = buildEmbedMappingByReversingExtract();
        }
        return embedMapping;
    }
```



```java
    private Map<String, Set<String>> buildEmbedMapping(Properties properties, Map<String, String> namespacesByPrefix)
    {
        Map<String, Set<String>> convertedMapping = new HashMap<>(17);
        for (Map.Entry<Object, Object> entry : properties.entrySet())
        {
            String modelProperty = (String) entry.getKey();
            String metadataKeysString = (String) entry.getValue();
            if (modelProperty.startsWith(NAMESPACE_PROPERTY_PREFIX))
            {
                continue;
            }

            modelProperty = getQNameString(namespacesByPrefix, entry, modelProperty, EMBED);
            String[] metadataKeysArray = metadataKeysString.split(",");
            Set<String> metadataKeys = new HashSet<String>(metadataKeysArray.length);
            for (String metadataKey : metadataKeysArray) {
                metadataKeys.add(metadataKey.trim());
            }
            // Create the entry
            convertedMapping.put(modelProperty, metadataKeys);
            if (logger.isTraceEnabled())
            {
                logger.trace("Added mapping from " + modelProperty + " to " + metadataKeysString);
            }
        }
        return convertedMapping;
    }
```

(*See* https://github.com/Alfresco/alfresco-transform-core/blob/e575ec943a5fa5dddca5593e6795a17a2bbb3cb6/alfresco-transformer-base/src/main/java/org/alfresco/transformer/metadataExtractors/AbstractMetadataExtractor.java)



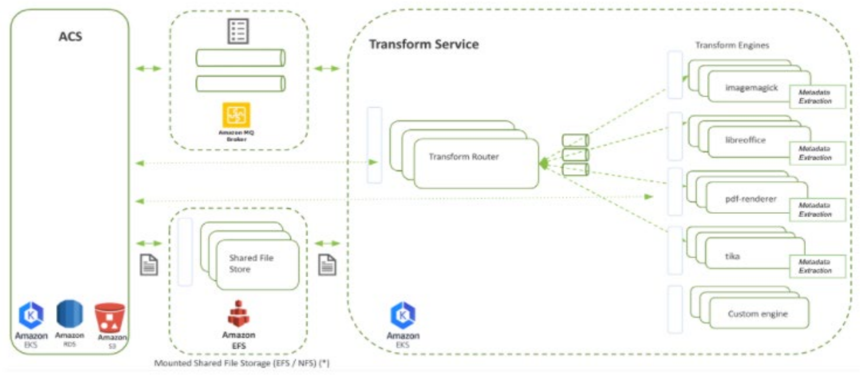
```java
    /**
     * @deprecated The content repository's TikaPoweredMetadataExtracter provides no non test implementations.
     *             This code exists in case there are custom implementations, that need to be converted to T-Engines.
     *             It is simply a copy and paste from the content repository and has received limited testing.
     */
    @Override
    public void embedMetadata(String sourceMimetype, String targetMimetype, Map<String, String> transformOptions,
                              File sourceFile, File targetFile) throws Exception
    {
        Embedder embedder = getEmbedder();
        if (embedder == null)
        {
            return;
        }

        Metadata metadataToEmbed = getTikaMetadata(transformOptions);

        try (InputStream inputStream = new FileInputStream(sourceFile);
             OutputStream outputStream = new FileOutputStream(targetFile))
        {
            embedder.embed(metadataToEmbed, inputStream, outputStream, null);
        }
    }
```

76. The Accused Products perform a method that includes *sending an output data stream to a destination, the output data stream comprising the output data*. For example, the Alfresco Transform Service uses Shared File Store "as temporary storage for the original file (stored by the repository), intermediate files for multi-step transforms, and the final transformed target file. The target file is retrieved by the

repository after it's been processed by one or more of the Transform Engines."

- **Transform Engines**: The Transform Engines transform files referenced by the repository and retrieved from the shared file store. Here are some example transformations for each Transform Engine (this is not an exhaustive list):
  - LibreOffice (e.g. docx to pdf)
  - ImageMagick (e.g. resize)
  - Alfresco PDF Renderer (e.g. pdf to png)
  - Tika (e.g. docx to plain text)
  - Misc. (not included in diagram)
- **Shared File Store**: This is used as temporary storage for the original source file (stored by the repository), intermediate files for multi-step transforms, and the final transformed target file. The target file is retrieved by the repository after it's been processed by one or more of the Transform Engines.

The following diagram shows a simple representation of the Transform Service components:



Note that from Transform Service version 1.3.2 the metadata extraction that usually takes part in the core repository legacy transform engines has now been lifted out into the separate transform engine processes. This enables scaling of the metadata extraction.

(*See* https://docs.alfresco.com/transform-service/latest/admin/)

77. Each claim in the '146 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the '146 Patent.

78. There has been significant effort by Hyland to imitate OpenText's patent-protected products to compete with OpenText in the ECM and EIM markets and to increase Hyland's share of that market at the expense of OpenText's market share. (*See, e.g.*, Exhibits 24, Exhibit B (2020.09.09 - Hyland enters definitive agreement to acquire Alfresco, hyland.com), Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com).) Hyland's efforts have resulted in the Accused Products, which infringe at least claim 1 of the '146 patent as described above, and

those efforts would have exposed Hyland to the ’146 patent prior to the filing of the original Complaint in this action.

79. Hyland has known of the ’146 Patent since receiving a letter identifying the patent and the infringement on September 2, 2022. At the very least, Hyland has been aware of the ’146 patent and of its infringement based on the Accused Products since at least the filing and/or service of this Complaint. Further, OpenText marks its products with the ’146 patent.

80. On information and belief, at least as of the filing of the Complaint in this action, Hyland has knowingly and actively induced and is knowingly and actively inducing at least its customers and partners to directly infringe at least claim 1 of the ’146 patent, and has done so with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b), by activities relating to selling, marketing, advertising, promoting, supporting, installing, and distributing the Accused Products in the United States. (Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com).) On information and belief, those activities continue.

81. On information and belief, Hyland deliberately and knowingly encourages, instructs, directs, and/or requires third parties—including its partners, customers, and/or end users—to use the Accused Products in a way that infringes at least claim 1 of the ’146 patent as described above.

82. Hyland’s partners, customers, and end users of its Accused Products directly infringe at least claim 1 of the ’146 patent, at least by using Accused Products, as described above.

83. For example, on information and belief, Hyland knowingly and intentionally shares instructions, guides, and manuals, including through its website, training programs, and/or YouTube, which advertise and instruct third parties on how to use the Accused Products in a way that directly infringes at least claim 1 of the ’146

patent as described above, including at least Hyland's customers. On further information and belief, Hyland knowingly and intentionally provides customer service or technical support to purchasers of the infringing Accused Products, which directs and encourages Hyland's customers to use the Accused Products in a way that directly infringes at least claim 1 of the '146 patent as described above.

84. On information and belief, the infringing actions of each customer and/or end-user of the Accused Products are attributable to Hyland.

85. On information and belief, Hyland sells and offers for sale the Accused Products and provides technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. On information and belief, each customer enters into a contractual relationship with Hyland, which obligates each customer to perform certain actions as a condition to use the Accused Products. Further, in order to receive the benefit of Hyland's continued technical support and its specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that infringes the '146 patent. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Products remain operational for each customer through ongoing technical support, on information and belief, Hyland establishes the manner and timing of each customer's performance of activities that infringe the '146 patent.

86. On information and belief, Hyland forms a joint enterprise with its customers to engage in directly infringing the '146 patent. On further information and belief, Hyland together with each customer operate under a contractual agreement; have a common purpose to operate the Accused Products in a way that directly infringes the '146 patent as outlined in the paragraphs above; have pecuniary interests in operating the Accused Products by directly profiting from the sale and/or maintenance of the Accused Products or by indirectly profiting from the increased efficiency resulting from

use of the Accused Products; and have equal rights to a voice in the direction of the enterprise either by guiding and advising on the operation and capabilities of the Accused Products with product-specific know-how and expertise or by requesting that certain customer-specific capabilities be implemented in the Accused Products.

87. Hyland also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the inventions claimed, and not a staple article or commodity of commerce suitable for substantial non-infringing uses.

88. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including the transformation of files from their current format into other formats and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’146 patent, that functionality could not be performed.

89. Additionally, the accused functionality, including the transformation of files from their current format into other formats and related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For example, without the Alfresco Transformation Services, the Accused Products could not convert files from one format to another, including metadata. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer function for its intended

purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’146 Patent, that functionality could not be performed.

90. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Hyland constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionalities (including the file transformation functionality) constitute a material part of the inventions claimed because such functionality is integral to the processes identified above (such as to detect “patterns in an input file of the input data stream to identify events,” create “messages for each identified event containing text from the event according to a generic data structure corresponding to the event,” and “create output data of a second format from the messages”) as recited in the claims of the ’146 Patent. None of these products are staple goods—they are sophisticated and customized ECM products, methods, and systems.

91. OpenText “consists of four revenue streams: license, cloud services and subscriptions, customer support, and professional service and other.” (Exhibit A at 9-10 (Aug. 6, 2020 10-K).) Each revenue stream relates directly to the ability of OpenText to acquire and retain customers for its software products in a market that is “highly competitive” and increasingly more competitive “as a result of ongoing software industry consolidation,” such as Hyland’s acquisition of Alfresco. (Exhibit A at 11 (Aug. 6, 2020 10-K); *see also* Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com); Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com); Exhibit F at 4 (“The Forrester Wave: ECM Content Platforms, Q3 2019”); Exhibit E at 3 (2020.11.16 - Gartner Content Services Report 2020).) OpenText is an innovator in the market and has acquired multiple patents, including the Patents-in-Suit, to give it an advantage over such competition. Hyland’s infringing activities have resulted and will continue to

result in irreparable harm to OpenText because of the competitive threat that Hyland—including Hyland's acquisition of Alfresco—has to OpenText's share of the relevant "highly competitive" market, and the impact that Hyland's infringing activities have on each one of OpenText's four revenue streams. Further, public interest factors favor OpenText as the owner and assignee of government-issued patents, including the Patents-in-Suit, that serve to recognize OpenText's innovative contribution to the public knowledge in exchange for the patent protection that Hyland is now infringing.

92. For past infringement, OpenText has suffered damages, including lost profits, as a result of Hyland's infringement of the '146 patent. Hyland is therefore liable to OpenText under 35 U.S.C. § 284 for past damages in an amount that adequately compensates OpenText for Hyland's infringement, but no less than a reasonable royalty.

93. OpenText is entitled to a preliminary injunction to maintain the status quo between OpenText and Hyland, which, through its acquisition of Alfresco, is now one of OpenText's biggest competitors (*see, e.g.*, Exhibit B (2020.09.09 - Hyland enters definitive agreement to acquire Alfresco, hyland.com), Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com)), and is using OpenText's patented technology to compete with OpenText in the ECM and EIM markets.

94. For ongoing and future infringement, OpenText will continue to suffer irreparable harm, including without limitation, loss of market share, customers and/or convoyed sales and services which cannot be accurately quantified nor adequately compensated for by money damages, unless this Court preliminarily and permanently enjoins Hyland, its agents, employees, representatives, and all others acting in concert with Hyland from infringing the '146 patent.

95. In the alternative, OpenText is entitled to damages in lieu of an injunction, in an amount consistent with the facts, for future infringement. Hyland's continued

infringement, at least since it had notice of the ’146 patent, is knowing and willful. Hyland will be an adjudicated infringer of a valid patent and, thus, Hyland’s future infringement will be willful as a matter of law.

96. Hyland’s infringement is without license or other authorization.

97. This case is exceptional, entitling Plaintiffs to enhanced damages under 35 U.S.C. § 284 and an award of attorneys’ fees and costs incurred in prosecuting this action under 35 U.S.C. § 285.

SECOND CAUSE OF ACTION

(INFRINGEMENT OF THE ’830 PATENT)

98. Plaintiffs reallege and incorporate the preceding paragraphs of this complaint.

99. Defendants have infringed and continue to infringe one or more claims of the ’830 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including features of the Alfresco Enterprise Content Management System (ECM) and Alfresco Content Services, such as Alfresco Transform Service, at least when used for their ordinary and customary purposes, practice each element of at least claim 1 of the ’830 Patent as described below.

100. For example, claim 1 of the ’830 patent recites:

> 1. A method for processing a data stream in a network environment, comprising:
>
> at a server computer, creating a plurality of input threads, wherein each input thread of the plurality of input threads listens to a physical port from which the input thread receives data;
>
> receiving an input data stream of file data at a physical input associated with a first input thread of the plurality of input threads;

producing filtered data from the input data stream by passing the input data stream through a filter associated with the first input thread;

identifying, by an agent associated with the first input thread, each event in the filtered data, wherein the agent creates a message for each event containing text from the event and a generic data structure corresponding to the event for a thread job manager associated with the first input thread;

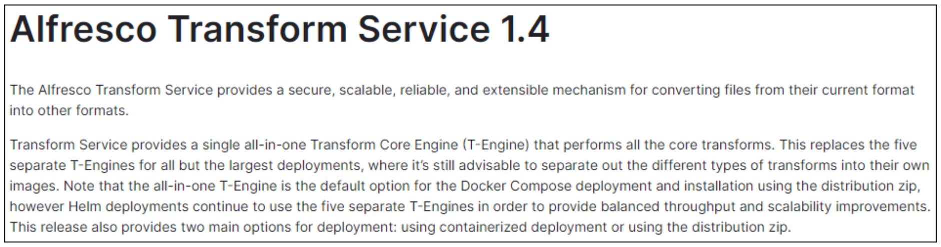
creating, by the thread job manager associated with the first input thread, a process for transforming the messages according to the generic data structure into output data, wherein

input data in the first data stream is of a first document format and the output data is of a second document format;

creating an output pipeline for the process; and

executing the process to produce a physical output object through the output pipeline.

101. The Accused Products perform the method of claim 1 of the ’830 Patent. To the extent the preamble is construed to be limiting, the Accused Products perform *a method for processing a data stream in a network environment*, as further explained below. For example, the Alfresco Transform Service converts files “from their current format into other formats” in a network environment.



Alfresco Transform Service 1.4

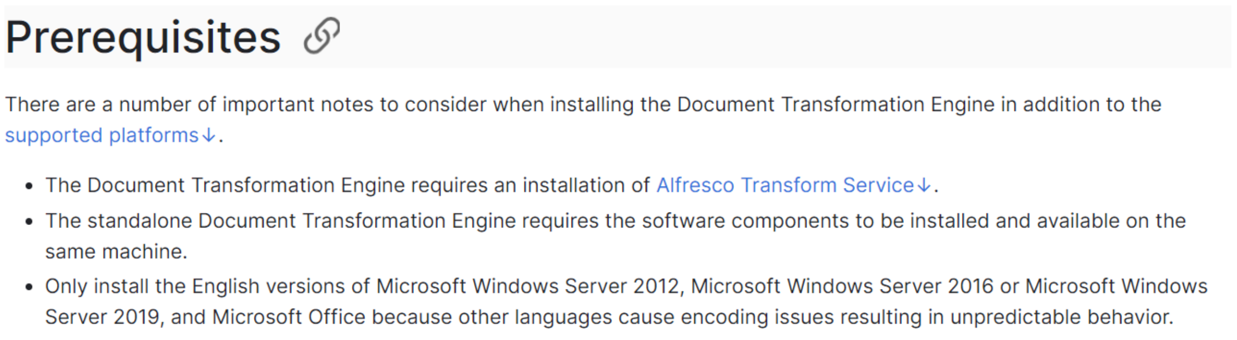
The Alfresco Transform Service provides a secure, scalable, reliable, and extensible mechanism for converting files from their current format into other formats.

Transform Service provides a single all-in-one Transform Core Engine (T-Engine) that performs all the core transforms. This replaces the five separate T-Engines for all but the largest deployments, where it's still advisable to separate out the different types of transforms into their own images. Note that the all-in-one T-Engine is the default option for the Docker Compose deployment and installation using the distribution zip, however Helm deployments continue to use the five separate T-Engines in order to provide balanced throughput and scalability improvements. This release also provides two main options for deployment: using containerized deployment or using the distribution zip.

(*See* https://docs.alfresco.com/transform-service/1.4/.)

102. On information and belief, the Accused Products perform a method that includes *at a server computer, creating a plurality of input threads, wherein each input*

thread of the plurality of input threads listens to a physical port from which the input thread receives data. For example, the Alfresco Transform Service includes multiple transform routers, transformation engines and metadata extractors, wherein the transformation engines are installed for example on Window Server 2012, Window Server 2016 or Windows Server 2019. A source file is fed into the Alfresco Transform Service via a port and a proper transformer is selected to convert the source file from its current format into another format.



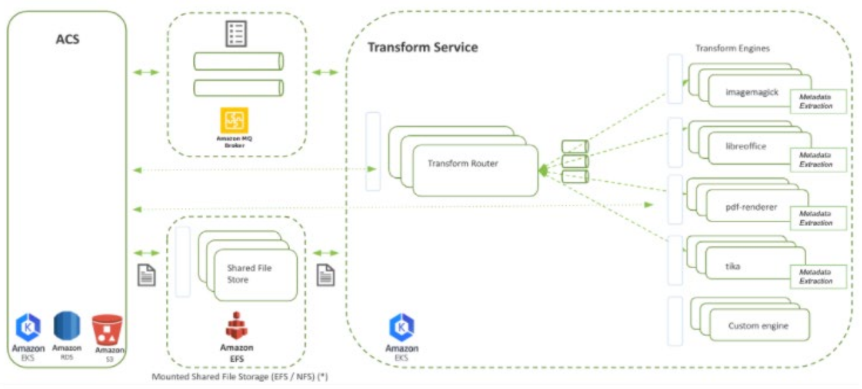
Prerequisites

There are a number of important notes to consider when installing the Document Transformation Engine in addition to the supported platforms↓.

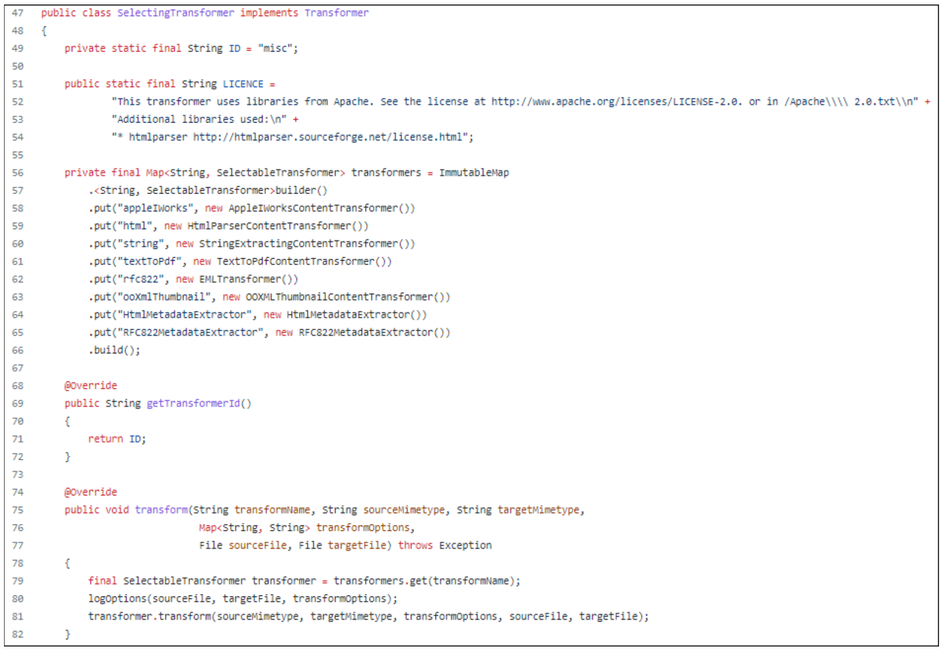
- The Document Transformation Engine requires an installation of Alfresco Transform Service↓.
- The standalone Document Transformation Engine requires the software components to be installed and available on the same machine.
- Only install the English versions of Microsoft Windows Server 2012, Microsoft Windows Server 2016 or Microsoft Windows Server 2019, and Microsoft Office because other languages cause encoding issues resulting in unpredictable behavior.

(*See* https://docs.alfresco.com/transformation-engine/latest/install/)

The following diagram shows a simple representation of the Transform Service components:



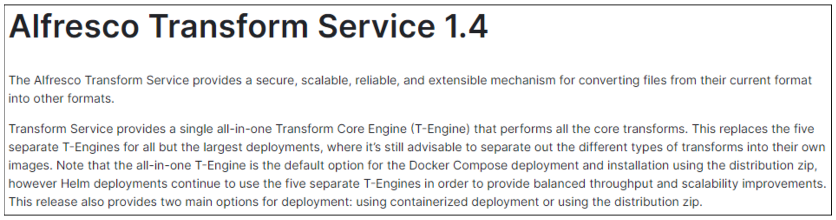
(*See* https://docs.alfresco.com/transform-service/1.4/admin/)



```java
public class SelectingTransformer implements Transformer
{
    private static final String ID = "misc";

    public static final String LICENCE =
            "This transformer uses libraries from Apache. See the license at http://www.apache.org/licenses/LICENSE-2.0. or in /Apache\\\\ 2.0.txt\\n" +
            "Additional libraries used:\n" +
            "* htmlparser http://htmlparser.sourceforge.net/license.html";

    private final Map<String, SelectableTransformer> transformers = ImmutableMap
        .<String, SelectableTransformer>builder()
        .put("appleIWorks", new AppleIWorksContentTransformer())
        .put("html", new HtmlParserContentTransformer())
        .put("string", new StringExtractingContentTransformer())
        .put("textToPdf", new TextToPdfContentTransformer())
        .put("rfc822", new EMLTransformer())
        .put("ooXmlThumbnail", new OOXMLThumbnailContentTransformer())
        .put("HtmlMetadataExtractor", new HtmlMetadataExtractor())
        .put("RFC822MetadataExtractor", new RFC822MetadataExtractor())
        .build();

    @Override
    public String getTransformerId()
    {
        return ID;
    }

    @Override
    public void transform(String transformName, String sourceMimetype, String targetMimetype,
                          Map<String, String> transformOptions,
                          File sourceFile, File targetFile) throws Exception
    {
        final SelectableTransformer transformer = transformers.get(transformName);
        logOptions(sourceFile, targetFile, transformOptions);
        transformer.transform(sourceMimetype, targetMimetype, transformOptions, sourceFile, targetFile);
    }
```

(*See* https://github.com/Alfresco/alfresco-transform-core/blob/master/alfresco-transform-misc/alfresco-transform-misc/src/main/java/org/alfresco/transformer/transformers/SelectingTransformer.java)

103. The Accused Products perform a method that includes *receiving an input data stream of file data at a physical input associated with a first input thread of the plurality of input threads*. For example, the Alfresco Transform Service includes T-Engine/Transform Engines which "transforms files referenced by the repository and retrieved from the shared file store". The shared file store "is used as temporary storage for the original source files (stored by the repository)."



Alfresco Transform Service 1.4

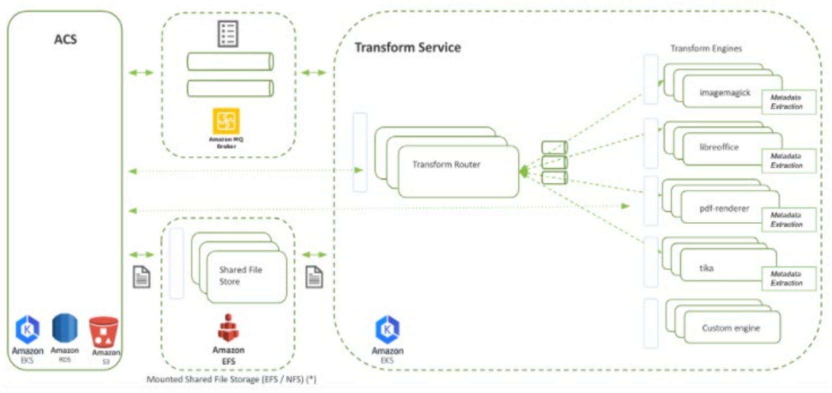
The Alfresco Transform Service provides a secure, scalable, reliable, and extensible mechanism for converting files from their current format into other formats.

Transform Service provides a single all-in-one Transform Core Engine (T-Engine) that performs all the core transforms. This replaces the five separate T-Engines for all but the largest deployments, where it's still advisable to separate out the different types of transforms into their own images. Note that the all-in-one T-Engine is the default option for the Docker Compose deployment and installation using the distribution zip, however Helm deployments continue to use the five separate T-Engines in order to provide balanced throughput and scalability improvements. This release also provides two main options for deployment: using containerized deployment or using the distribution zip.

(*See* https://docs.alfresco.com/transform-service/1.4/.)

- **Transform Engines**: The Transform Engines transform files referenced by the repository and retrieved from the shared file store. Here are some example transformations for each Transform Engine (this is not an exhaustive list):
  - LibreOffice (e.g. docx to pdf)
  - ImageMagick (e.g. resize)
  - Alfresco PDF Renderer (e.g. pdf to png)
  - Tika (e.g. docx to plain text)
  - Misc. (not included in diagram)
- **Shared File Store**: This is used as temporary storage for the original source file (stored by the repository), intermediate files for multi-step transforms, and the final transformed target file. The target file is retrieved by the repository after it's been processed by one or more of the Transform Engines.

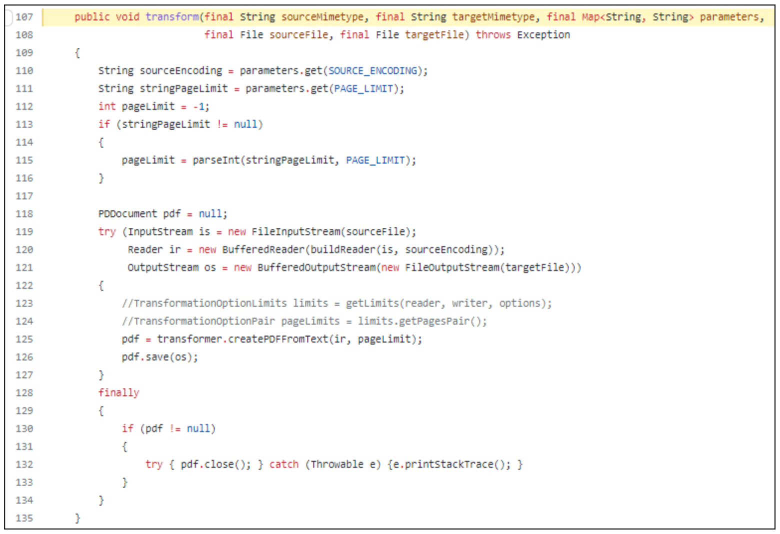
The following diagram shows a simple representation of the Transform Service components:



Note that from Transform Service version 1.3.2 the metadata extraction that usually takes part in the core repository legacy transform engines has now been lifted out into the separate transform engine processes. This enables scaling of the metadata extraction.

(*See* https://docs.alfresco.com/transform-service/1.4/admin/)

104. In another example, the Alfresco Transform Service receives a text file "sourceFile" from "FileInputStream" and transforms the text file into a PDF file.

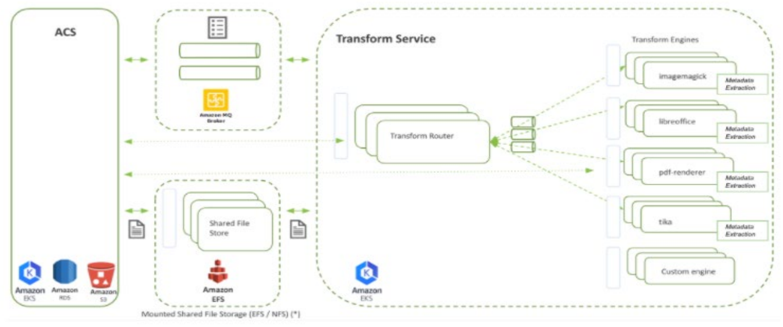


```
    public void transform(final String sourceMimetype, final String targetMimetype, final Map<String, String> parameters,
                          final File sourceFile, final File targetFile) throws Exception
    {
        String sourceEncoding = parameters.get(SOURCE_ENCODING);
        String stringPageLimit = parameters.get(PAGE_LIMIT);
        int pageLimit = -1;
        if (stringPageLimit != null)
        {
            pageLimit = parseInt(stringPageLimit, PAGE_LIMIT);
        }

        PDDocument pdf = null;
        try (InputStream is = new FileInputStream(sourceFile);
             Reader ir = new BufferedReader(buildReader(is, sourceEncoding));
             OutputStream os = new BufferedOutputStream(new FileOutputStream(targetFile)))
        {
            //TransformationOptionLimits limits = getLimits(reader, writer, options);
            //TransformationOptionPair pageLimits = limits.getPagesPair();
            pdf = transformer.createPDFFromText(ir, pageLimit);
            pdf.save(os);
        }
        finally
        {
            if (pdf != null)
            {
                try { pdf.close(); } catch (Throwable e) {e.printStackTrace(); }
            }
        }
    }
```

(*See* https://github.com/Alfresco/alfresco-transform-core/blob/e575ec943a5fa5dddca5593e6795a17a2bbb3cb6/alfresco-transform-misc/alfresco-transform-misc/src/main/java/org/alfresco/transformer/transformers/TextToPdfContentTransformer.java)

105. As shown in the screenshot below, the Alfresco Transform Service includes a plurality of input threads associated with Transform Engines and Metadata Extractor pipelines.

The following diagram shows a simple representation of the Transform Service components:

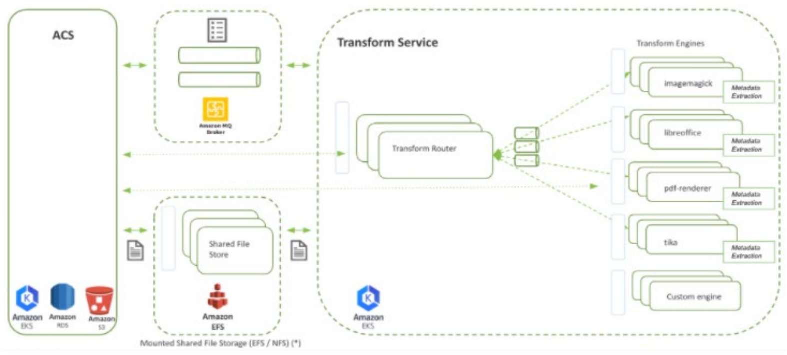


Note that from Transform Service version 1.3.2 the metadata extraction that usually takes part in the core repository legacy transform engines has now been lifted out into the separate transform engine processes. This enables scaling of the metadata extraction.

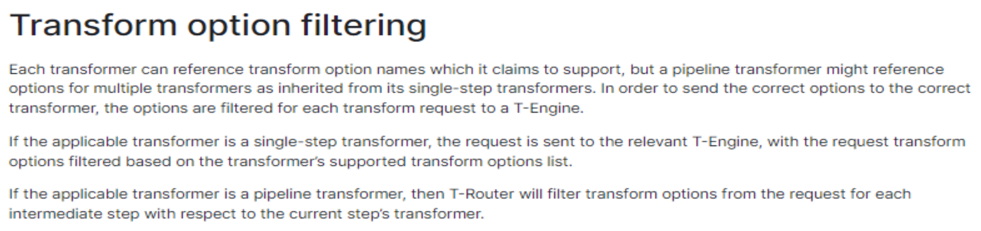
(*See* https://docs.alfresco.com/transform-service/1.4/admin/)

106. On information and belief, the Accused Products perform a method that includes *producing filtered data from the input data stream by passing the input data stream through a filter associated with the first input thread*. For example, the Alfresco Transform Service includes a transform router that filters transform options from the transform requests and select the proper Transform Engine to perform the content transformation.

The following diagram shows a simple representation of the Transform Service components:



(*See* https://docs.alfresco.com/transform-service/1.4/admin/)



Transform option filtering

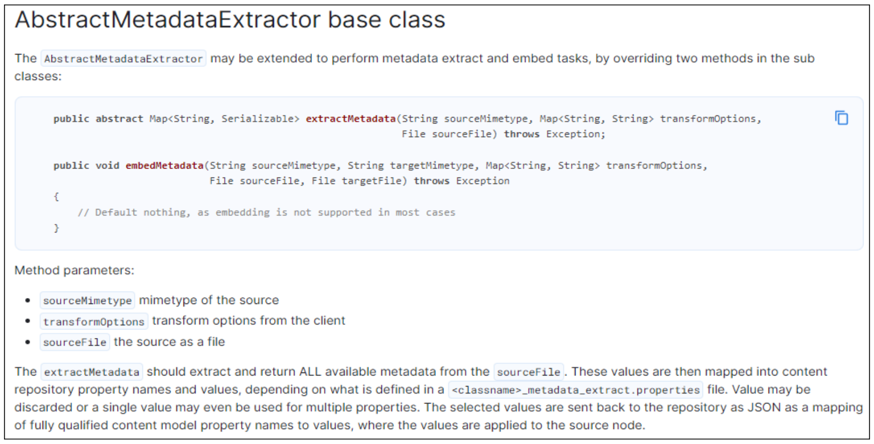
Each transformer can reference transform option names which it claims to support, but a pipeline transformer might reference options for multiple transformers as inherited from its single-step transformers. In order to send the correct options to the correct transformer, the options are filtered for each transform request to a T-Engine.

If the applicable transformer is a single-step transformer, the request is sent to the relevant T-Engine, with the request transform options filtered based on the transformer's supported transform options list.

If the applicable transformer is a pipeline transformer, then T-Router will filter transform options from the request for each intermediate step with respect to the current step's transformer.

(*See* https://docs.alfresco.com/transform-service/1.4/config/transformers/)

107. In addition, the Alfresco Transform Service defines a base class "AbstractMetadataExtractor" which includes a "<classname>_metadata_extract.properties" file that defines a filter to produce filtered data (e.g., metadata) extracted from input data stream (e.g., input file). Values added in the "<classname>_metadata_extract.properties" file are extracted from the input file. For example, "RFC822MetadataExtractor_metadata_extract.properties" file lists values that are extracted from the input file, e.g., messageFrom, messageTo, messageCc, messageSubject, messageSent, messageReceived, Thread-Index, and Message-ID.



AbstractMetadataExtractor base class

The `AbstractMetadataExtractor` may be extended to perform metadata extract and embed tasks, by overriding two methods in the sub classes:

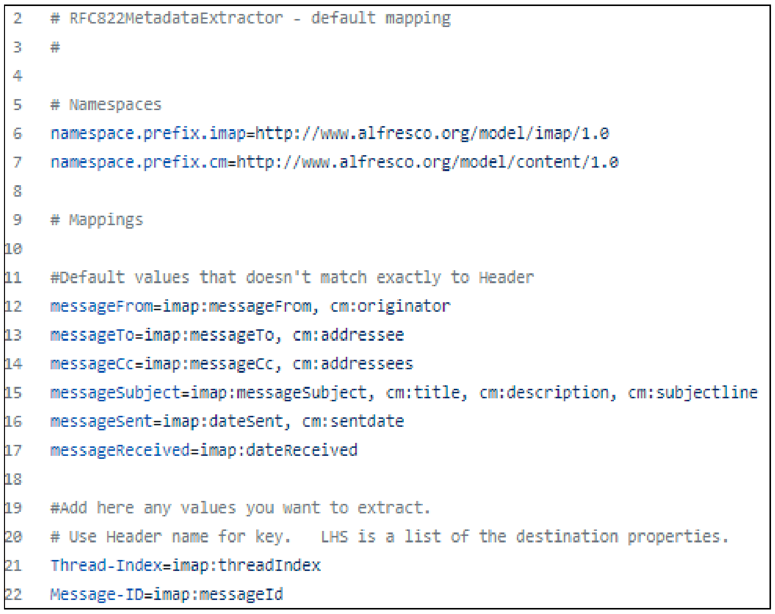
```
    public abstract Map<String, Serializable> extractMetadata(String sourceMimetype, Map<String, String> transformOptions,
                                                              File sourceFile) throws Exception;

    public void embedMetadata(String sourceMimetype, String targetMimetype, Map<String, String> transformOptions,
                              File sourceFile, File targetFile) throws Exception
    {
        // Default nothing, as embedding is not supported in most cases
    }
```

Method parameters:

- `sourceMimetype` mimetype of the source
- `transformOptions` transform options from the client
- `sourceFile` the source as a file

The `extractMetadata` should extract and return ALL available metadata from the `sourceFile`. These values are then mapped into content repository property names and values, depending on what is defined in a `<classname>_metadata_extract.properties` file. Value may be discarded or a single value may even be used for multiple properties. The selected values are sent back to the repository as JSON as a mapping of fully qualified content model property names to values, where the values are applied to the source node.

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/#introduction)

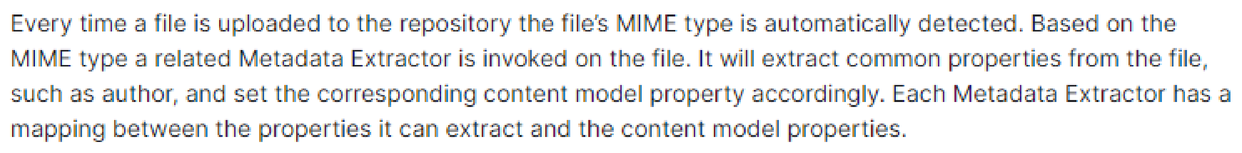


```
# RFC822MetadataExtractor - default mapping
#

# Namespaces
namespace.prefix.imap=http://www.alfresco.org/model/imap/1.0
namespace.prefix.cm=http://www.alfresco.org/model/content/1.0

# Mappings

#Default values that doesn't match exactly to Header
messageFrom=imap:messageFrom, cm:originator
messageTo=imap:messageTo, cm:addressee
messageCc=imap:messageCc, cm:addressees
messageSubject=imap:messageSubject, cm:title, cm:description, cm:subjectline
messageSent=imap:dateSent, cm:sentdate
messageReceived=imap:dateReceived

#Add here any values you want to extract.
# Use Header name for key.   LHS is a list of the destination properties.
Thread-Index=imap:threadIndex
Message-ID=imap:messageId
```

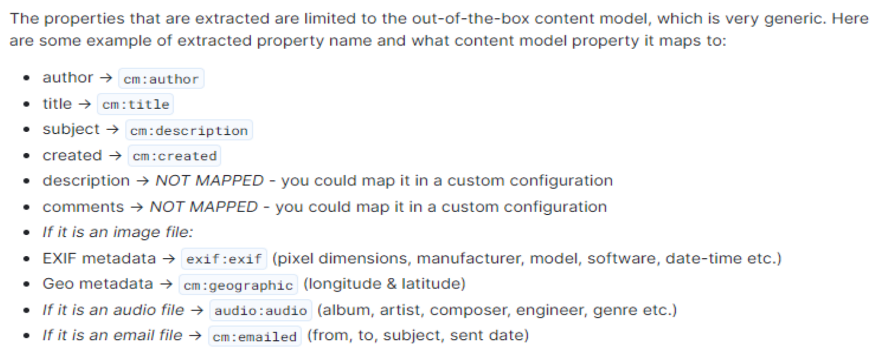
(*See* https://github.com/Alfresco/alfresco-transform-core/blob/master/alfresco-transform-misc/alfresco-transform-misc/src/main/resources/RFC822MetadataExtractor_metadata_extract.properties)

108. On information and belief, the Accused Products perform a method that includes *identifying, by an agent associated with the first input thread, each event in the*

filtered data, wherein the agent creates a message for each event containing text from the event and a generic data structure corresponding to the event for a thread job manager associated with the first input thread. For example, the Alfresco Transform Service's Metadata Extractor identifies and extracts metadata from source files, where the extracted metadata stores common properties, such as author, title, subject, etc. in a generic data structure by mapping the common properties to content model properties as name value pairs.

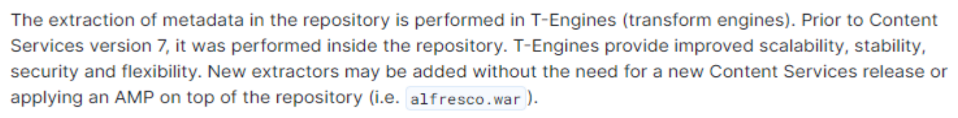

Every time a file is uploaded to the repository the file's MIME type is automatically detected. Based on the MIME type a related Metadata Extractor is invoked on the file. It will extract common properties from the file, such as author, and set the corresponding content model property accordingly. Each Metadata Extractor has a mapping between the properties it can extract and the content model properties.

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)

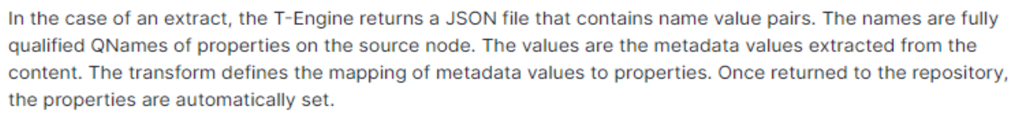

The properties that are extracted are limited to the out-of-the-box content model, which is very generic. Here are some example of extracted property name and what content model property it maps to:

- author → `cm:author`
- title → `cm:title`
- subject → `cm:description`
- created → `cm:created`
- description → *NOT MAPPED* - you could map it in a custom configuration
- comments → *NOT MAPPED* - you could map it in a custom configuration
- *If it is an image file:*
- EXIF metadata → `exif:exif` (pixel dimensions, manufacturer, model, software, date-time etc.)
- Geo metadata → `cm:geographic` (longitude & latitude)
- *If it is an audio file* → `audio:audio` (album, artist, composer, engineer, genre etc.)
- *If it is an email file* → `cm:emailed` (from, to, subject, sent date)

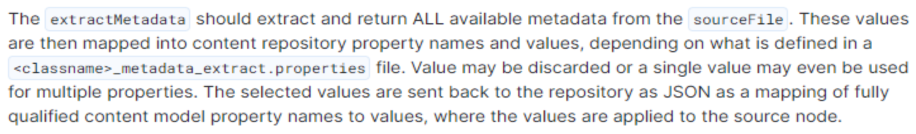
(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)


The extraction of metadata in the repository is performed in T-Engines (transform engines). Prior to Content Services version 7, it was performed inside the repository. T-Engines provide improved scalability, stability, security and flexibility. New extractors may be added without the need for a new Content Services release or applying an AMP on top of the repository (i.e. `alfresco.war`).

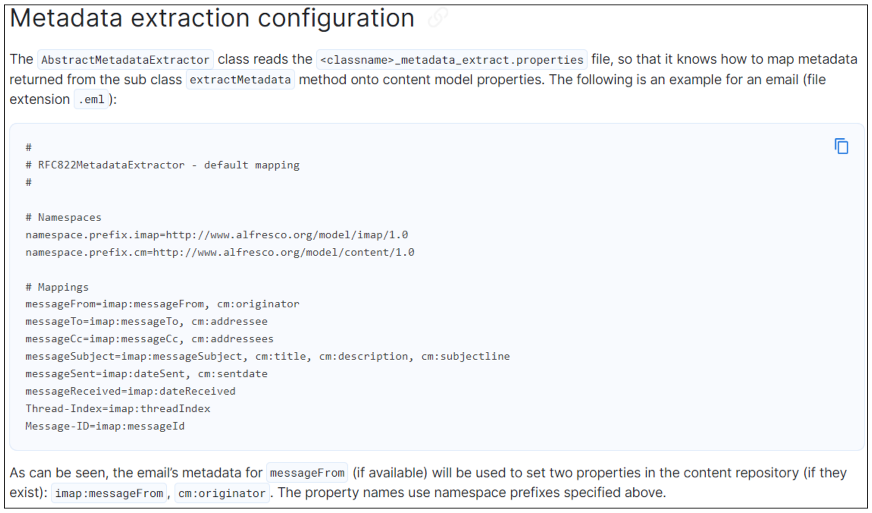
(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)


In the case of an extract, the T-Engine returns a JSON file that contains name value pairs. The names are fully qualified QNames of properties on the source node. The values are the metadata values extracted from the content. The transform defines the mapping of metadata values to properties. Once returned to the repository, the properties are automatically set.

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)


The `extractMetadata` should extract and return ALL available metadata from the `sourceFile`. These values are then mapped into content repository property names and values, depending on what is defined in a `<classname>_metadata_extract.properties` file. Value may be discarded or a single value may even be used for multiple properties. The selected values are sent back to the repository as JSON as a mapping of fully qualified content model property names to values, where the values are applied to the source node.

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)

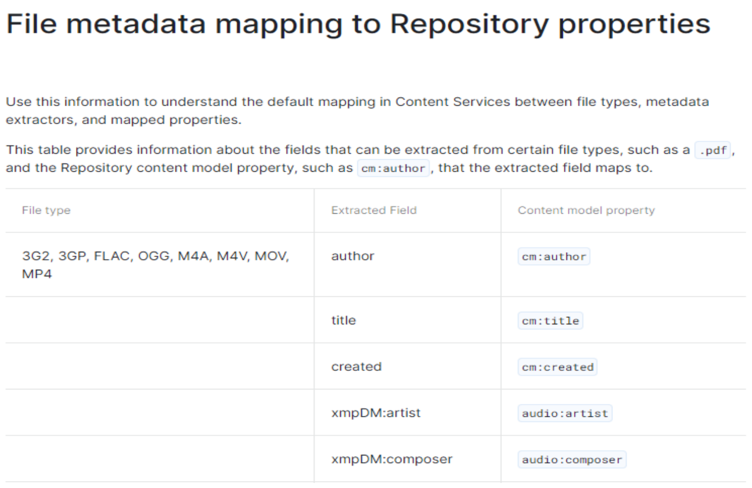

Metadata extraction configuration

The `AbstractMetadataExtractor` class reads the `<classname>_metadata_extract.properties` file, so that it knows how to map metadata returned from the sub class `extractMetadata` method onto content model properties. The following is an example for an email (file extension `.eml`):

```
#
# RFC822MetadataExtractor - default mapping
#

# Namespaces
namespace.prefix.imap=http://www.alfresco.org/model/imap/1.0
namespace.prefix.cm=http://www.alfresco.org/model/content/1.0

# Mappings
messageFrom=imap:messageFrom, cm:originator
messageTo=imap:messageTo, cm:addressee
messageCc=imap:messageCc, cm:addressees
messageSubject=imap:messageSubject, cm:title, cm:description, cm:subjectline
messageSent=imap:dateSent, cm:sentdate
messageReceived=imap:dateReceived
Thread-Index=imap:threadIndex
Message-ID=imap:messageId
```

As can be seen, the email's metadata for `messageFrom` (if available) will be used to set two properties in the content repository (if they exist): `imap:messageFrom`, `cm:originator`. The property names use namespace prefixes specified above.

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)



File metadata mapping to Repository properties

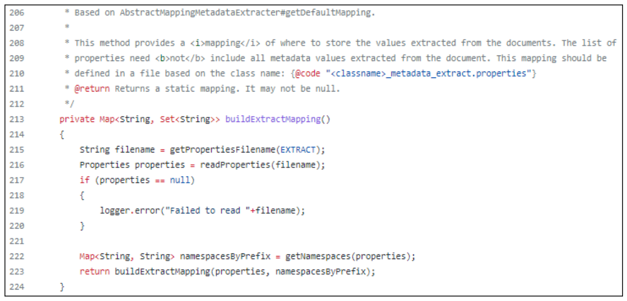
Use this information to understand the default mapping in Content Services between file types, metadata extractors, and mapped properties.

This table provides information about the fields that can be extracted from certain file types, such as a `.pdf`, and the Repository content model property, such as `cm:author`, that the extracted field maps to.

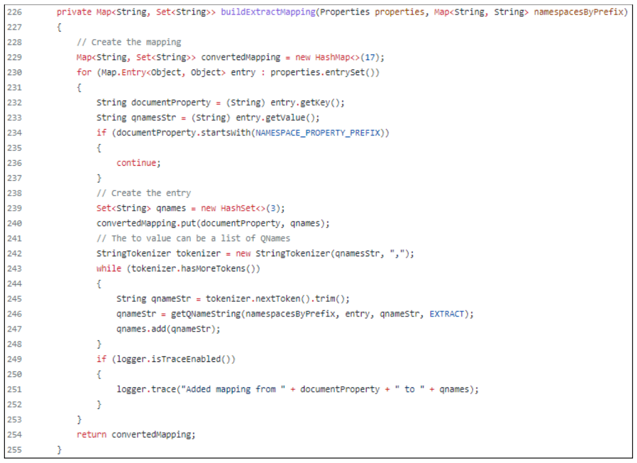
File type	Extracted Field	Content model property
3G2, 3GP, FLAC, OGG, M4A, M4V, MOV, MP4	author	`cm:author`
	title	`cm:title`
	created	`cm:created`
	xmpDM:artist	`audio:artist`
	xmpDM:composer	`audio:composer`

(*See* https://docs.alfresco.com/content-services/latest/admin/metadata-extraction/)

109. In another example, the Alfresco Transform Service implements a "buildExtractMapping" method to extract property values from an input file and store the values in "Map" data structure.



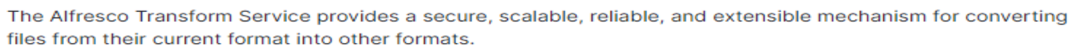
```
     * Based on AbstractMappingMetadataExtracter#getDefaultMapping.
     *
     * This method provides a <i>mapping</i> of where to store the values extracted from the documents. The list of
     * properties need <b>not</b> include all metadata values extracted from the document. This mapping should be
     * defined in a file based on the class name: {@code "<classname>_metadata_extract.properties"}
     * @return Returns a static mapping. It may not be null.
     */
    private Map<String, Set<String>> buildExtractMapping()
    {
        String filename = getPropertiesFilename(EXTRACT);
        Properties properties = readProperties(filename);
        if (properties == null)
        {
            logger.error("Failed to read "+filename);
        }

        Map<String, String> namespacesByPrefix = getNamespaces(properties);
        return buildExtractMapping(properties, namespacesByPrefix);
    }
```



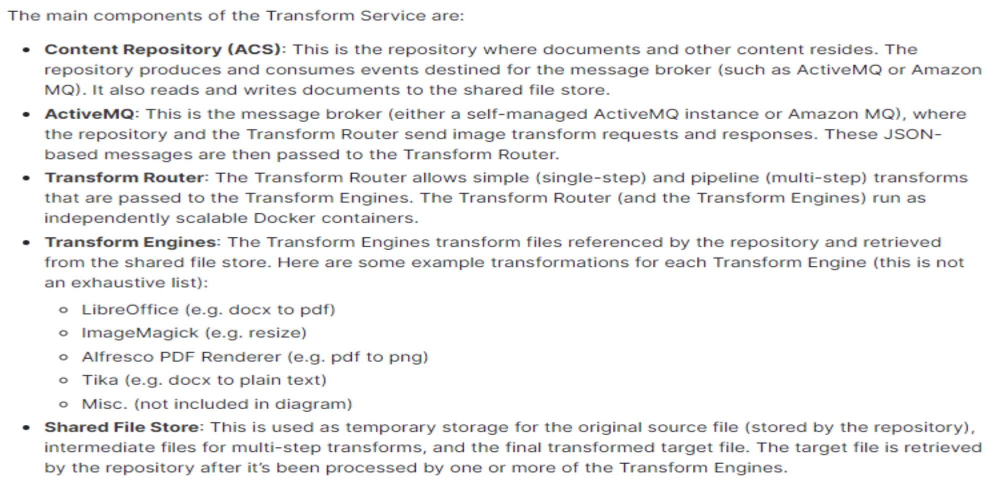
```
    private Map<String, Set<String>> buildExtractMapping(Properties properties, Map<String, String> namespacesByPrefix)
    {
        // Create the mapping
        Map<String, Set<String>> convertedMapping = new HashMap<>(17);
        for (Map.Entry<Object, Object> entry : properties.entrySet())
        {
            String documentProperty = (String) entry.getKey();
            String qnamesStr = (String) entry.getValue();
            if (documentProperty.startsWith(NAMESPACE_PROPERTY_PREFIX))
            {
                continue;
            }
            // Create the entry
            Set<String> qnames = new HashSet<>(3);
            convertedMapping.put(documentProperty, qnames);
            // The to value can be a list of QNames
            StringTokenizer tokenizer = new StringTokenizer(qnamesStr, ",");
            while (tokenizer.hasMoreTokens())
            {
                String qnameStr = tokenizer.nextToken().trim();
                qnameStr = getQNameString(namespacesByPrefix, entry, qnameStr, EXTRACT);
                qnames.add(qnameStr);
            }
            if (logger.isTraceEnabled())
            {
                logger.trace("Added mapping from " + documentProperty + " to " + qnames);
            }
        }
        return convertedMapping;
    }
```

(*See* https://github.com/Alfresco/alfresco-transform-core/blob/e575ec943a5fa5dddca5593e6795a17a2bbb3cb6/alfresco-transformer-base/src/main/java/org/alfresco/transformer/metadataExtractors/AbstractMetadataExtractor.java)

110. The Accused Products perform a method that includes *creating, by the thread job manager associated with the first input thread, a process for transforming the messages according to the generic data structure into output data*. For example, the Alfresco Transform Service provides "a secure, scalable, reliable, and extensible mechanism for converting files from their current format into other formats." The Transform Engines perform transformation for conversion of files from their current format into other format, e.g., docx to pdf, pdf to png, and docx to plain text.


The Alfresco Transform Service provides a secure, scalable, reliable, and extensible mechanism for converting files from their current format into other formats.

(*See* https://docs.alfresco.com/transform-service/latest/)

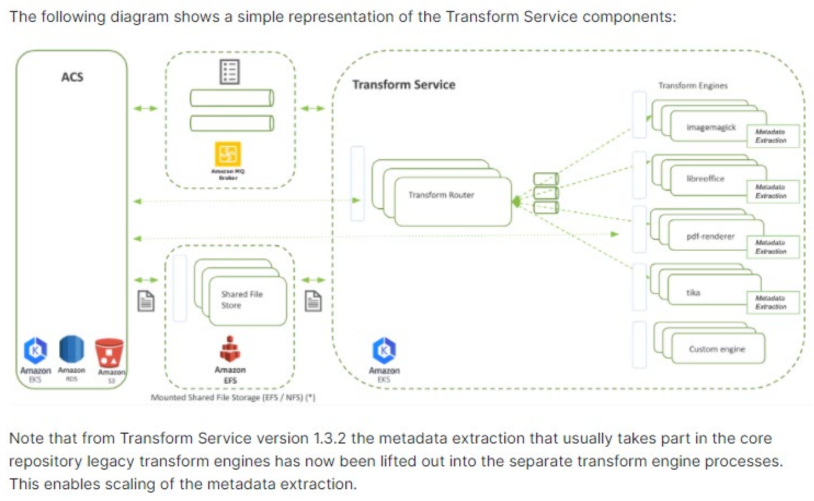

The main components of the Transform Service are:

- **Content Repository (ACS)**: This is the repository where documents and other content resides. The repository produces and consumes events destined for the message broker (such as ActiveMQ or Amazon MQ). It also reads and writes documents to the shared file store.
- **ActiveMQ**: This is the message broker (either a self-managed ActiveMQ instance or Amazon MQ), where the repository and the Transform Router send image transform requests and responses. These JSON-based messages are then passed to the Transform Router.
- **Transform Router**: The Transform Router allows simple (single-step) and pipeline (multi-step) transforms that are passed to the Transform Engines. The Transform Router (and the Transform Engines) run as independently scalable Docker containers.
- **Transform Engines**: The Transform Engines transform files referenced by the repository and retrieved from the shared file store. Here are some example transformations for each Transform Engine (this is not an exhaustive list):
    - LibreOffice (e.g. docx to pdf)
    - ImageMagick (e.g. resize)
    - Alfresco PDF Renderer (e.g. pdf to png)
    - Tika (e.g. docx to plain text)
    - Misc. (not included in diagram)
- **Shared File Store**: This is used as temporary storage for the original source file (stored by the repository), intermediate files for multi-step transforms, and the final transformed target file. The target file is retrieved by the repository after it's been processed by one or more of the Transform Engines.

(*See* https://docs.alfresco.com/transform-service/1.4/admin/)

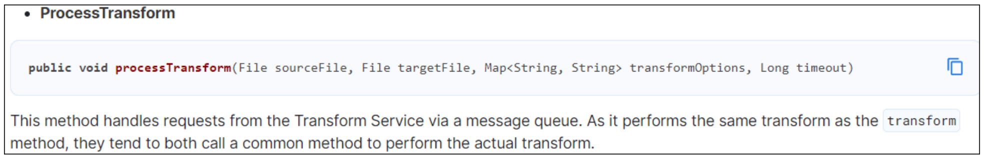
111. As shown in the diagram below, the Alfresco Transform Service includes

a message broker "ActiveMQ" and transform routers that create processes for identifying and extracting metadata from source files and transforming the properties of the files according to the content model into target files. The message broker "ActiveMQ" builds and maintains a message queue. A "processTransform" method is implemented in the Alfresco Transform Service to "handle[] requests from the Transform Service via a message queue ".


The following diagram shows a simple representation of the Transform Service components:

Note that from Transform Service version 1.3.2 the metadata extraction that usually takes part in the core repository legacy transform engines has now been lifted out into the separate transform engine processes. This enables scaling of the metadata extraction.

(*See* https://docs.alfresco.com/transform-service/1.4/admin/)


- **ProcessTransform**

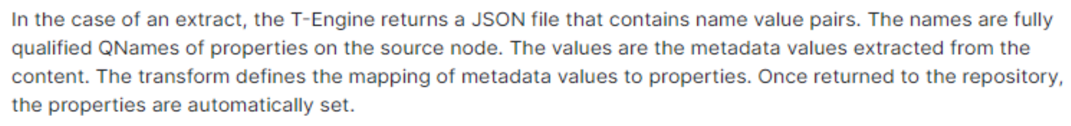
```
public void processTransform(File sourceFile, File targetFile, Map<String, String> transformOptions, Long timeout)
```

This method handles requests from the Transform Service via a message queue. As it performs the same transform as the `transform` method, they tend to both call a common method to perform the actual transform.

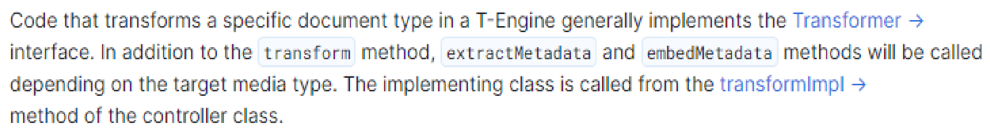
(*See* https://docs.alfresco.com/transform-service/1.4/config/engine/)

112. As an example, the Transform Engines perform metadata extraction by calling "extractMetadata" function, where the "extractMetadata" function extracts and returns "ALL available metadata from the sourceFile." "In the case of an extract, the T-Engine returns a JSON file that contains name value pairs" of the extracted

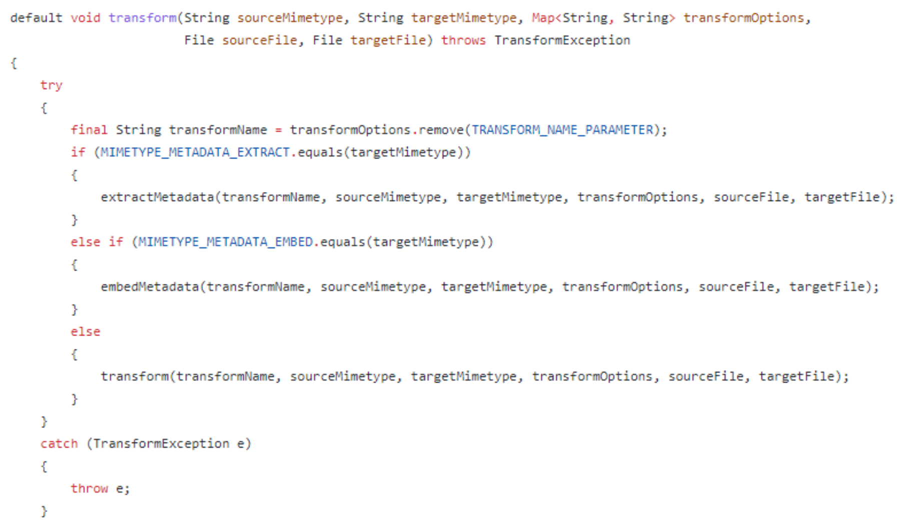
metadata and embeds the extracted metadata into the output data.


In the case of an extract, the T-Engine returns a JSON file that contains name value pairs. The names are fully qualified QNames of properties on the source node. The values are the metadata values extracted from the content. The transform defines the mapping of metadata values to properties. Once returned to the repository, the properties are automatically set.

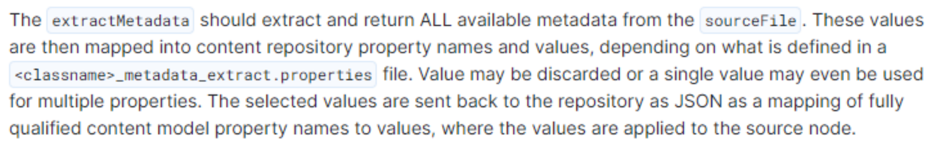
(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)


Code that transforms a specific document type in a T-Engine generally implements the Transformer → interface. In addition to the `transform` method, `extractMetadata` and `embedMetadata` methods will be called depending on the target media type. The implementing class is called from the transformImpl → method of the controller class.

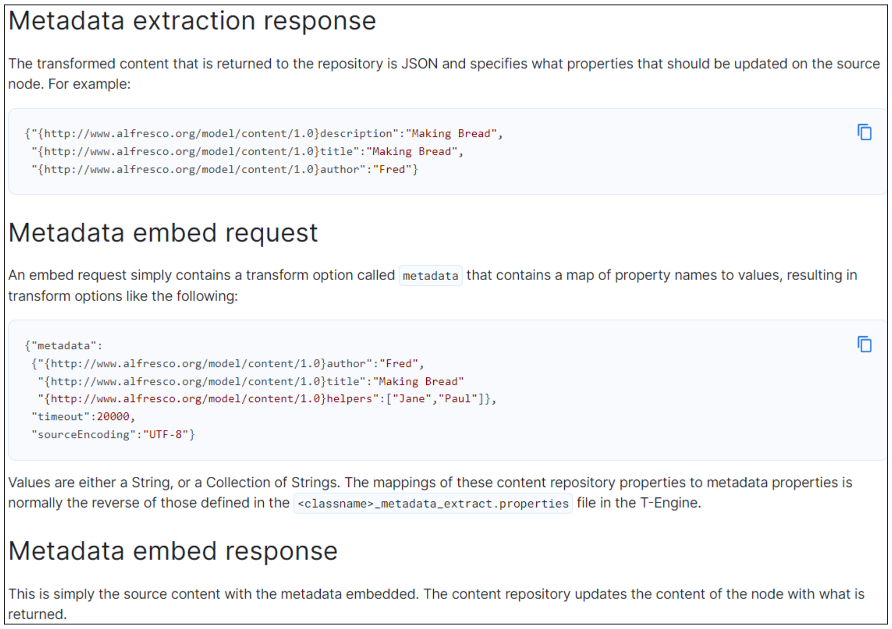
(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)


```
default void transform(String sourceMimetype, String targetMimetype, Map<String, String> transformOptions,
                       File sourceFile, File targetFile) throws TransformException
{
    try
    {
        final String transformName = transformOptions.remove(TRANSFORM_NAME_PARAMETER);
        if (MIMETYPE_METADATA_EXTRACT.equals(targetMimetype))
        {
            extractMetadata(transformName, sourceMimetype, targetMimetype, transformOptions, sourceFile, targetFile);
        }
        else if (MIMETYPE_METADATA_EMBED.equals(targetMimetype))
        {
            embedMetadata(transformName, sourceMimetype, targetMimetype, transformOptions, sourceFile, targetFile);
        }
        else
        {
            transform(transformName, sourceMimetype, targetMimetype, transformOptions, sourceFile, targetFile);
        }
    }
    catch (TransformException e)
    {
        throw e;
    }
```

(*See* https://github.com/Alfresco/alfresco-transform-core/blob/master/alfresco-transformer-base/src/main/java/org/alfresco/transformer/executors/Transformer.java)


The `extractMetadata` should extract and return ALL available metadata from the `sourceFile`. These values are then mapped into content repository property names and values, depending on what is defined in a `<classname>_metadata_extract.properties` file. Value may be discarded or a single value may even be used for multiple properties. The selected values are sent back to the repository as JSON as a mapping of fully qualified content model property names to values, where the values are applied to the source node.

(*See* https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)



Metadata extraction response

The transformed content that is returned to the repository is JSON and specifies what properties that should be updated on the source node. For example:

```
{"{http://www.alfresco.org/model/content/1.0}description":"Making Bread",
 "{http://www.alfresco.org/model/content/1.0}title":"Making Bread",
 "{http://www.alfresco.org/model/content/1.0}author":"Fred"}
```

Metadata embed request

An embed request simply contains a transform option called `metadata` that contains a map of property names to values, resulting in transform options like the following:

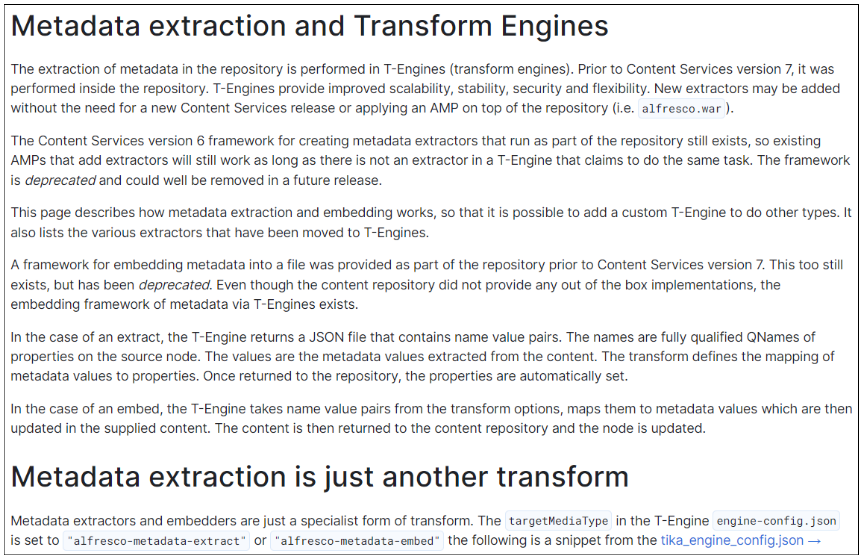
```
{"metadata":
 {"{http://www.alfresco.org/model/content/1.0}author":"Fred",
  "{http://www.alfresco.org/model/content/1.0}title":"Making Bread"
  "{http://www.alfresco.org/model/content/1.0}helpers":["Jane","Paul"]},
 "timeout":20000,
 "sourceEncoding":"UTF-8"}
```

Values are either a String, or a Collection of Strings. The mappings of these content repository properties to metadata properties is normally the reverse of those defined in the `<classname>_metadata_extract.properties` file in the T-Engine.

Metadata embed response

This is simply the source content with the metadata embedded. The content repository updates the content of the node with what is returned.

(See https://docs.alfresco.com/content-services/latest/develop/repo-ext-points/metadata-extractors/)



Metadata extraction and Transform Engines

The extraction of metadata in the repository is performed in T-Engines (transform engines). Prior to Content Services version 7, it was performed inside the repository. T-Engines provide improved scalability, stability, security and flexibility. New extractors may be added without the need for a new Content Services release or applying an AMP on top of the repository (i.e. `alfresco.war`).

The Content Services version 6 framework for creating metadata extractors that run as part of the repository still exists, so existing AMPs that add extractors will still work as long as there is not an extractor in a T-Engine that claims to do the same task. The framework is *deprecated* and could well be removed in a future release.

This page describes how metadata extraction and embedding works, so that it is possible to add a custom T-Engine to do other types. It also lists the various extractors that have been moved to T-Engines.

A framework for embedding metadata into a file was provided as part of the repository prior to Content Services version 7. This too still exists, but has been *deprecated*. Even though the content repository did not provide any out of the box implementations, the embedding framework of metadata via T-Engines exists.

In the case of an extract, the T-Engine returns a JSON file that contains name value pairs. The names are fully qualified QNames of properties on the source node. The values are the metadata values extracted from the content. The transform defines the mapping of metadata values to properties. Once returned to the repository, the properties are automatically set.

In the case of an embed, the T-Engine takes name value pairs from the transform options, maps them to metadata values which are then updated in the supplied content. The content is then returned to the content repository and the node is updated.

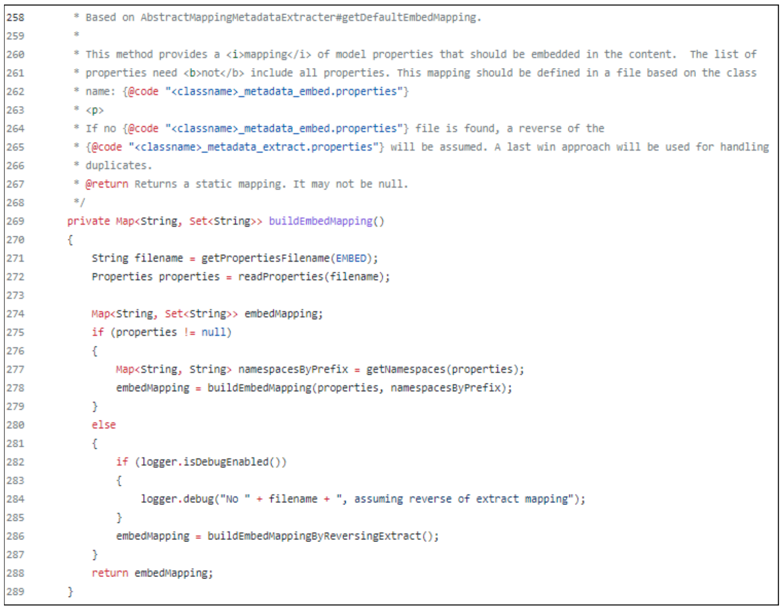
Metadata extraction is just another transform

Metadata extractors and embedders are just a specialist form of transform. The `targetMediaType` in the T-Engine `engine-config.json` is set to `"alfresco-metadata-extract"` or `"alfresco-metadata-embed"` the following is a snippet from the tika_engine_config.json →

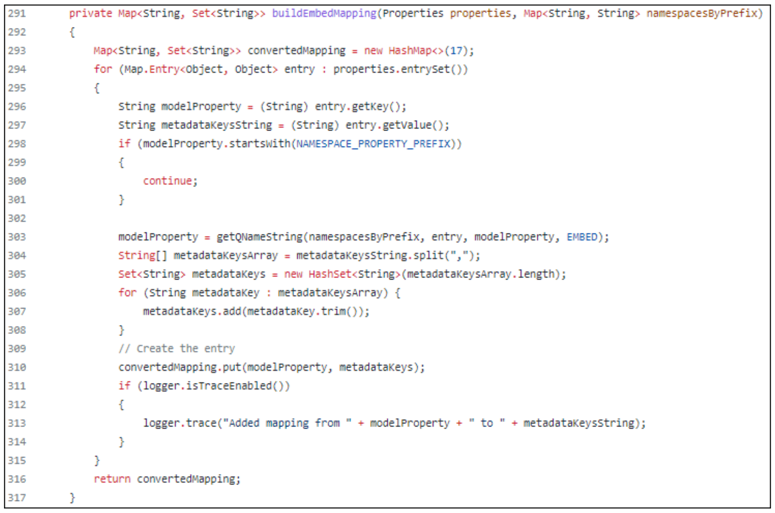
(*See* https://docs.alfresco.com/content-services/community/develop/repo-ext-points/metadata-extractors/)

113. As another example, as shown below, the Alfresco Transform Service

implements "buildEmbedMapping" method to provide mappings of model properties to metadata and then to embed the metadata into the content of target file.



```
     * Based on AbstractMappingMetadataExtracter#getDefaultEmbedMapping.
     *
     * This method provides a <i>mapping</i> of model properties that should be embedded in the content.  The list of
     * properties need <b>not</b> include all properties. This mapping should be defined in a file based on the class
     * name: {@code "<classname>_metadata_embed.properties"}
     * <p>
     * If no {@code "<classname>_metadata_embed.properties"} file is found, a reverse of the
     * {@code "<classname>_metadata_extract.properties"} will be assumed. A last win approach will be used for handling
     * duplicates.
     * @return Returns a static mapping. It may not be null.
     */
    private Map<String, Set<String>> buildEmbedMapping()
    {
        String filename = getPropertiesFilename(EMBED);
        Properties properties = readProperties(filename);

        Map<String, Set<String>> embedMapping;
        if (properties != null)
        {
            Map<String, String> namespacesByPrefix = getNamespaces(properties);
            embedMapping = buildEmbedMapping(properties, namespacesByPrefix);
        }
        else
        {
            if (logger.isDebugEnabled())
            {
                logger.debug("No " + filename + ", assuming reverse of extract mapping");
            }
            embedMapping = buildEmbedMappingByReversingExtract();
        }
        return embedMapping;
    }
```

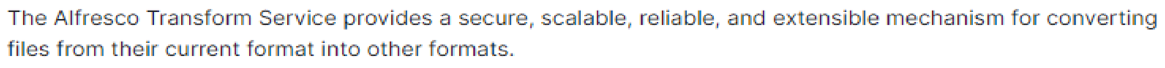


```
    private Map<String, Set<String>> buildEmbedMapping(Properties properties, Map<String, String> namespacesByPrefix)
    {
        Map<String, Set<String>> convertedMapping = new HashMap<>(17);
        for (Map.Entry<Object, Object> entry : properties.entrySet())
        {
            String modelProperty = (String) entry.getKey();
            String metadataKeysString = (String) entry.getValue();
            if (modelProperty.startsWith(NAMESPACE_PROPERTY_PREFIX))
            {
                continue;
            }

            modelProperty = getQNameString(namespacesByPrefix, entry, modelProperty, EMBED);
            String[] metadataKeysArray = metadataKeysString.split(",");
            Set<String> metadataKeys = new HashSet<String>(metadataKeysArray.length);
            for (String metadataKey : metadataKeysArray) {
                metadataKeys.add(metadataKey.trim());
            }
            // Create the entry
            convertedMapping.put(modelProperty, metadataKeys);
            if (logger.isTraceEnabled())
            {
                logger.trace("Added mapping from " + modelProperty + " to " + metadataKeysString);
            }
        }
        return convertedMapping;
    }
```

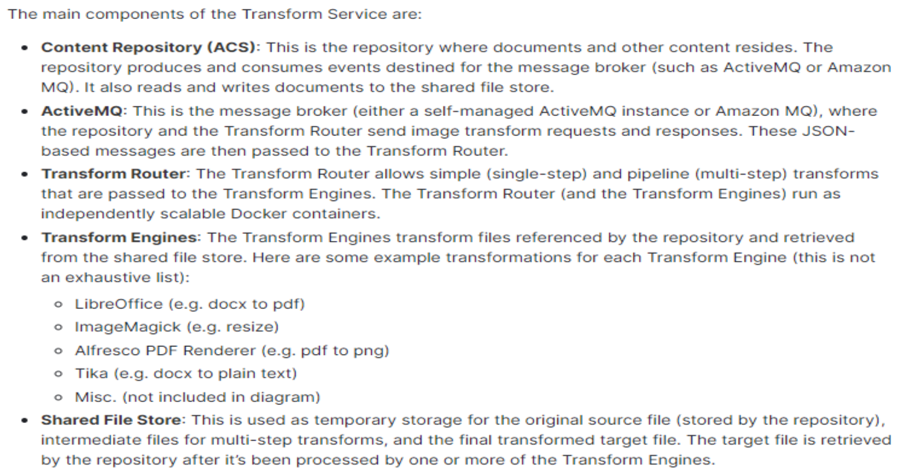
(See https://github.com/Alfresco/alfresco-transform-core/blob/e575ec943a5fa5dddca5593e6795a17a2bbb3cb6/alfresco-transformer-base/src/main/java/org/alfresco/transformer/metadataExtractors/AbstractMetadataExtractor.java)

114. The Accused Products perform a method wherein *input data in the first data stream is of a first document format and the output data is of a second document*

format. For example, the Alfresco Transform Service performs transformation for conversion of files from their current format into other format, e.g., docx to pdf, pdf to png, and docx to plain text.


The Alfresco Transform Service provides a secure, scalable, reliable, and extensible mechanism for converting files from their current format into other formats.

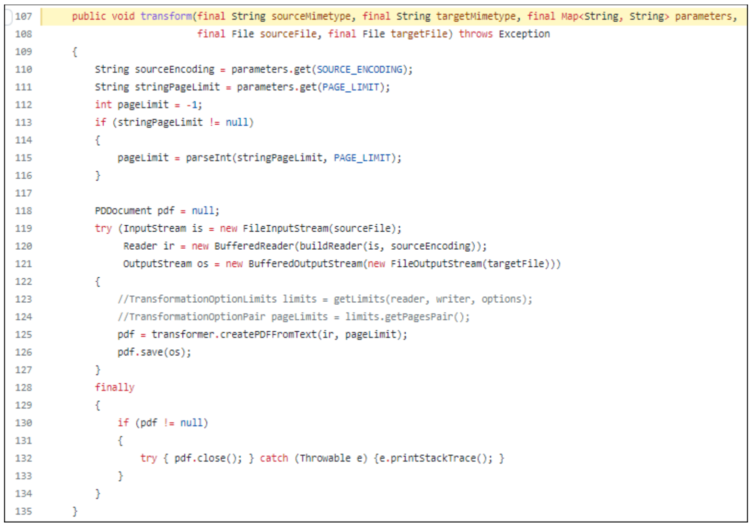
(*See* https://docs.alfresco.com/transform-service/latest/)

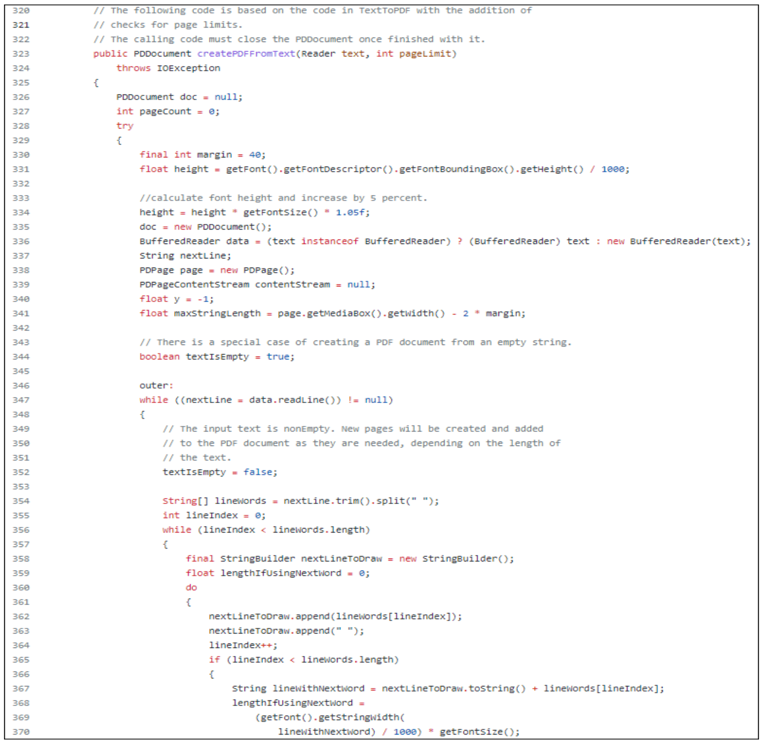

The main components of the Transform Service are:

- **Content Repository (ACS)**: This is the repository where documents and other content resides. The repository produces and consumes events destined for the message broker (such as ActiveMQ or Amazon MQ). It also reads and writes documents to the shared file store.
- **ActiveMQ**: This is the message broker (either a self-managed ActiveMQ instance or Amazon MQ), where the repository and the Transform Router send image transform requests and responses. These JSON-based messages are then passed to the Transform Router.
- **Transform Router**: The Transform Router allows simple (single-step) and pipeline (multi-step) transforms that are passed to the Transform Engines. The Transform Router (and the Transform Engines) run as independently scalable Docker containers.
- **Transform Engines**: The Transform Engines transform files referenced by the repository and retrieved from the shared file store. Here are some example transformations for each Transform Engine (this is not an exhaustive list):
    - LibreOffice (e.g. docx to pdf)
    - ImageMagick (e.g. resize)
    - Alfresco PDF Renderer (e.g. pdf to png)
    - Tika (e.g. docx to plain text)
    - Misc. (not included in diagram)
- **Shared File Store**: This is used as temporary storage for the original source file (stored by the repository), intermediate files for multi-step transforms, and the final transformed target file. The target file is retrieved by the repository after it's been processed by one or more of the Transform Engines.

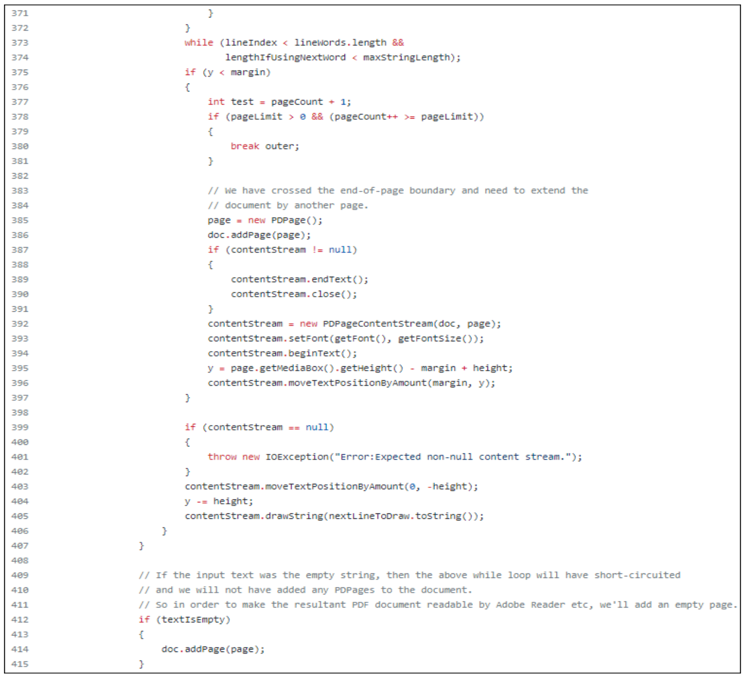
(*See* https://docs.alfresco.com/transform-service/latest/admin/)

115. In an example implementation, the Alfresco Transform Service implements a text to PDF content transformation wherein text is the input data in the first document format and PDF is the output data in the second document format.


```java
    public void transform(final String sourceMimetype, final String targetMimetype, final Map<String, String> parameters,
                          final File sourceFile, final File targetFile) throws Exception
    {
        String sourceEncoding = parameters.get(SOURCE_ENCODING);
        String stringPageLimit = parameters.get(PAGE_LIMIT);
        int pageLimit = -1;
        if (stringPageLimit != null)
        {
            pageLimit = parseInt(stringPageLimit, PAGE_LIMIT);
        }

        PDDocument pdf = null;
        try (InputStream is = new FileInputStream(sourceFile);
             Reader ir = new BufferedReader(buildReader(is, sourceEncoding));
             OutputStream os = new BufferedOutputStream(new FileOutputStream(targetFile)))
        {
            //TransformationOptionLimits limits = getLimits(reader, writer, options);
            //TransformationOptionPair pageLimits = limits.getPagesPair();
            pdf = transformer.createPDFFromText(ir, pageLimit);
            pdf.save(os);
        }
        finally
        {
            if (pdf != null)
            {
                try { pdf.close(); } catch (Throwable e) {e.printStackTrace(); }
            }
        }
    }
```



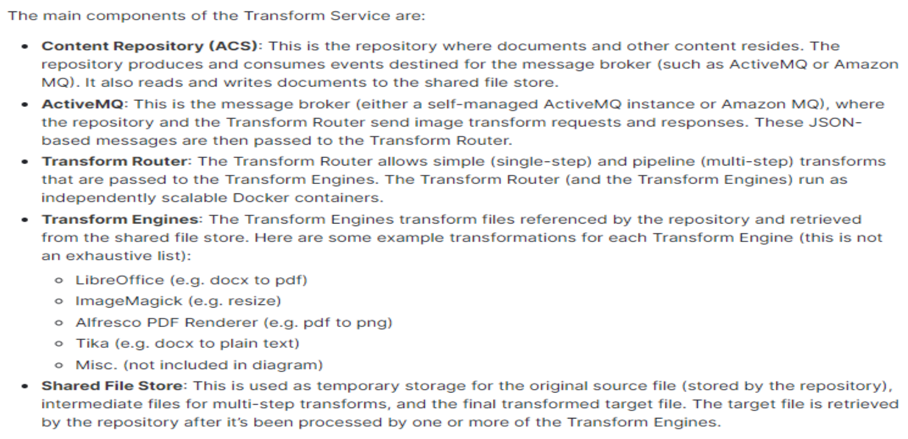
```
        // The following code is based on the code in TextToPDF with the addition of
        // checks for page limits.
        // The calling code must close the PDDocument once finished with it.
        public PDDocument createPDFFromText(Reader text, int pageLimit)
            throws IOException
        {
            PDDocument doc = null;
            int pageCount = 0;
            try
            {
                final int margin = 40;
                float height = getFont().getFontDescriptor().getFontBoundingBox().getHeight() / 1000;

                //calculate font height and increase by 5 percent.
                height = height * getFontSize() * 1.05f;
                doc = new PDDocument();
                BufferedReader data = (text instanceof BufferedReader) ? (BufferedReader) text : new BufferedReader(text);
                String nextLine;
                PDPage page = new PDPage();
                PDPageContentStream contentStream = null;
                float y = -1;
                float maxStringLength = page.getMediaBox().getWidth() - 2 * margin;

                // There is a special case of creating a PDF document from an empty string.
                boolean textIsEmpty = true;

                outer:
                while ((nextLine = data.readLine()) != null)
                {
                    // The input text is nonEmpty. New pages will be created and added
                    // to the PDF document as they are needed, depending on the length of
                    // the text.
                    textIsEmpty = false;

                    String[] lineWords = nextLine.trim().split(" ");
                    int lineIndex = 0;
                    while (lineIndex < lineWords.length)
                    {
                        final StringBuilder nextLineToDraw = new StringBuilder();
                        float lengthIfUsingNextWord = 0;
                        do
                        {
                            nextLineToDraw.append(lineWords[lineIndex]);
                            nextLineToDraw.append(" ");
                            lineIndex++;
                            if (lineIndex < lineWords.length)
                            {
                                String lineWithNextWord = nextLineToDraw.toString() + lineWords[lineIndex];
                                lengthIfUsingNextWord =
                                    (getFont().getStringWidth(
                                        lineWithNextWord) / 1000) * getFontSize();
```



```
                            }
                        }
                        while (lineIndex < lineWords.length &&
                               lengthIfUsingNextWord < maxStringLength);
                        if (y < margin)
                        {
                            int test = pageCount + 1;
                            if (pageLimit > 0 && (pageCount++ >= pageLimit))
                            {
                                break outer;
                            }

                            // We have crossed the end-of-page boundary and need to extend the
                            // document by another page.
                            page = new PDPage();
                            doc.addPage(page);
                            if (contentStream != null)
                            {
                                contentStream.endText();
                                contentStream.close();
                            }
                            contentStream = new PDPageContentStream(doc, page);
                            contentStream.setFont(getFont(), getFontSize());
                            contentStream.beginText();
                            y = page.getMediaBox().getHeight() - margin + height;
                            contentStream.moveTextPositionByAmount(margin, y);
                        }

                        if (contentStream == null)
                        {
                            throw new IOException("Error:Expected non-null content stream.");
                        }
                        contentStream.moveTextPositionByAmount(0, -height);
                        y -= height;
                        contentStream.drawString(nextLineToDraw.toString());
                    }
                }

                // If the input text was the empty string, then the above while loop will have short-circuited
                // and we will not have added any PDPages to the document.
                // So in order to make the resultant PDF document readable by Adobe Reader etc, we'll add an empty page.
                if (textIsEmpty)
                {
                    doc.addPage(page);
                }
```

(*See* https://github.com/Alfresco/alfresco-transform-core/blob/e575ec943a5fa5dddca5593e6795a17a2bbb3cb6/alfresco-transform-misc/alfresco-transform-misc/src/main/java/org/alfresco/transformer/transformers/TextToPdfContentTransformer.java)

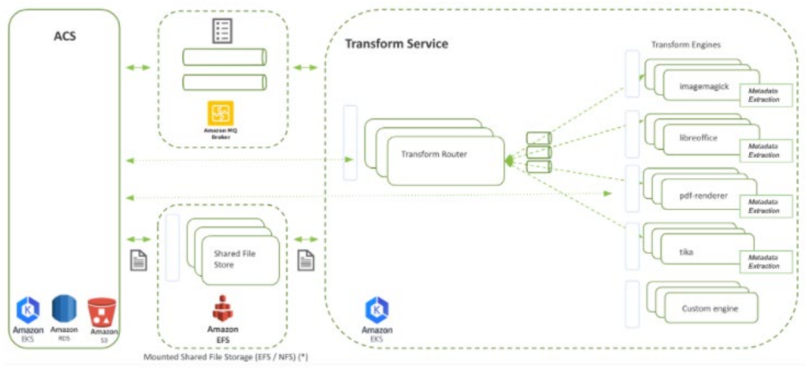
116. The Accused Products perform a method that includes *creating an output pipeline for the process* and *executing the process to produce a physical output object through the output pipeline*. For example, the Alfresco Transform Service includes the content repository (ACS) where documents and other content resides, and shared file store "used as temporary storage for the original source file (stored by the repository), intermediate files for multi-step transforms, and the final transformed target file." The target file is generated and sent to the ACS via an output port.


The main components of the Transform Service are:

- **Content Repository (ACS)**: This is the repository where documents and other content resides. The repository produces and consumes events destined for the message broker (such as ActiveMQ or Amazon MQ). It also reads and writes documents to the shared file store.
- **ActiveMQ**: This is the message broker (either a self-managed ActiveMQ instance or Amazon MQ), where the repository and the Transform Router send image transform requests and responses. These JSON-based messages are then passed to the Transform Router.
- **Transform Router**: The Transform Router allows simple (single-step) and pipeline (multi-step) transforms that are passed to the Transform Engines. The Transform Router (and the Transform Engines) run as independently scalable Docker containers.
- **Transform Engines**: The Transform Engines transform files referenced by the repository and retrieved from the shared file store. Here are some example transformations for each Transform Engine (this is not an exhaustive list):
  - LibreOffice (e.g. docx to pdf)
  - ImageMagick (e.g. resize)
  - Alfresco PDF Renderer (e.g. pdf to png)
  - Tika (e.g. docx to plain text)
  - Misc. (not included in diagram)
- **Shared File Store**: This is used as temporary storage for the original source file (stored by the repository), intermediate files for multi-step transforms, and the final transformed target file. The target file is retrieved by the repository after it's been processed by one or more of the Transform Engines.

(*See* https://docs.alfresco.com/transform-service/1.4/admin/)

The following diagram shows a simple representation of the Transform Service components:



Note that from Transform Service version 1.3.2 the metadata extraction that usually takes part in the core repository legacy transform engines has now been lifted out into the separate transform engine processes. This enables scaling of the metadata extraction.

(*See* https://docs.alfresco.com/transform-service/1.4/admin/)

117. Each claim in the ’830 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the ’830 Patent.

118. On information and belief, there has been significant effort by Hyland to imitate OpenText’s patent-protected products to compete with OpenText in the ECM and EIM markets and to increase Hyland’s share of that market at the expense of OpenText’s market share. (*See, e.g.*, Exhibit B (2020.09.09 - Hyland enters definitive agreement to acquire Alfresco, hyland.com), Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com).) Hyland’s efforts have resulted in the Accused Products, which infringe at least claim 1 of the ’830 patent as described above, and those efforts would have exposed Hyland to the ’830 patent prior to the filing of the original Complaint in this action.

119. Defendant has known of the ’830 Patent since receiving a letter identifying the patent and the infringement on September 2, 2022. At the very least, Hyland has been aware of the ’830 patent and of its infringement based on the Accused Products

since at least the filing and/or service of this Complaint. Further, OpenText marks its products with the ’830 patent.

120. On information and belief, at least as of the filing of the Complaint in this action, Hyland has knowingly and actively induced and is knowingly and actively inducing at least its customers and partners to directly infringe at least claim 1 of the ’830 patent, and has done so with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b), by activities relating to selling, marketing, advertising, promoting, supporting, installing, and distributing its Accused Products in the United States. (Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com).) On information and belief, those activities continue.

121. On information and belief, Hyland deliberately and knowingly encourages, instructs, directs, and/or requires third parties—including its partners, customers, and/or end users—to use the Accused Products in a way that infringes at least claim 1 of the ’830 patent as described above.

122. Hyland’s partners, customers, and end users of its Accused Products directly infringe at least claim 1 of the ’830 patent, at least by using the Accused Products, as described above.

123. For example, on information and belief, Hyland knowingly and intentionally shares instructions, guides, and manuals, including through its website, training programs, and/or YouTube, which advertise and instruct third parties on how to use the Accused Products in a way that directly infringes at least claim 1 of the ’830 patent as described above, including at least Hyland’s customers. On further information and belief, Hyland knowingly and intentionally provides customer service or technical support to purchasers of the infringing Accused Products, which directs and encourages Hyland’s customers to use the Accused Products in a way that directly infringes at least claim 1 of the ’830 patent as described above.

124. On information and belief, the infringing actions of each customer and/or end-user of the Accused Products are attributable to Hyland.

125. On information and belief, Hyland sells and offers for sale the Accused Products and provides technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. On information and belief, each customer enters into a contractual relationship with Hyland, which obligates each customer to perform certain actions as a condition to use the Accused Products. Further, in order to receive the benefit of Hyland's continued technical support and its specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that infringes the '830 patent. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Products remain operational for each customer through ongoing technical support, on information and belief, Hyland establishes the manner and timing of each customer's performance of activities that infringe the '830 patent.

126. On information and belief, Hyland forms a joint enterprise with its customers to engage in directly infringing the '830 patent. On further information and belief, Hyland together with each customer operate under a contractual agreement; have a common purpose to operate the Accused Products in a way that directly infringes the '830 patent as outlined in the paragraphs above; have pecuniary interests in operating the Accused Products by directly profiting from the sale and/or maintenance of the Accused Products or by indirectly profiting from the increased efficiency resulting from use of the Accused Products; and have equal rights to a voice in the direction of the enterprise either by guiding and advising on the operation and capabilities of the Accused Products with product-specific know-how and expertise or by requesting that certain customer-specific capabilities be implemented in the Accused Products.

127. Hyland also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the inventions claimed, and not a staple article or commodity of commerce suitable for substantial non-infringing uses.

128. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including the transformation of files from their current format into other formats and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’830 patent, that functionality could not be performed.

129. Additionally, the accused functionality, including the transformation of files from their current format into other formats and related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For example, without the Alfresco Transformation Services, the Accused Products could not convert files from one format to another, including metadata. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’830 Patent, that functionality could not be performed.

130. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Hyland constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionalities (including the file transformation functionality) constitute a material part of the inventions claimed because such functionality is integral to the processes identified above (such as to "creating a plurality of input threads," "producing filtered data," identifying "each event in the filtered data," and creating "a process for transforming the messages according to the generic data structure into output data") as recited in the claims of the '830 Patent. None of these products are staple goods—they are sophisticated and customized ECM products, methods, and systems.

131. OpenText "consists of four revenue streams: license, cloud services and subscriptions, customer support, and professional service and other." (Exhibit A at 9-10 (Aug. 6, 2020 10-K).) Each revenue stream relates directly to the ability of OpenText to acquire and retain customers for its software products in a market that is "highly competitive" and increasingly more competitive "as a result of ongoing software industry consolidation," such as Hyland's acquisition of Alfresco. (Exhibit A at 11 (Aug. 6, 2020 10-K); *see also* Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com); Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com); Exhibit F at 4 ("The Forrester Wave: ECM Content Platforms, Q3 2019"); Exhibit E at 3 (2020.11.16 - Gartner Content Services Report 2020).) OpenText is an innovator in the market and has acquired multiple patents, including the Patents-in-Suit, to give it an advantage over such competition. Hyland's infringing activities have resulted and will continue to result in irreparable harm to OpenText because of the competitive threat that Hyland—including Hyland's acquisition of Alfresco—has to OpenText's share of the relevant "highly competitive" market, and the impact that Hyland's infringing activities have on each one of OpenText's four revenue streams. Further, public interest factors favor

OpenText as the owner and assignee of government-issued patents, including the Patents-in-Suit, that serve to recognize OpenText's innovative contribution to the public knowledge in exchange for the patent protection that Hyland is now infringing.

132. For past infringement, OpenText has suffered damages, including lost profits, as a result of Hyland's infringement of the '830 patent. Hyland is therefore liable to OpenText under 35 U.S.C. § 284 for past damages in an amount that adequately compensates OpenText for Hyland's infringement, but no less than a reasonable royalty.

133. OpenText is entitled to a preliminary injunction to maintain the status quo between OpenText and Hyland, which, through its acquisition of Alfresco, is now one of OpenText's biggest competitors (*see, e.g.*, Exhibits 24, Exhibit B (2020.09.09 - Hyland enters definitive agreement to acquire Alfresco, hyland.com), Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com)), and is using OpenText's patented technology to compete with OpenText in the ECM and EIM markets.

134. For ongoing and future infringement, OpenText will continue to suffer irreparable harm, including without limitation, loss of market share, customers and/or convoyed sales and services which cannot be accurately quantified nor adequately compensated for by money damages, unless this Court preliminarily and permanently enjoins Hyland, its agents, employees, representatives, and all others acting in concert with Hyland from infringing the '830 patent.

135. In the alternative, OpenText is entitled to damages in lieu of an injunction, in an amount consistent with the facts, for future infringement. Hyland's continued infringement, at least since it had notice of the '830 patent, is knowing and willful. Hyland will be an adjudicated infringer of a valid patent and, thus, Hyland's future infringement will be willful as a matter of law.

136. Hyland's infringement is without license or other authorization.

137. This case is exceptional, entitling Plaintiffs to enhanced damages under 35 U.S.C. § 284 and an award of attorneys' fees and costs incurred in prosecuting this action under 35 U.S.C. § 285.

THIRD CAUSE OF ACTION

(INFRINGEMENT OF THE '381 PATENT)

138. OpenText realleges and incorporates by reference the allegations of the preceding paragraphs of this First Amended Complaint.

139. Hyland has infringed and continues to infringe one or more claims of the '381 patent in violation of 35 U.S.C. § 271 in this judicial district and elsewhere in the United States and will continue to do so unless enjoined by this Court. Hyland directly infringes at least claim 8 of the '381 patent by making, using, selling, and/or offering to sell at least the Alfresco Enterprise Content Management System (ECM), including the Alfresco Transform Service, at least when used for their ordinary and customary purposes.

140. For example, claim 8 of the '381 patent recites:

> 8. A method for streamed transformation of data, the method comprising:
>
> a first computing device receiving a request from a second computing device to read or write a file, the first computing device comprising a memory, a processor, at least one non-transitory computer-readable medium, and stored instructions translatable by the processor;
>
> responsive to the request, the first computing device creating a transformation pipeline, the transformation pipeline providing a processing sequence and traffic path for streamed data and comprising a sequence of transformation streams corresponding to a set of transformations to be applied to the file, the creating the transformation pipeline comprising instantiating a stream object for each transformation stream of the transformation streams, the stream object including a write method for moving a unit of data into the each transformation stream and a read method for retrieving the unit of data from the each transformation

stream, calling a transformation function of the set of transformations to transform the unit of data, and providing the unit of data thus transformed within the each transformation stream to a next transformation stream or, if there are no more transformation streams in the transformation pipeline, to a destination device;

the first computing device receiving file data for the file streamed from a source device;

buffering the file data from the source device in a memory buffer at the first computing device;

the first computing device segmenting buffered file data to produce units of file data;

the first computing device passing and transforming the file data through the transformation pipeline one unit of file data at a time, wherein transforming a unit of file data comprises:

providing the unit of file data to a first transformation stream in the transformation pipeline, the providing performed by a write method associated with the first transformation stream;

applying a transformation associated with the first transformation stream to the unit of file data, the applying performed by a read method associated with the first transformation stream, the applying including the read method calling a transformation function to perform the transformation; and

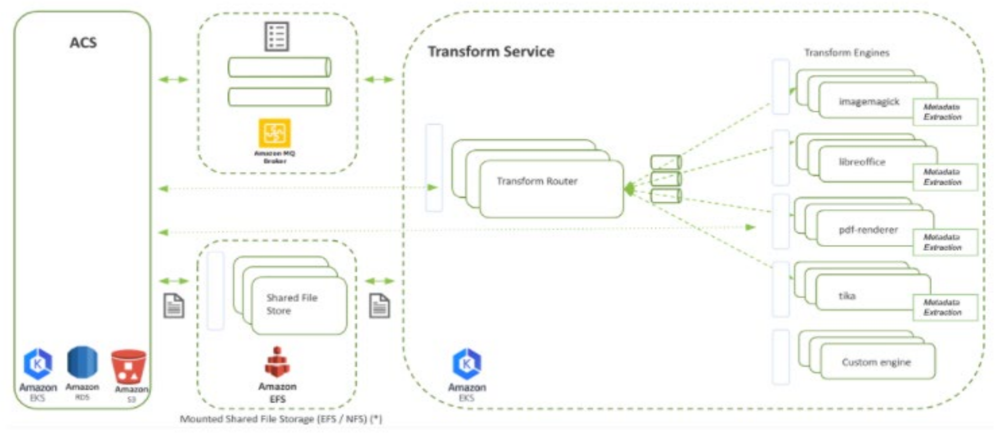
performing the providing and the applying until all transformations in the set of transformations have been applied to the unit of file data through the sequence of transformation streams in the transformation pipeline; and

the first computing device sending the unit of file data so transformed by the set of transformations to a destination device.

141. The Accused Products perform the method of claim 8 of the ’381 Patent. To the extent the preamble is construed to be limiting, the Accused Products perform *a*

method for streamed transformation of data, as further explained below. For example, the Alfresco Transform Service converts files "from their current format into other formats" in a network environment by pipeline transformer having multiple T-engines.

The following diagram shows a simple representation of the Transform Service components:



Note that from Transform Service version 1.3.2 the metadata extraction that usually takes part in the core repository legacy transform engines has now been lifted out into the separate transform engine processes. This enables scaling of the metadata extraction.

(*See* https://docs.alfresco.com/transform-service/1.4/.)

(*See* https://docs.alfresco.com/transform-service/1.4/admin/)

- **Pipeline transformer**: This maps T-Requests to a sequence of intermediate T-Request steps, which are handled by multiple `T-Engines`. These transformers handle situations where there is no single engine that can directly transform one media type to another, but that can be achieved through intermediate media types and transformations.

  For example: `application/msword` to `image/png` can't be directly performed by one single engine, but it can be handled by `LIBREOFFICE` (which would generate `application/pdf`) and then `PDF_RENDERER`.

(*See* https://docs.alfresco.com/transform-service/1.4/config/transformers/)

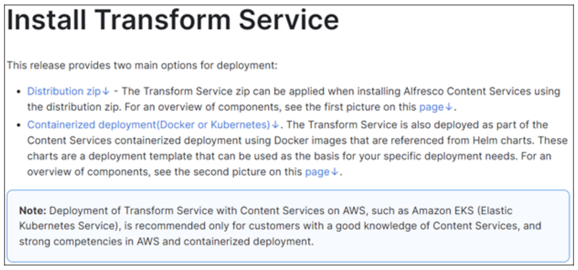
Configure a custom transform pipeline

Local Transforms may be combined together in a pipeline to form a new transform, where the output from one becomes the input to the next and so on. A pipeline definition (JSON) defines the sequence of transforms and intermediate Media Types. Like any other transformer, it specifies a list of supported source and target Media Types. If you don't supply any, all possible combinations are assumed to be available. The definition may reuse the transformOptions of transformers in the pipeline, but typically will define its own subset of these.

images. Note that the all-in-one T-Engine is the default option for the Docker Compose deployment and installation using the distribution zip, however Helm deployments continue to use the five separate T-Engines in order to provide balanced throughput and scalability improvements. This release also provides two main options for deployment: using containerized deployment or using the distribution zip.

(*See* https://github.com/Alfresco/acs-packaging/blob/release/6.2.N/docs/custom-transforms-and-renditions.md#configure-a-custom-transform-pipeline)

142. The Accused Products perform a method that includes *a first computing device receiving a request from a second computing device to read or write a file, the first computing device comprising a memory, a processor, at least one non-transitory computer-readable medium, and stored instructions translatable by the processor*. For example, Alfresco Transform Service may be deployed in a number of ways, including installed on a computer by using the distribution zip file or "deployed as part of the Content Services containerized deployment using Docker images" in a Kubernetes cluster, for example, on Amazon Web Services (AWS). In either case, the system includes servers and/or other computers that receive requests from devices such as other servers or end user devices. In each case, the servers and or end user devices each include memory, processors, and non-transitory computer-readable medium, as well instructions that are executed by one or more processors.

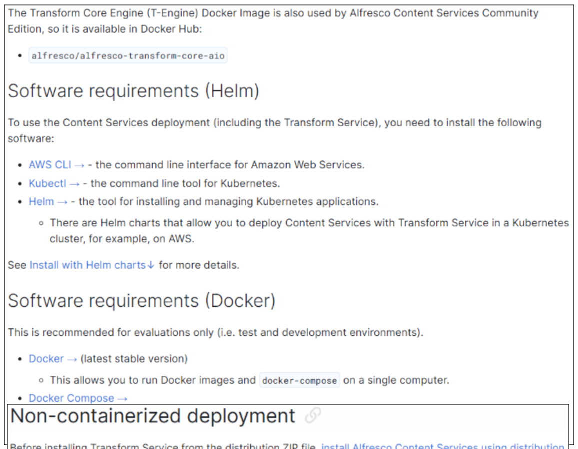


Install Transform Service

This release provides two main options for deployment:

- Distribution zip↓ - The Transform Service zip can be applied when installing Alfresco Content Services using the distribution zip. For an overview of components, see the first picture on this page↓.
- Containerized deployment(Docker or Kubernetes)↓. The Transform Service is also deployed as part of the Content Services containerized deployment using Docker images that are referenced from Helm charts. These charts are a deployment template that can be used as the basis for your specific deployment needs. For an overview of components, see the second picture on this page↓.

Note: Deployment of Transform Service with Content Services on AWS, such as Amazon EKS (Elastic Kubernetes Service), is recommended only for customers with a good knowledge of Content Services, and strong competencies in AWS and containerized deployment.



The Transform Core Engine (T-Engine) Docker Image is also used by Alfresco Content Services Community Edition, so it is available in Docker Hub:

- `alfresco/alfresco-transform-core-aio`

Software requirements (Helm)

To use the Content Services deployment (including the Transform Service), you need to install the following software:

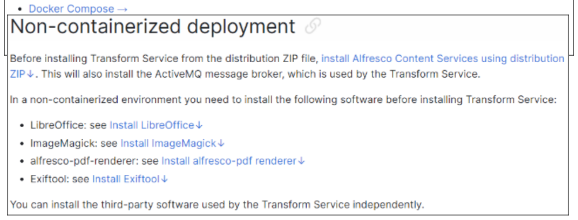
- AWS CLI → - the command line interface for Amazon Web Services.
- Kubectl → - the command line tool for Kubernetes.
- Helm → - the tool for installing and managing Kubernetes applications.
  - There are Helm charts that allow you to deploy Content Services with Transform Service in a Kubernetes cluster, for example, on AWS.

See Install with Helm charts↓ for more details.

Software requirements (Docker)

This is recommended for evaluations only (i.e. test and development environments).

- Docker → (latest stable version)
  - This allows you to run Docker images and `docker-compose` on a single computer.
- Docker Compose →



Non-containerized deployment

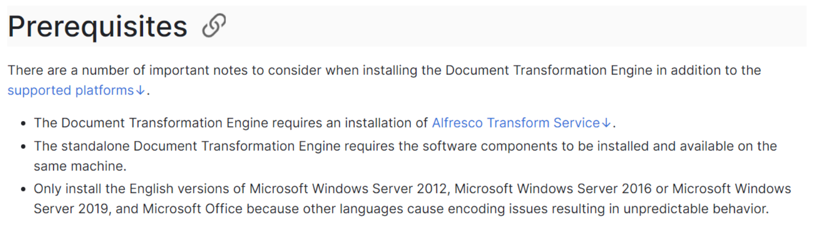
Before installing Transform Service from the distribution ZIP file, install Alfresco Content Services using distribution ZIP↓. This will also install the ActiveMQ message broker, which is used by the Transform Service.

In a non-containerized environment you need to install the following software before installing Transform Service:

- LibreOffice: see Install LibreOffice↓
- ImageMagick: see Install ImageMagick↓
- alfresco-pdf-renderer: see Install alfresco-pdf renderer↓
- Exiftool: see Install Exiftool↓

You can install the third-party software used by the Transform Service independently.

(*See* https://docs.alfresco.com/transform-service/1.4/install/)



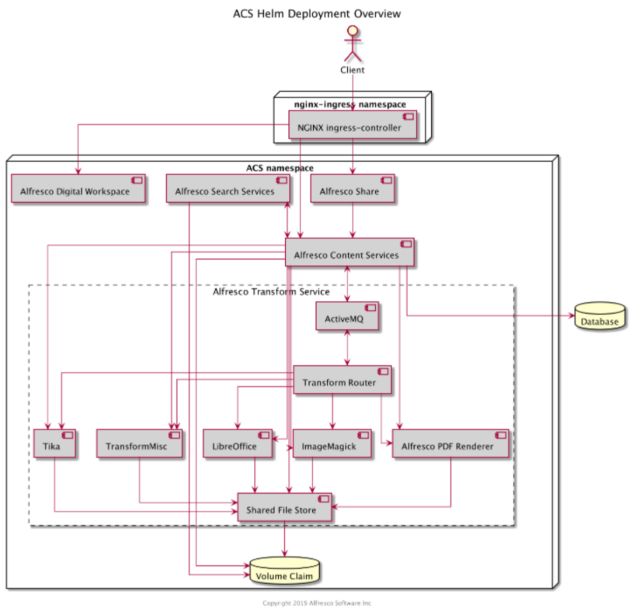
Prerequisites

There are a number of important notes to consider when installing the Document Transformation Engine in addition to the supported platforms↓.

- The Document Transformation Engine requires an installation of Alfresco Transform Service↓.
- The standalone Document Transformation Engine requires the software components to be installed and available on the same machine.
- Only install the English versions of Microsoft Windows Server 2012, Microsoft Windows Server 2016 or Microsoft Windows Server 2019, and Microsoft Office because other languages cause encoding issues resulting in unpredictable behavior.

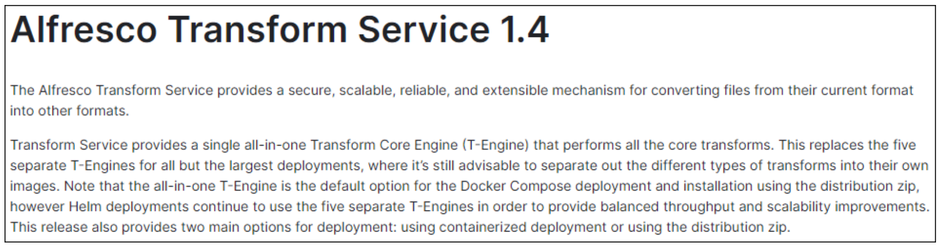
(*See* https://docs.alfresco.com/transform-service/1.4/install/)

143. As another example, the Alfresco Transform Service includes a "message broker (either a self-managed ActiveMQ instance or Amazon MQ)" that receives requests from a client to read and write a file.



(*See* https://docs.alfresco.com/transform-service/1.4/install/)

144. The Accused Products perform a method that includes *responsive to the request, the first computing device creating a transformation pipeline, the transformation pipeline providing a processing sequence and traffic path for streamed data and comprising a sequence of transformation streams corresponding to a set of*

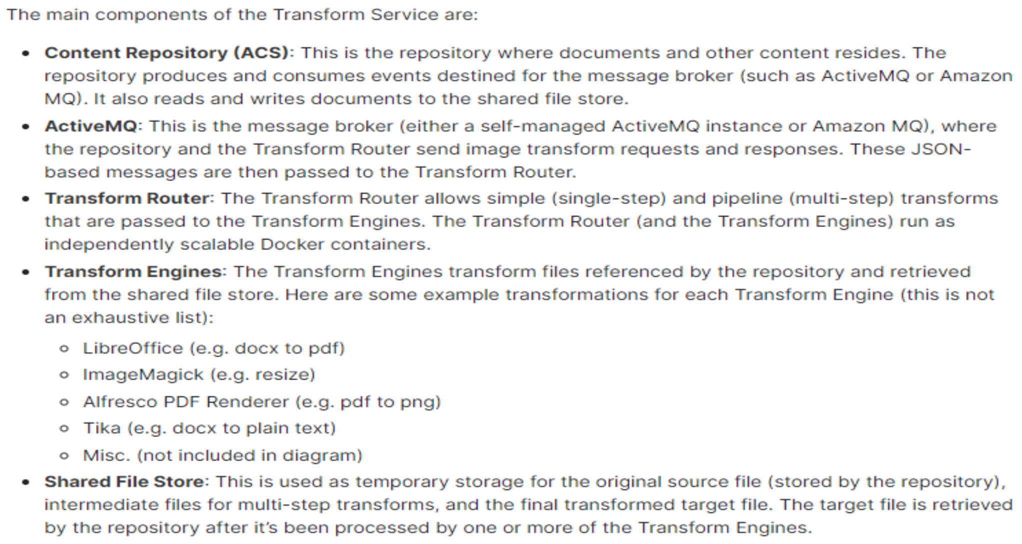


Alfresco Transform Service 1.4

The Alfresco Transform Service provides a secure, scalable, reliable, and extensible mechanism for converting files from their current format into other formats.

Transform Service provides a single all-in-one Transform Core Engine (T-Engine) that performs all the core transforms. This replaces the five separate T-Engines for all but the largest deployments, where it's still advisable to separate out the different types of transforms into their own images. Note that the all-in-one T-Engine is the default option for the Docker Compose deployment and installation using the distribution zip, however Helm deployments continue to use the five separate T-Engines in order to provide balanced throughput and scalability improvements. This release also provides two main options for deployment: using containerized deployment or using the distribution zip.

transformations to be applied to the file, the creating the transformation pipeline comprising instantiating a stream object for each transformation stream of the transformation streams, the stream object including a write method for moving a unit of data into the each transformation stream and a read method for retrieving the unit of data from the each transformation stream, calling a transformation function of the set of transformations to transform the unit of data, and providing the unit of data thus transformed within the each transformation stream to a next transformation stream or, if there are no more transformation streams in the transformation pipeline, to a destination device. For example, the Accused Products create a pipeline transformer in response to T-Request/Transform Request from the "message broker (either a self-managed ActiveMQ instance or Amazon MQ)." The pipeline transformer "maps T-Requests to a sequence of intermediate T-Request steps, which are handled by multiple T-Engines" for "converting files from their current format into other formats." Each T-Engine performs its own single-step transformation by reading and retrieving source and intermediate files for multi-step transforms from temporary storage (shared file store) and writing the transformed intermediate files and final transformed target file into the temporary storage (shared file store.) (*See* https://docs.alfresco.com/transform-service/1.4/)

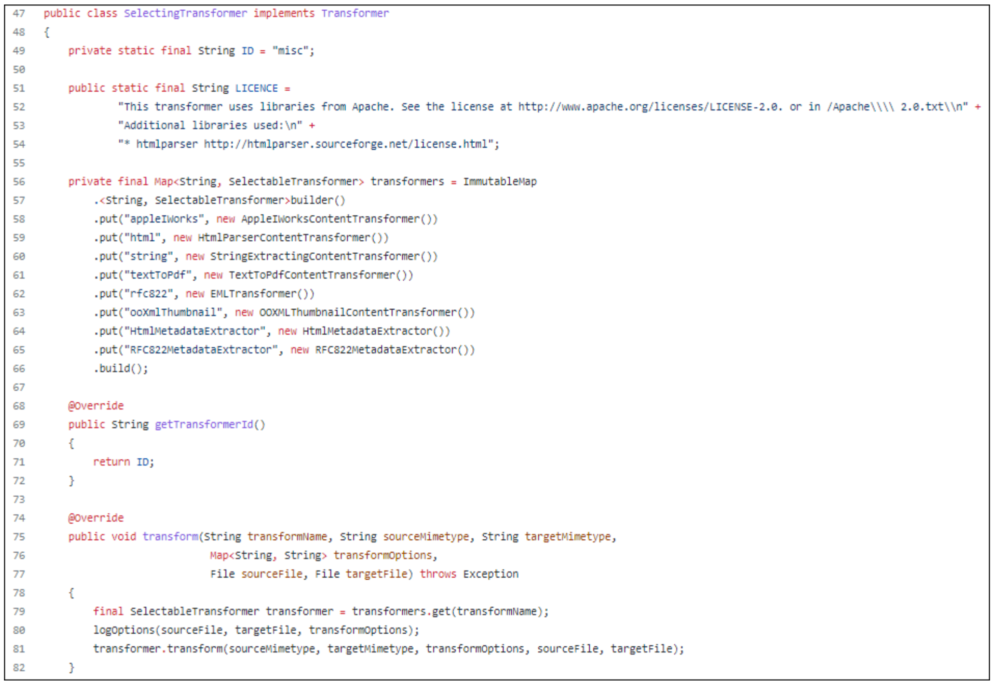


The main components of the Transform Service are:

- **Content Repository (ACS)**: This is the repository where documents and other content resides. The repository produces and consumes events destined for the message broker (such as ActiveMQ or Amazon MQ). It also reads and writes documents to the shared file store.
- **ActiveMQ**: This is the message broker (either a self-managed ActiveMQ instance or Amazon MQ), where the repository and the Transform Router send image transform requests and responses. These JSON-based messages are then passed to the Transform Router.
- **Transform Router**: The Transform Router allows simple (single-step) and pipeline (multi-step) transforms that are passed to the Transform Engines. The Transform Router (and the Transform Engines) run as independently scalable Docker containers.
- **Transform Engines**: The Transform Engines transform files referenced by the repository and retrieved from the shared file store. Here are some example transformations for each Transform Engine (this is not an exhaustive list):
  - LibreOffice (e.g. docx to pdf)
  - ImageMagick (e.g. resize)
  - Alfresco PDF Renderer (e.g. pdf to png)
  - Tika (e.g. docx to plain text)
  - Misc. (not included in diagram)
- **Shared File Store**: This is used as temporary storage for the original source file (stored by the repository), intermediate files for multi-step transforms, and the final transformed target file. The target file is retrieved by the repository after it's been processed by one or more of the Transform Engines.

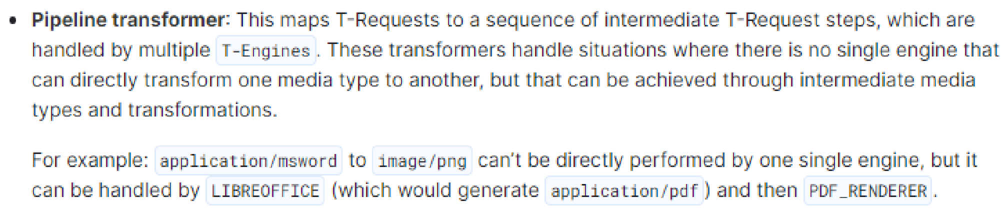
(https://docs.alfresco.com/transform-service/1.4/admin/.)

145. As an example, the Alfresco Transform Service implements multiple transformers and uses "SelectingTransformer" class to select the proper transformer to perform the content transformation. "These transformers handle situations where there is no single engine that can directly transform one media type to another, but that can be achieved through intermediate media types and transformations." For example, "application/msword to image/png can't be directly performed by one single engine, but it can be handled by LIBREOFFICE (which would generate application/pdf) and then PDF_RENDERER." In this example, the process reads the word docx file from the temporary storage (shared file store); calls a T-Engine LibreOffice to convert it into a PDF file; and passes the PDF file to the next transformer by writing it into the temporary storage (shared file store), where it then reads the intermediate PDF file from the temporary storage (shared file store); calls a T-Engine Alfresco PDF Renderer to transform the PDF file to a PNG image file; and writes the PNG image file into the temporary storage (shared file store). The final transformed PNG image file is then provided to the appropriate device(s).



```
public class SelectingTransformer implements Transformer
{
    private static final String ID = "misc";

    public static final String LICENCE =
            "This transformer uses libraries from Apache. See the license at http://www.apache.org/licenses/LICENSE-2.0. or in /Apache\\\\ 2.0.txt\\n" +
            "Additional libraries used:\n" +
            "* htmlparser http://htmlparser.sourceforge.net/license.html";

    private final Map<String, SelectableTransformer> transformers = ImmutableMap
        .<String, SelectableTransformer>builder()
        .put("appleIWorks", new AppleIWorksContentTransformer())
        .put("html", new HtmlParserContentTransformer())
        .put("string", new StringExtractingContentTransformer())
        .put("textToPdf", new TextToPdfContentTransformer())
        .put("rfc822", new EMLTransformer())
        .put("ooXmlThumbnail", new OOXMLThumbnailContentTransformer())
        .put("HtmlMetadataExtractor", new HtmlMetadataExtractor())
        .put("RFC822MetadataExtractor", new RFC822MetadataExtractor())
        .build();

    @Override
    public String getTransformerId()
    {
        return ID;
    }

    @Override
    public void transform(String transformName, String sourceMimetype, String targetMimetype,
                          Map<String, String> transformOptions,
                          File sourceFile, File targetFile) throws Exception
    {
        final SelectableTransformer transformer = transformers.get(transformName);
        logOptions(sourceFile, targetFile, transformOptions);
        transformer.transform(sourceMimetype, targetMimetype, transformOptions, sourceFile, targetFile);
    }
```

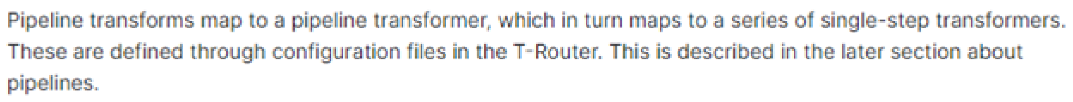
(*See* https://github.com/Alfresco/alfresco-transform-core/blob/master/alfresco-transform-misc/alfresco-transform-misc/src/main/java/org/alfresco/transformer/transformers/SelectingTransformer.java.)



- **Pipeline transformer:** This maps T-Requests to a sequence of intermediate T-Request steps, which are handled by multiple `T-Engines`. These transformers handle situations where there is no single engine that can directly transform one media type to another, but that can be achieved through intermediate media types and transformations.

  For example: `application/msword` to `image/png` can't be directly performed by one single engine, but it can be handled by `LIBREOFFICE` (which would generate `application/pdf`) and then `PDF_RENDERER`.

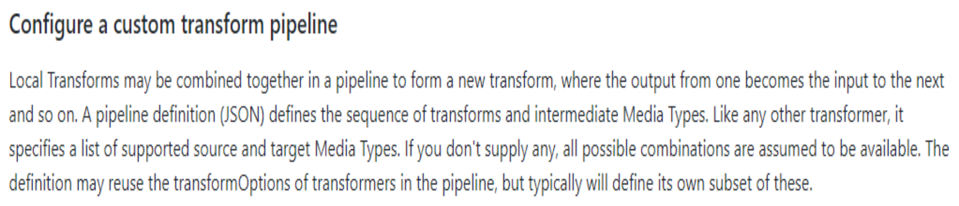
(*See* https://docs.alfresco.com/transform-service/1.4/config/transformers/)



Pipeline transforms map to a pipeline transformer, which in turn maps to a series of single-step transformers. These are defined through configuration files in the T-Router. This is described in the later section about pipelines.

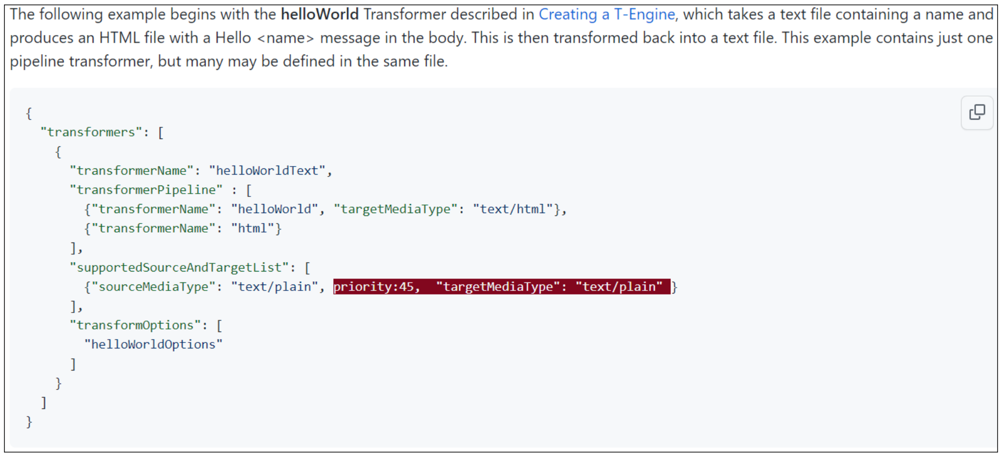
(*See* https://docs.alfresco.com/transform-service/1.4/config/transformers/)

146. In another example, the Accused Products provide "Local Transforms [that] may be combined together in a pipeline to form a new transform." The custom transform pipeline operates such that "the output from one [transform] becomes the input to the next [transform] and so on."



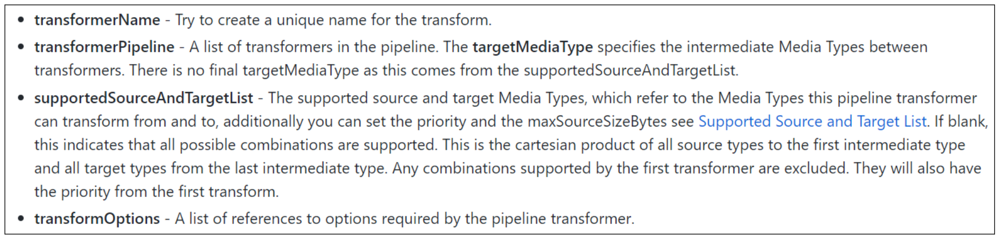
Configure a custom transform pipeline

Local Transforms may be combined together in a pipeline to form a new transform, where the output from one becomes the input to the next and so on. A pipeline definition (JSON) defines the sequence of transforms and intermediate Media Types. Like any other transformer, it specifies a list of supported source and target Media Types. If you don't supply any, all possible combinations are assumed to be available. The definition may reuse the transformOptions of transformers in the pipeline, but typically will define its own subset of these.



The following example begins with the **helloWorld** Transformer described in Creating a T-Engine, which takes a text file containing a name and produces an HTML file with a Hello <name> message in the body. This is then transformed back into a text file. This example contains just one pipeline transformer, but many may be defined in the same file.

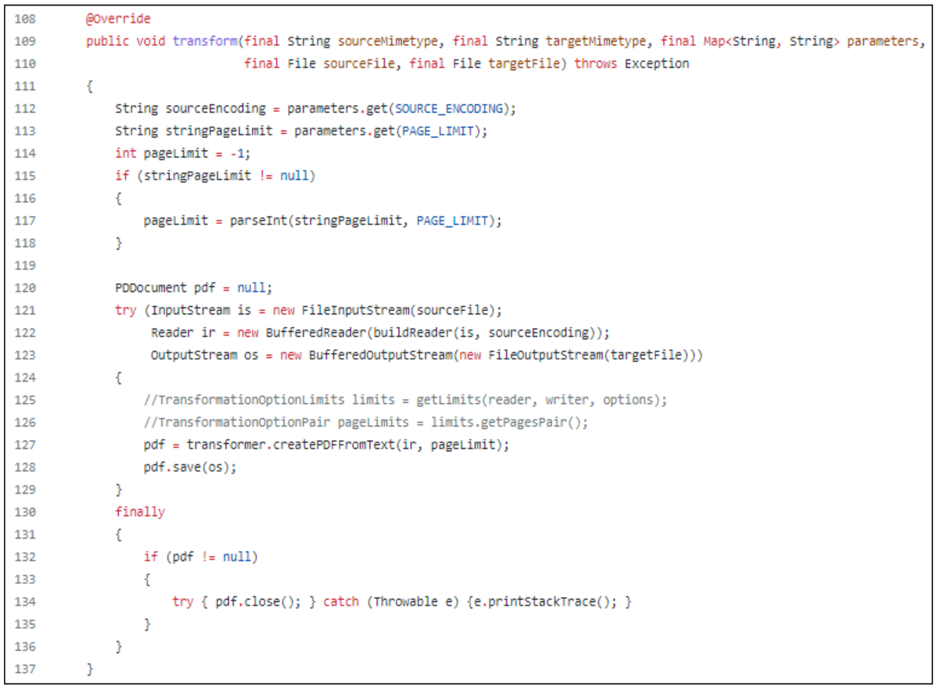
```
{
  "transformers": [
    {
      "transformerName": "helloWorldText",
      "transformerPipeline" : [
        {"transformerName": "helloWorld", "targetMediaType": "text/html"},
        {"transformerName": "html"}
      ],
      "supportedSourceAndTargetList": [
        {"sourceMediaType": "text/plain", priority:45,  "targetMediaType": "text/plain" }
      ],
      "transformOptions": [
        "helloWorldOptions"
      ]
    }
  ]
}
```



- **transformerName** - Try to create a unique name for the transform.
- **transformerPipeline** - A list of transformers in the pipeline. The **targetMediaType** specifies the intermediate Media Types between transformers. There is no final targetMediaType as this comes from the supportedSourceAndTargetList.
- **supportedSourceAndTargetList** - The supported source and target Media Types, which refer to the Media Types this pipeline transformer can transform from and to, additionally you can set the priority and the maxSourceSizeBytes see Supported Source and Target List. If blank, this indicates that all possible combinations are supported. This is the cartesian product of all source types to the first intermediate type and all target types from the last intermediate type. Any combinations supported by the first transformer are excluded. They will also have the priority from the first transform.
- **transformOptions** - A list of references to options required by the pipeline transformer.

(*See* https://github.com/Alfresco/acs-packaging/blob/release/6.2.N/docs/custom-transforms-and-renditions.md#configure-a-custom-transform-pipeline)

147. The Accused Products perform a method that includes *the first computing device receiving file data for the file streamed from a source device* and *buffering the file data from the source device in a memory buffer at the first computing device*. For example, as explained above, the Alfresco Transform Service receives an original source file from the content repository (ACS) and temporarily stores it in the shared file store. (*See* https://docs.alfresco.com/transform-service/1.4/admin/). In addition, the Alfresco Transform Service's source code uses "InputStream" to receive file data ("sourceFile"). The Alfresco Transform Service also uses "BufferedReader" to read the source file.

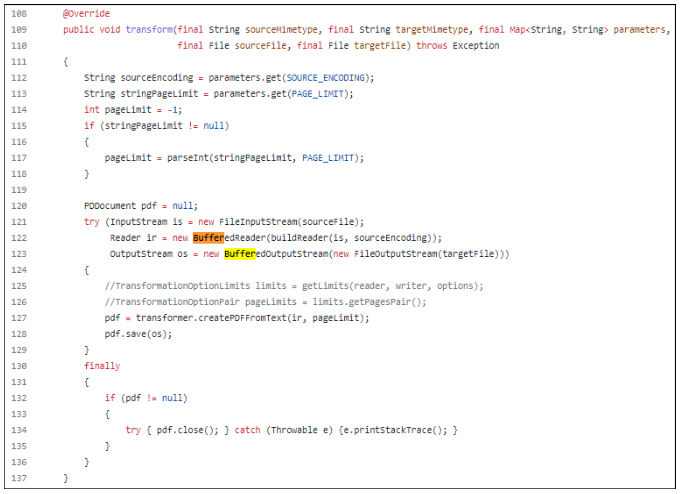


```
@Override
public void transform(final String sourceMimetype, final String targetMimetype, final Map<String, String> parameters,
                      final File sourceFile, final File targetFile) throws Exception
{
    String sourceEncoding = parameters.get(SOURCE_ENCODING);
    String stringPageLimit = parameters.get(PAGE_LIMIT);
    int pageLimit = -1;
    if (stringPageLimit != null)
    {
        pageLimit = parseInt(stringPageLimit, PAGE_LIMIT);
    }

    PDDocument pdf = null;
    try (InputStream is = new FileInputStream(sourceFile);
         Reader ir = new BufferedReader(buildReader(is, sourceEncoding));
         OutputStream os = new BufferedOutputStream(new FileOutputStream(targetFile)))
    {
        //TransformationOptionLimits limits = getLimits(reader, writer, options);
        //TransformationOptionPair pageLimits = limits.getPagesPair();
        pdf = transformer.createPDFFromText(ir, pageLimit);
        pdf.save(os);
    }
    finally
    {
        if (pdf != null)
        {
            try { pdf.close(); } catch (Throwable e) {e.printStackTrace(); }
        }
    }
}
```

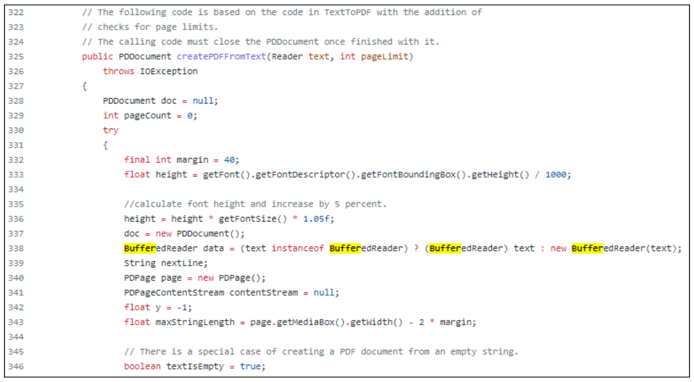
(*See* https://github.com/Alfresco/alfresco-transform-core/blob/f24969199c13140f32ba976db65d6c15425fb3b0/alfresco-transform-misc/alfresco-transform-misc/src/main/java/org/alfresco/transformer/transformers/TextToPdfContentTransformer.java)

148. On information and belief, the Accused Products perform a method that includes *the first computing device segmenting buffered file data to produce units of file data* and *the first computing device passing and transforming the file data through the transformation pipeline one unit of file data at a time*. As explained above, the Alfresco Transform Service transforms file to other formats by passing it through pipeline transformer which consists of multiple Transform Engines/T-Engines. The pipeline transformer "maps T-Requests to a sequence of intermediate T-Requests steps, which are handled by multiple T-Engines," such that file data is transformed from one format into another "through intermediate media types and transformations." For example, the Alfresco Transform Service uses "BufferedReader" to read an input data stream and store the input data in a buffered reader "ir." The method "createPDFFromText" is used

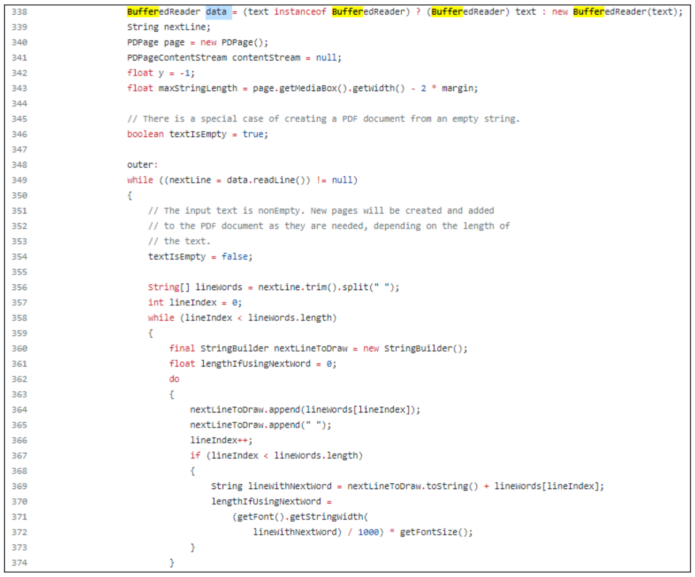
to create a buffered "data" from input "text" file and then convert the "text" file to a PDF file. In particular, the buffered "data" of the "text" file is transformed line by line to the PDF file.



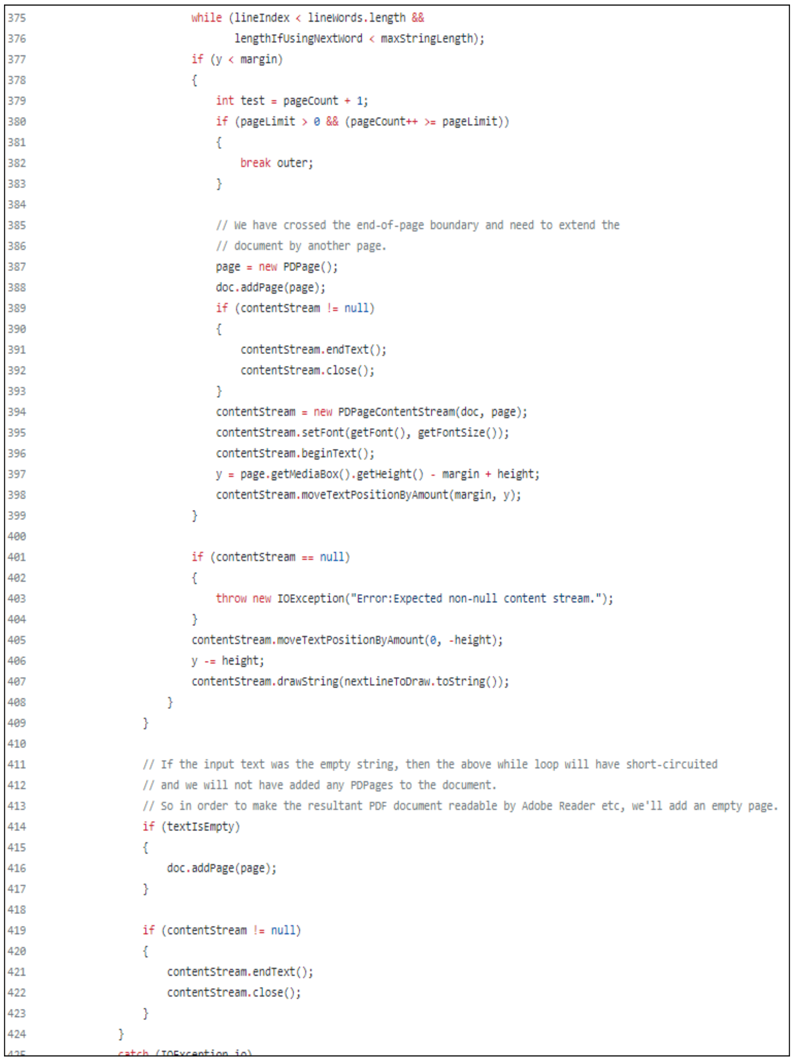
```
@Override
public void transform(final String sourceMimetype, final String targetMimetype, final Map<String, String> parameters,
                      final File sourceFile, final File targetFile) throws Exception
{
    String sourceEncoding = parameters.get(SOURCE_ENCODING);
    String stringPageLimit = parameters.get(PAGE_LIMIT);
    int pageLimit = -1;
    if (stringPageLimit != null)
    {
        pageLimit = parseInt(stringPageLimit, PAGE_LIMIT);
    }

    PDDocument pdf = null;
    try (InputStream is = new FileInputStream(sourceFile);
         Reader ir = new BufferedReader(buildReader(is, sourceEncoding));
         OutputStream os = new BufferedOutputStream(new FileOutputStream(targetFile)))
    {
        //TransformationOptionLimits limits = getLimits(reader, writer, options);
        //TransformationOptionPair pageLimits = limits.getPagesPair();
        pdf = transformer.createPDFFromText(ir, pageLimit);
        pdf.save(os);
    }
    finally
    {
        if (pdf != null)
        {
            try { pdf.close(); } catch (Throwable e) {e.printStackTrace(); }
        }
    }
}
```



```
    // The following code is based on the code in TextToPDF with the addition of
    // checks for page limits.
    // The calling code must close the PDDocument once finished with it.
    public PDDocument createPDFFromText(Reader text, int pageLimit)
        throws IOException
    {
        PDDocument doc = null;
        int pageCount = 0;
        try
        {
            final int margin = 40;
            float height = getFont().getFontDescriptor().getFontBoundingBox().getHeight() / 1000;

            //calculate font height and increase by 5 percent.
            height = height * getFontSize() * 1.05f;
            doc = new PDDocument();
            BufferedReader data = (text instanceof BufferedReader) ? (BufferedReader) text : new BufferedReader(text);
            String nextLine;
            PDPage page = new PDPage();
            PDPageContentStream contentStream = null;
            float y = -1;
            float maxStringLength = page.getMediaBox().getWidth() - 2 * margin;

            // There is a special case of creating a PDF document from an empty string.
            boolean textIsEmpty = true;
```



```
            BufferedReader data = (text instanceof BufferedReader) ? (BufferedReader) text : new BufferedReader(text);
            String nextLine;
            PDPage page = new PDPage();
            PDPageContentStream contentStream = null;
            float y = -1;
            float maxStringLength = page.getMediaBox().getWidth() - 2 * margin;

            // There is a special case of creating a PDF document from an empty string.
            boolean textIsEmpty = true;

            outer:
            while ((nextLine = data.readLine()) != null)
            {
                // The input text is nonEmpty. New pages will be created and added
                // to the PDF document as they are needed, depending on the length of
                // the text.
                textIsEmpty = false;

                String[] lineWords = nextLine.trim().split(" ");
                int lineIndex = 0;
                while (lineIndex < lineWords.length)
                {
                    final StringBuilder nextLineToDraw = new StringBuilder();
                    float lengthIfUsingNextWord = 0;
                    do
                    {
                        nextLineToDraw.append(lineWords[lineIndex]);
                        nextLineToDraw.append(" ");
                        lineIndex++;
                        if (lineIndex < lineWords.length)
                        {
                            String lineWithNextWord = nextLineToDraw.toString() + lineWords[lineIndex];
                            lengthIfUsingNextWord =
                                (getFont().getStringWidth(
                                    lineWithNextWord) / 1000) * getFontSize();
                        }
                    }
```

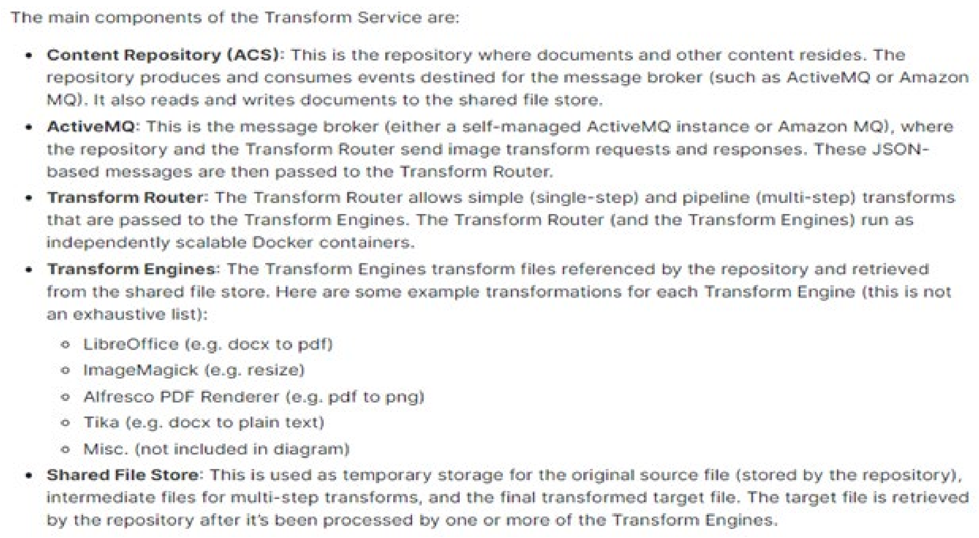


```
                        while (lineIndex < lineWords.length &&
                               lengthIfUsingNextWord < maxStringLength);
                        if (y < margin)
                        {
                            int test = pageCount + 1;
                            if (pageLimit > 0 && (pageCount++ >= pageLimit))
                            {
                                break outer;
                            }

                            // We have crossed the end-of-page boundary and need to extend the
                            // document by another page.
                            page = new PDPage();
                            doc.addPage(page);
                            if (contentStream != null)
                            {
                                contentStream.endText();
                                contentStream.close();
                            }
                            contentStream = new PDPageContentStream(doc, page);
                            contentStream.setFont(getFont(), getFontSize());
                            contentStream.beginText();
                            y = page.getMediaBox().getHeight() - margin + height;
                            contentStream.moveTextPositionByAmount(margin, y);
                        }

                        if (contentStream == null)
                        {
                            throw new IOException("Error:Expected non-null content stream.");
                        }
                        contentStream.moveTextPositionByAmount(0, -height);
                        y -= height;
                        contentStream.drawString(nextLineToDraw.toString());
                    }
                }

                // If the input text was the empty string, then the above while loop will have short-circuited
                // and we will not have added any PDPages to the document.
                // So in order to make the resultant PDF document readable by Adobe Reader etc, we'll add an empty page.
                if (textIsEmpty)
                {
                    doc.addPage(page);
                }

                if (contentStream != null)
                {
                    contentStream.endText();
                    contentStream.close();
                }
            }
            catch (IOException io)
```

(*See* https://github.com/Alfresco/alfresco-transform-core/blob/f24969199c13140f32ba976db65d6c15425fb3b0/alfresco-transform-misc/alfresco-transform-misc/src/main/java/org/alfresco/transformer/transformers/TextToPdfContentTransformer.java)

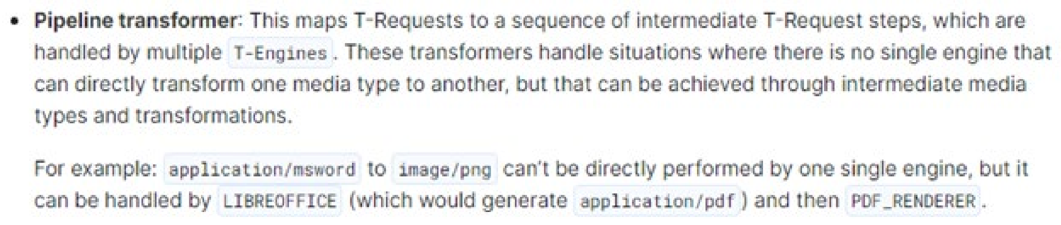
149. The Accused Products perform a method that includes *providing the unit of file data to a first transformation stream in the transformation pipeline, the providing performed by a write method associated with the first transformation stream*. As

explained above, the Alfresco Transform Service transforms an input file to other formats through a pipeline transformer if "there is no single engine that can directly transform one media type to another." For example, as input file application/msword cannot be directly transformed to output file image/png, the Alfresco Transform Service uses a first transform engine (*i.e.*, LibreOffice) to convert the input file application/msword to an intermediate PDF file. The Transform Engines retrieve files from the temporary storage (shared file store), where the original file and intermediate files are stored.


The main components of the Transform Service are:

- **Content Repository (ACS):** This is the repository where documents and other content resides. The repository produces and consumes events destined for the message broker (such as ActiveMQ or Amazon MQ). It also reads and writes documents to the shared file store.
- **ActiveMQ:** This is the message broker (either a self-managed ActiveMQ instance or Amazon MQ), where the repository and the Transform Router send image transform requests and responses. These JSON-based messages are then passed to the Transform Router.
- **Transform Router:** The Transform Router allows simple (single-step) and pipeline (multi-step) transforms that are passed to the Transform Engines. The Transform Router (and the Transform Engines) run as independently scalable Docker containers.
- **Transform Engines:** The Transform Engines transform files referenced by the repository and retrieved from the shared file store. Here are some example transformations for each Transform Engine (this is not an exhaustive list):
  - LibreOffice (e.g. docx to pdf)
  - ImageMagick (e.g. resize)
  - Alfresco PDF Renderer (e.g. pdf to png)
  - Tika (e.g. docx to plain text)
  - Misc. (not included in diagram)
- **Shared File Store:** This is used as temporary storage for the original source file (stored by the repository), intermediate files for multi-step transforms, and the final transformed target file. The target file is retrieved by the repository after it's been processed by one or more of the Transform Engines.

(*See* https://docs.alfresco.com/transform-service/1.4/admin/)

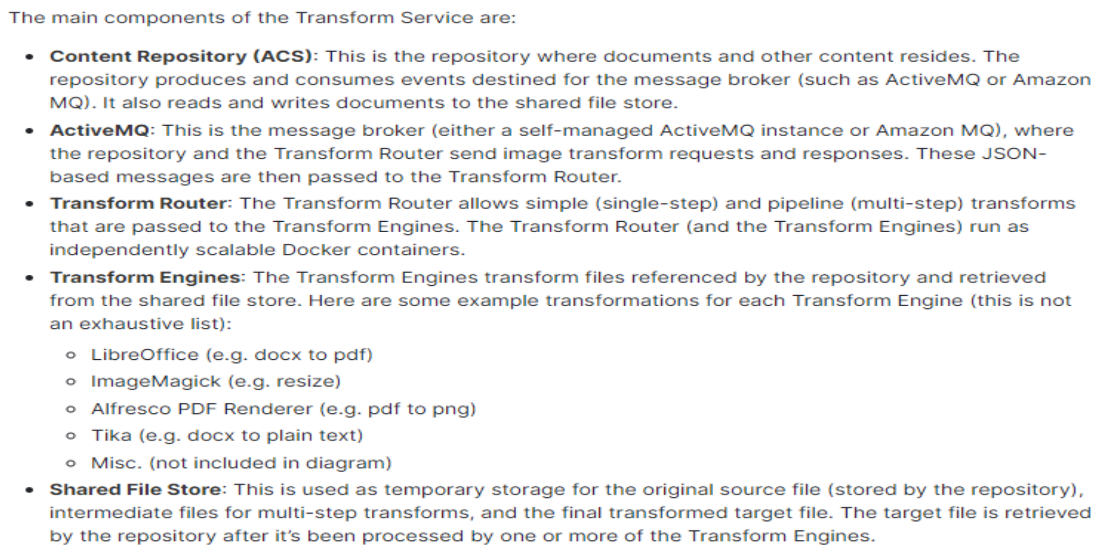

- **Pipeline transformer:** This maps T-Requests to a sequence of intermediate T-Request steps, which are handled by multiple `T-Engines`. These transformers handle situations where there is no single engine that can directly transform one media type to another, but that can be achieved through intermediate media types and transformations.

  For example: `application/msword` to `image/png` can't be directly performed by one single engine, but it can be handled by `LIBREOFFICE` (which would generate `application/pdf`) and then `PDF_RENDERER`.

(*See* https://docs.alfresco.com/transform-service/1.4/config/transformers/)

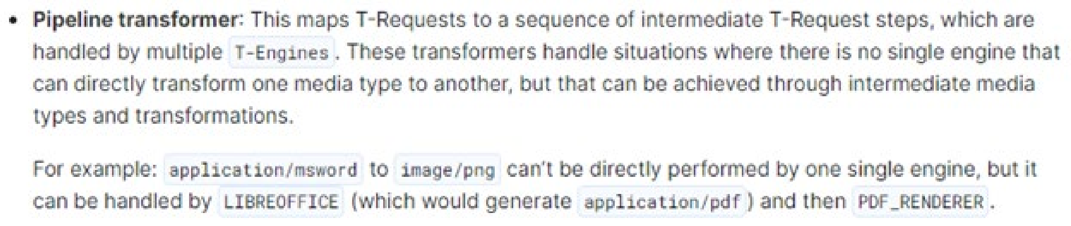
150. The Accused Products perform a method that includes *applying a transformation associated with the first transformation stream to the unit of file data, the applying performed by a read method associated with the first transformation stream, the applying including the read method calling a transformation function to perform the transformation*. As explained above, the Alfresco Transform Service stores

an input file in temporary storage (shared file store) temporarily for subsequent transformation steps. As an example, for transformation of application/msword to image/png, the Alfresco Transform Service retrieves the input file from the temporary storage (shared file store) and applies a first Transform Engine (i.e., LibreOffice) to convert the input application/msword file to an intermediate PDF file.


The main components of the Transform Service are:

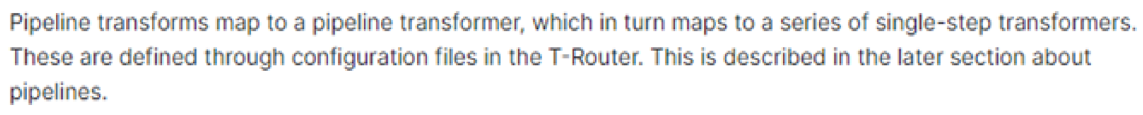
- **Content Repository (ACS)**: This is the repository where documents and other content resides. The repository produces and consumes events destined for the message broker (such as ActiveMQ or Amazon MQ). It also reads and writes documents to the shared file store.
- **ActiveMQ**: This is the message broker (either a self-managed ActiveMQ instance or Amazon MQ), where the repository and the Transform Router send image transform requests and responses. These JSON-based messages are then passed to the Transform Router.
- **Transform Router**: The Transform Router allows simple (single-step) and pipeline (multi-step) transforms that are passed to the Transform Engines. The Transform Router (and the Transform Engines) run as independently scalable Docker containers.
- **Transform Engines**: The Transform Engines transform files referenced by the repository and retrieved from the shared file store. Here are some example transformations for each Transform Engine (this is not an exhaustive list):
  - LibreOffice (e.g. docx to pdf)
  - ImageMagick (e.g. resize)
  - Alfresco PDF Renderer (e.g. pdf to png)
  - Tika (e.g. docx to plain text)
  - Misc. (not included in diagram)
- **Shared File Store**: This is used as temporary storage for the original source file (stored by the repository), intermediate files for multi-step transforms, and the final transformed target file. The target file is retrieved by the repository after it's been processed by one or more of the Transform Engines.

(*See* https://docs.alfresco.com/transform-service/1.4/admin/)

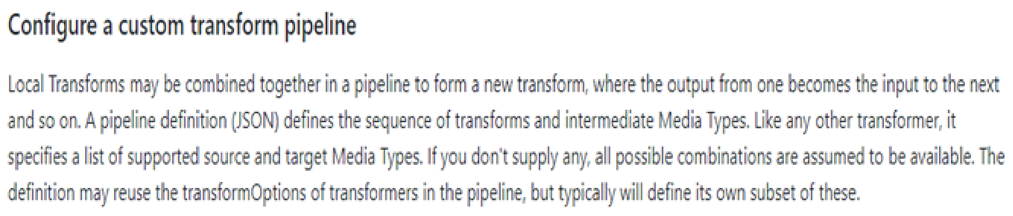

- **Pipeline transformer**: This maps T-Requests to a sequence of intermediate T-Request steps, which are handled by multiple `T-Engines`. These transformers handle situations where there is no single engine that can directly transform one media type to another, but that can be achieved through intermediate media types and transformations.

  For example: `application/msword` to `image/png` can't be directly performed by one single engine, but it can be handled by `LIBREOFFICE` (which would generate `application/pdf`) and then `PDF_RENDERER`.

(*See* https://docs.alfresco.com/transform-service/1.4/config/transformers/)


Pipeline transforms map to a pipeline transformer, which in turn maps to a series of single-step transformers. These are defined through configuration files in the T-Router. This is described in the later section about pipelines.

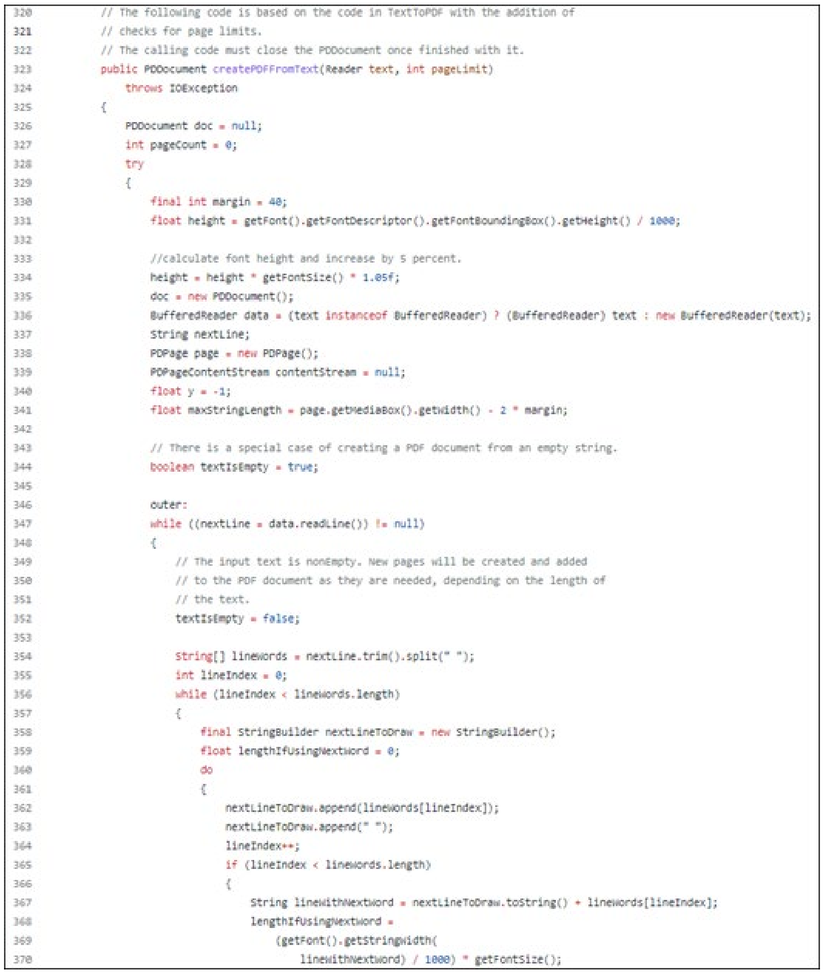
(*See* https://docs.alfresco.com/transform-service/1.4/config/transformers/)


Configure a custom transform pipeline

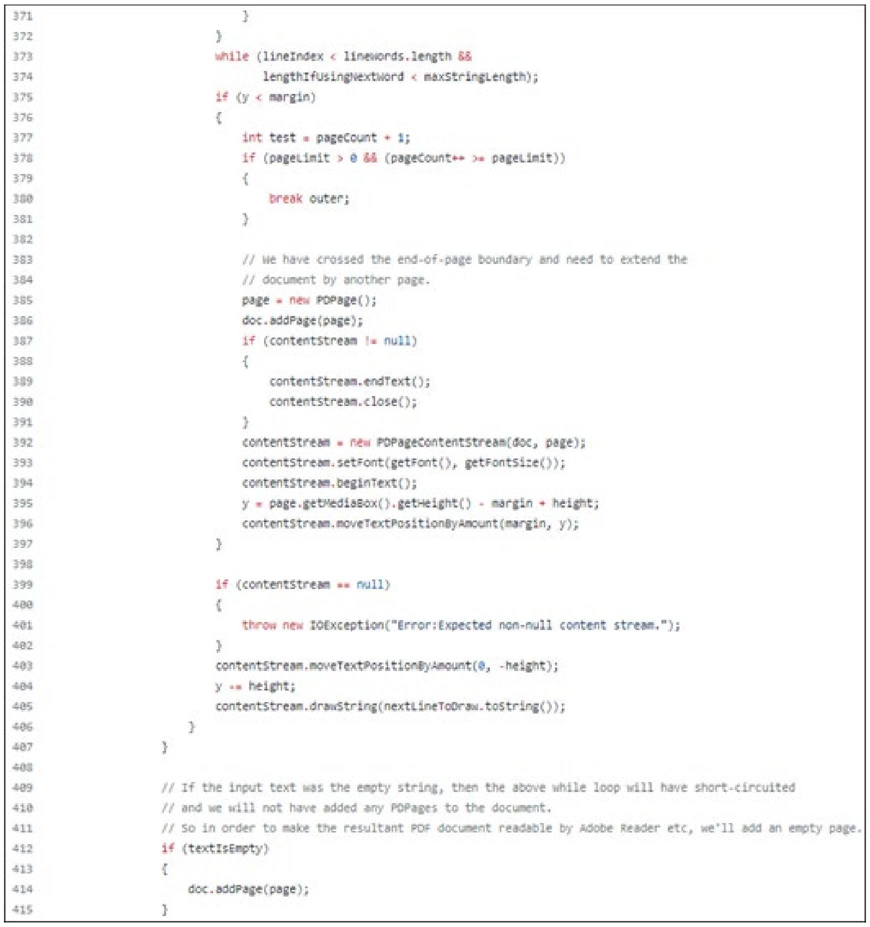
Local Transforms may be combined together in a pipeline to form a new transform, where the output from one becomes the input to the next and so on. A pipeline definition (JSON) defines the sequence of transforms and intermediate Media Types. Like any other transformer, it specifies a list of supported source and target Media Types. If you don't supply any, all possible combinations are assumed to be available. The definition may reuse the transformOptions of transformers in the pipeline, but typically will define its own subset of these.

(*See* https://github.com/Alfresco/acs-packaging/blob/release/6.2.N/docs/custom-transforms-and-renditions.md#configure-a-custom-transform-pipeline)

151. As an example, the Alfresco Transform Service provides a transformation function "createPDFFromText" that is performed line by line.



```
    // The following code is based on the code in TextToPDF with the addition of
    // checks for page limits.
    // The calling code must close the PDDocument once finished with it.
    public PDDocument createPDFFromText(Reader text, int pageLimit)
        throws IOException
    {
        PDDocument doc = null;
        int pageCount = 0;
        try
        {
            final int margin = 40;
            float height = getFont().getFontDescriptor().getFontBoundingBox().getHeight() / 1000;

            //calculate font height and increase by 5 percent.
            height = height * getFontSize() * 1.05f;
            doc = new PDDocument();
            BufferedReader data = (text instanceof BufferedReader) ? (BufferedReader) text : new BufferedReader(text);
            String nextLine;
            PDPage page = new PDPage();
            PDPageContentStream contentStream = null;
            float y = -1;
            float maxStringLength = page.getMediaBox().getWidth() - 2 * margin;

            // There is a special case of creating a PDF document from an empty string.
            boolean textIsEmpty = true;

            outer:
            while ((nextLine = data.readLine()) != null)
            {
                // The input text is nonEmpty. New pages will be created and added
                // to the PDF document as they are needed, depending on the length of
                // the text.
                textIsEmpty = false;

                String[] lineWords = nextLine.trim().split(" ");
                int lineIndex = 0;
                while (lineIndex < lineWords.length)
                {
                    final StringBuilder nextLineToDraw = new StringBuilder();
                    float lengthIfUsingNextWord = 0;
                    do
                    {
                        nextLineToDraw.append(lineWords[lineIndex]);
                        nextLineToDraw.append(" ");
                        lineIndex++;
                        if (lineIndex < lineWords.length)
                        {
                            String lineWithNextWord = nextLineToDraw.toString() + lineWords[lineIndex];
                            lengthIfUsingNextWord =
                                (getFont().getStringWidth(
                                    lineWithNextWord) / 1000) * getFontSize();
```

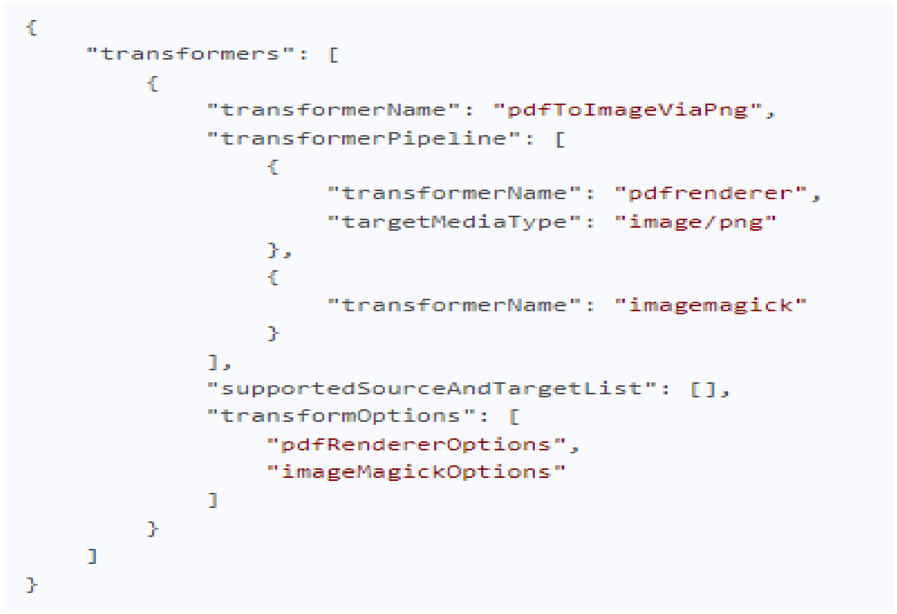


```
                }
            }
            while (lineIndex < lineWords.length &&
                    lengthIfUsingNextWord < maxStringLength);
            if (y < margin)
            {
                int test = pageCount + 1;
                if (pageLimit > 0 && (pageCount++ >= pageLimit))
                {
                    break outer;
                }

                // We have crossed the end-of-page boundary and need to extend the
                // document by another page.
                page = new PDPage();
                doc.addPage(page);
                if (contentStream != null)
                {
                    contentStream.endText();
                    contentStream.close();
                }
                contentStream = new PDPageContentStream(doc, page);
                contentStream.setFont(getFont(), getFontSize());
                contentStream.beginText();
                y = page.getMediaBox().getHeight() - margin + height;
                contentStream.moveTextPositionByAmount(margin, y);
            }

            if (contentStream == null)
            {
                throw new IOException("Error:Expected non-null content stream.");
            }
            contentStream.moveTextPositionByAmount(0, -height);
            y -= height;
            contentStream.drawString(nextLineToDraw.toString());
        }
    }

    // If the input text was the empty string, then the above while loop will have short-circuited
    // and we will not have added any PDPages to the document.
    // So in order to make the resultant PDF document readable by Adobe Reader etc, we'll add an empty page.
    if (textIsEmpty)
    {
        doc.addPage(page);
    }
```

(*See* https://github.com/Alfresco/alfresco-transform-core/blob/e575ec943a5fa5dddca5593e6795a17a2bbb3cb6/alfresco-transform-misc/alfresco-transform-misc/src/main/java/org/alfresco/transformer/transformers/TextToPdfContentTransformer.java)

152. On information and belief, the Accused Products perform a method that includes *performing the providing and the applying until all transformations in the set of transformations have been applied to the unit of file data through the sequence of transformation streams in the transformation pipeline*. As explained above, the Alfresco Transform Service transforms an input file from application/msword to image/png through a pipeline transformer if "there is no single engine that can directly transform one media type to another." As an example, a transformation pipeline may

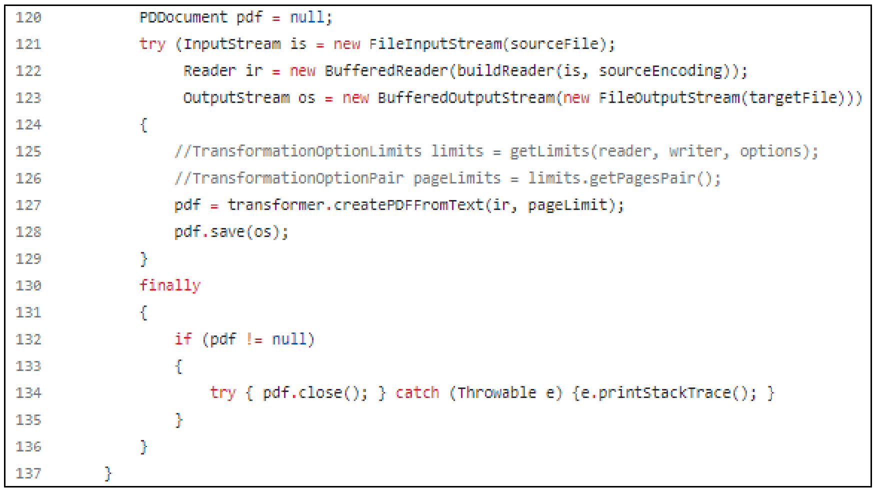
include a first Transform Engine (i.e., LibreOffice) and a second Transform Engine (i.e., Alfresco PDF Renderer). The transformed file obtained from the Transform Engine LibreOffice is input to the next sequenced Transform Engine (i.e., Alfresco PDF Renderer). (*See* https://docs.alfresco.com/transform-service/1.4/admin/; https://docs.alfresco.com/transform-service/1.4/config/transformers/; https://github.com/Alfresco/acs-packaging/blob/release/6.2.N/docs/custom-transforms-and-renditions.md#configure-a-custom-transform-pipeline.)



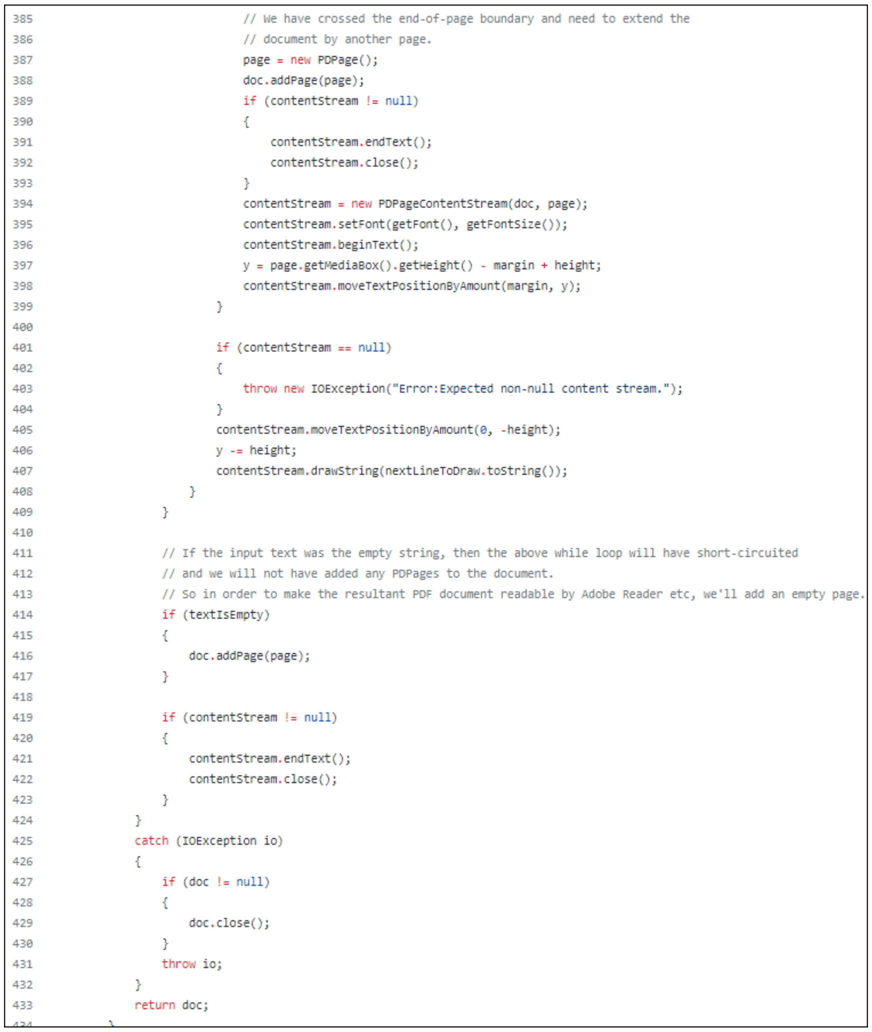
```
{
    "transformers": [
        {
            "transformerName": "pdfToImageViaPng",
            "transformerPipeline": [
                {
                    "transformerName": "pdfrenderer",
                    "targetMediaType": "image/png"
                },
                {
                    "transformerName": "imagemagick"
                }
            ],
            "supportedSourceAndTargetList": [],
            "transformOptions": [
                "pdfRendererOptions",
                "imageMagickOptions"
            ]
        }
    ]
}
```

(*See* https://docs.alfresco.com/transform-service/1.4/config/transformers/)

153. The Accused Products perform a method that includes *the first computing device sending the unit of file data so transformed by the set of transformations to a destination device*. As explained above, the Alfresco Transform Service sends the transformed file data to a temporary storage (shared file store) and content repository (ACS). The shared file store is "used as temporary storage for the original source file (stored by the repository), intermediate files for multi-step transforms, and the final transformed target file." (*See* https://docs.alfresco.com/transform-service/1.4/admin/) As an example, the transformed file data is sent through the "OutputStream os."



```
        PDDocument pdf = null;
        try (InputStream is = new FileInputStream(sourceFile);
             Reader ir = new BufferedReader(buildReader(is, sourceEncoding));
             OutputStream os = new BufferedOutputStream(new FileOutputStream(targetFile)))
        {
            //TransformationOptionLimits limits = getLimits(reader, writer, options);
            //TransformationOptionPair pageLimits = limits.getPagesPair();
            pdf = transformer.createPDFFromText(ir, pageLimit);
            pdf.save(os);
        }
        finally
        {
            if (pdf != null)
            {
                try { pdf.close(); } catch (Throwable e) {e.printStackTrace(); }
            }
        }
    }
```



```
                        // We have crossed the end-of-page boundary and need to extend the
                        // document by another page.
                        page = new PDPage();
                        doc.addPage(page);
                        if (contentStream != null)
                        {
                            contentStream.endText();
                            contentStream.close();
                        }
                        contentStream = new PDPageContentStream(doc, page);
                        contentStream.setFont(getFont(), getFontSize());
                        contentStream.beginText();
                        y = page.getMediaBox().getHeight() - margin + height;
                        contentStream.moveTextPositionByAmount(margin, y);
                    }

                    if (contentStream == null)
                    {
                        throw new IOException("Error:Expected non-null content stream.");
                    }
                    contentStream.moveTextPositionByAmount(0, -height);
                    y -= height;
                    contentStream.drawString(nextLineToDraw.toString());
                }
            }

            // If the input text was the empty string, then the above while loop will have short-circuited
            // and we will not have added any PDPages to the document.
            // So in order to make the resultant PDF document readable by Adobe Reader etc, we'll add an empty page.
            if (textIsEmpty)
            {
                doc.addPage(page);
            }

            if (contentStream != null)
            {
                contentStream.endText();
                contentStream.close();
            }
        }
        catch (IOException io)
        {
            if (doc != null)
            {
                doc.close();
            }
            throw io;
        }
        return doc;
```

(*See* https://github.com/Alfresco/alfresco-transform-core/blob/f24969199c13140f32ba976db65d6c15425fb3b0/alfresco-transform-misc/alfresco-transform-misc/src/main/java/org/alfresco/transformer/transformers/TextToPdfContentTransformer.java)

154. Each claim in the ’381 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the ’381 Patent.

155. On information and belief, there has been significant effort by Hyland to imitate OpenText’s patent-protected products to compete with OpenText in the ECM and EIM markets and to increase Hyland’s share of that market at the expense of OpenText’s market share. (*See, e.g.*, Exhibit B (2020.09.09 - Hyland enters definitive agreement to acquire Alfresco, hyland.com), Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com).) Hyland’s efforts have resulted in the Accused Products, which infringe at least claim 1 of the ’381 patent as described above, and those efforts would have exposed Hyland to the ’381 patent prior to the filing of the original Complaint in this action.

156. Hyland has known of the ’381 Patent since receiving a letter identifying the patent and the infringement on September 2, 2022. At the very least, Hyland has been aware of the ’381 patent and of its infringement based on the Accused Products since at least the filing and/or service of this Complaint.

157. On information and belief, at least as of the filing of the Complaint in this action, Hyland has knowingly and actively induced and is knowingly and actively inducing at least its customers and partners to directly infringe at least claim 1 of the ’381 patent, and has done so with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b), by activities relating to selling, marketing, advertising, promoting, supporting, installing, and distributing its Accused Products in the United States. (Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D

(2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com).) On information and belief, those activities continue.

158. On information and belief, Hyland deliberately and knowingly encourages, instructs, directs, and/or requires third parties—including its partners, customers, and/or end users—to use the Accused Products in a way that infringes at least claim 1 of the ’726 patent as described above.

159. Hyland’s partners, customers, and end users of its Accused Products directly infringe at least claim 1 of the ’381 patent, at least by using the Accused Products, as described above.

160. For example, on information and belief, Hyland knowingly and intentionally shares instructions, guides, and manuals, including through its website, training programs, and/or YouTube, which advertise and instruct third parties on how to use the Accused Products in a way that directly infringes at least claim 1 of the ’381 patent as described above, including at least Hyland’s customers. On further information and belief, Hyland knowingly and intentionally provides customer service or technical support to purchasers of the infringing Accused Products, which directs and encourages Hyland’s customers to use the Accused Products in a way that directly infringes at least claim 1 of the ’381 patent as described above.

161. On information and belief, the infringing actions of each customer and/or end-user of the Accused Products are attributable to Hyland.

162. On information and belief, Hyland sells and offers for sale the Accused Products and provides technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. On information and belief, each customer enters into a contractual relationship with Hyland, which obligates each customer to perform certain actions as a condition to use the Accused Products. Further, in order to receive the benefit of Hyland’s continued technical support and its specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the

Accused Products in a way that infringes the ’381 patent. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Products remain operational for each customer through ongoing technical support, on information and belief, Hyland establishes the manner and timing of each customer’s performance of activities that infringe the ’381 patent.

163. On information and belief, Hyland forms a joint enterprise with its customers to engage in directly infringing the ’381 patent. On further information and belief, Hyland together with each customer operate under a contractual agreement; have a common purpose to operate the Accused Products in a way that directly infringes the ’381 patent as outlined in the paragraphs above; have pecuniary interests in operating the Accused Products by directly profiting from the sale and/or maintenance of the Accused Products or by indirectly profiting from the increased efficiency resulting from use of the Accused Products; and have equal rights to a voice in the direction of the enterprise either by guiding and advising on the operation and capabilities of the Accused Products with product-specific know-how and expertise or by requesting that certain customer-specific capabilities be implemented in the Accused Products.

164. Hyland also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the inventions claimed, and not a staple article or commodity of commerce suitable for substantial non-infringing uses.

165. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including the transformation of files from their current format into other formats and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. These processes are continually running

when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’381 patent, that functionality could not be performed.

166. Additionally, the accused functionality, including the transformation of files from their current format into other formats and related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For example, without the Alfresco Transformation Services, the Accused Products could not convert files from one format to another, including metadata. These processes are continually running when the system is in use and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’381 Patent, that functionality could not be performed.

167. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Hyland constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionalities (including the file transformation functionality) constitute a material part of the inventions claimed because such functionality is integral to the processes identified above (such as “receiving a request from a second computing device to read or write a file,” “creating a transformation pipeline,” “providing the unit of file data to a first transformation stream,” “applying a transformation”, and “performing the applying until all transformations have been applied”) as recited in the claims of the ’381 Patent. None of these products are staple goods—they are sophisticated and customized ECM products, methods, and systems.

168. OpenText "consists of four revenue streams: license, cloud services and subscriptions, customer support, and professional service and other." (Exhibit A at 9-10 (Aug. 6, 2020 10-K).) Each revenue stream relates directly to the ability of OpenText to acquire and retain customers for its software products in a market that is "highly competitive" and increasingly more competitive "as a result of ongoing software industry consolidation," such as Hyland's acquisition of Alfresco. (Exhibit A at 11 (Aug. 6, 2020 10-K); *see also* Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com); Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com); Exhibit F at 4 ("The Forrester Wave: ECM Content Platforms, Q3 2019"); Exhibit E at 3 (2020.11.16 - Gartner Content Services Report 2020).) OpenText is an innovator in the market and has acquired multiple patents, including the Patents-in-Suit, to give it an advantage over such competition. Hyland's infringing activities have resulted and will continue to result in irreparable harm to OpenText because of the competitive threat that Hyland—including Hyland's acquisition of Alfresco—has to OpenText's share of the relevant "highly competitive" market, and the impact that Hyland's infringing activities have on each one of OpenText's four revenue streams. Further, public interest factors favor OpenText as the owner and assignee of government-issued patents, including the Patents-in-Suit, that serve to recognize OpenText's innovative contribution to the public knowledge in exchange for the patent protection that Hyland is now infringing.

169. For past infringement, OpenText has suffered damages, including lost profits, as a result of Hyland's infringement of the '381 patent. Hyland is therefore liable to OpenText under 35 U.S.C. § 284 for past damages in an amount that adequately compensates OpenText for Hyland's infringement, but no less than a reasonable royalty.

170. OpenText is entitled to a preliminary injunction to maintain the status quo between OpenText and Hyland, which, through its acquisition of Alfresco, is now one of OpenText's biggest competitors (*see, e.g.*, Exhibit B (2020.09.09 - Hyland enters

definitive agreement to acquire Alfresco, hyland.com), Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com)), and is using OpenText's patented technology to compete with OpenText in the ECM and EIM markets.

171. For ongoing and future infringement, OpenText will continue to suffer irreparable harm, including without limitation, loss of market share, customers and/or convoyed sales and services which cannot be accurately quantified nor adequately compensated for by money damages, unless this Court preliminarily and permanently enjoins Hyland, its agents, employees, representatives, and all others acting in concert with Hyland from infringing the '381 patent.

172. In the alternative, OpenText is entitled to damages in lieu of an injunction, in an amount consistent with the facts, for future infringement. Hyland's continued infringement, at least since it had notice of the '381 patent, is knowing and willful. Hyland will be an adjudicated infringer of a valid patent and, thus, Hyland's future infringement will be willful as a matter of law.

173. Hyland's infringement is without license or other authorization.

174. This case is exceptional, entitling Plaintiffs to enhanced damages under 35 U.S.C. § 284 and an award of attorneys' fees and costs incurred in prosecuting this action under 35 U.S.C. § 285.

FOURTH CAUSE OF ACTION

(INFRINGEMENT OF THE '786 PATENT)

175. Plaintiffs reallege and incorporate by reference the allegations of the preceding paragraphs of this Complaint.

176. Defendants have infringed and continue to infringe one or more claims of the '786 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including features such as Alfresco's Application Development Framework

("ADF"), at least when used for their ordinary and customary purposes, practice each element of at least claim 1 of the '786 Patent as described below.

177. Claim 1 of the '786 Patent recites:

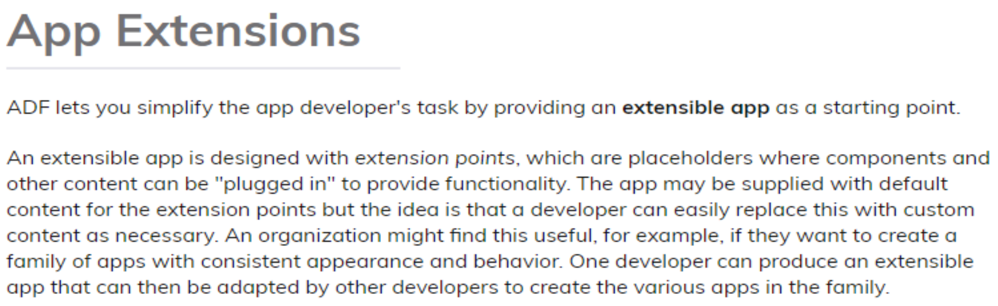
1. A method comprising:

receiving via a software development tool interface a definition of a context menu option and an associated action; and

generating, using one or more processors, programmatically based at least in part on the definition an application code to implement the context menu at runtime, including performing processing at runtime as defined in the definition to determine one or both of the context menu option and the associated action, and to create an invisible object that is associated with an application page during execution of the application page, wherein the invisible object provides, to the context menu, information with which the context menu is updated during execution of the application page, and

and wherein the context menu is displayed in response to detection of a predetermined event during execution of the application page.

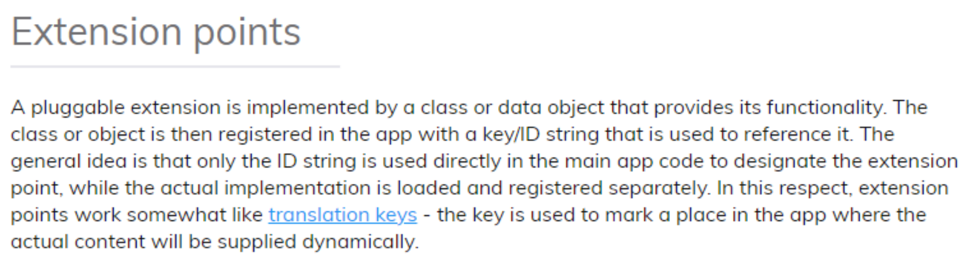
178. To the extent the preamble is construed to be limiting, the Accused Products perform a *method* as further explained below. For example, the Accused Products perform a method for plugging context menus into an "extensible app," where definitions for context menus and other components are incorporated into "extension points" within that app.


App Extensions

ADF lets you simplify the app developer's task by providing an **extensible app** as a starting point.

An extensible app is designed with *extension points*, which are placeholders where components and other content can be "plugged in" to provide functionality. The app may be supplied with default content for the extension points but the idea is that a developer can easily replace this with custom content as necessary. An organization might find this useful, for example, if they want to create a family of apps with consistent appearance and behavior. One developer can produce an extensible app that can then be adapted by other developers to create the various apps in the family.

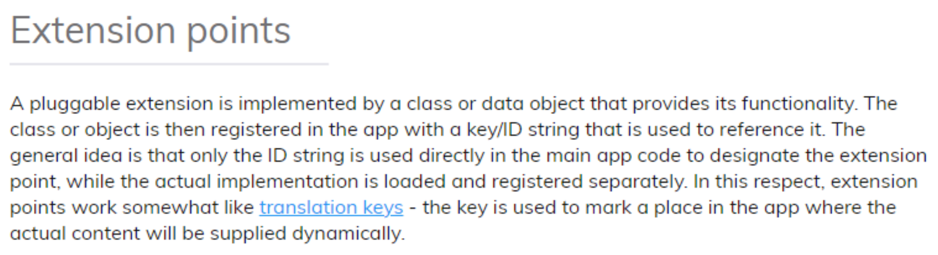
(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/)


Extension points

A pluggable extension is implemented by a class or data object that provides its functionality. The class or object is then registered in the app with a key/ID string that is used to reference it. The general idea is that only the ID string is used directly in the main app code to designate the extension point, while the actual implementation is loaded and registered separately. In this respect, extension points work somewhat like translation keys - the key is used to mark a place in the app where the actual content will be supplied dynamically.

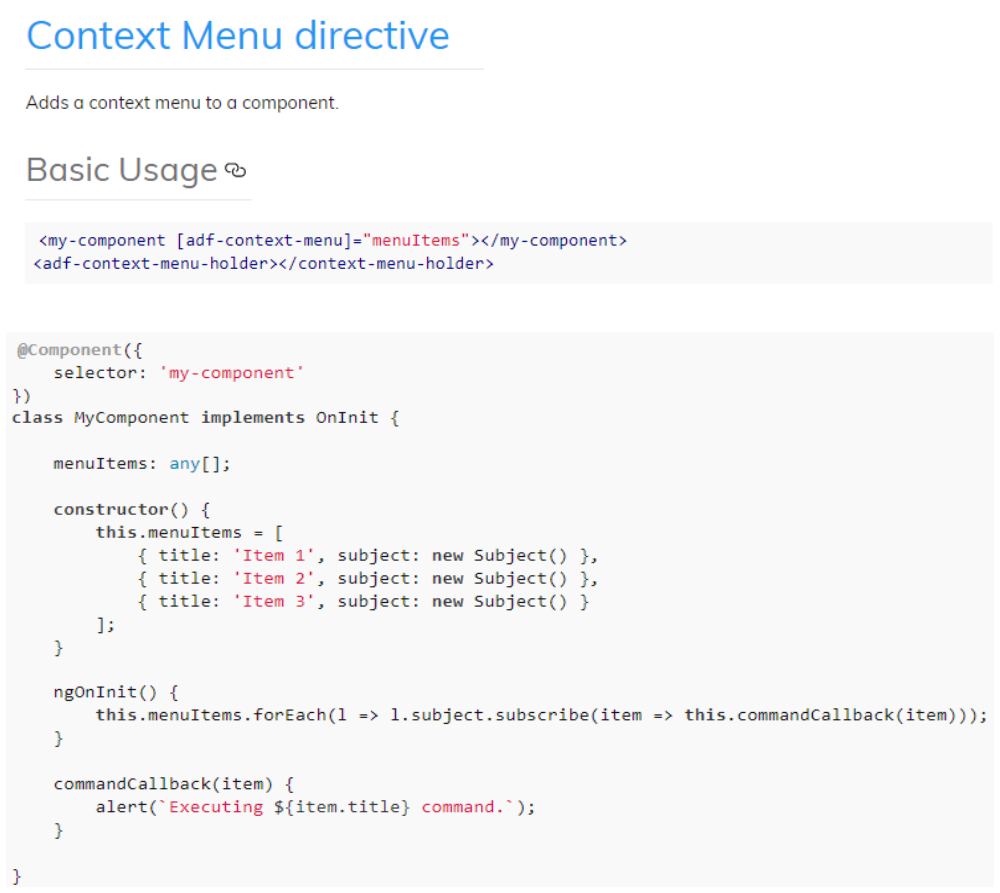
(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/.)

179. The Accused Products perform a method of *receiving via a software development tool interface a definition of a context menu option and an associated action* and *generating, using one or more processors, programmatically based at least in part on the definition an application code to implement the context menu at runtime, including performing processing at runtime as defined in the definition to determine one or both of the context menu option and the associated action*. In the example shown below of one possible encoding of a "context menu" within the Accused Products, the "items" or options comprised by that menu and a set of actions associated with those options is shown as being loaded in as a "dynamic component" into an "extensible app." That context menu is plugged into the app at an "extension point" and referred to only by its "key" or "ID string" within the "main app code," "while the actual implementation is loaded and registered separately." An "extension point…is used to mark a place in the app where the actual content will be supplied dynamically." That context menu "dynamically" is loaded at run time, including the actions to be performed that are associated with that menu's options.


Extension points

A pluggable extension is implemented by a class or data object that provides its functionality. The class or object is then registered in the app with a key/ID string that is used to reference it. The general idea is that only the ID string is used directly in the main app code to designate the extension point, while the actual implementation is loaded and registered separately. In this respect, extension points work somewhat like translation keys - the key is used to mark a place in the app where the actual content will be supplied dynamically.

(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/.)



Context Menu directive

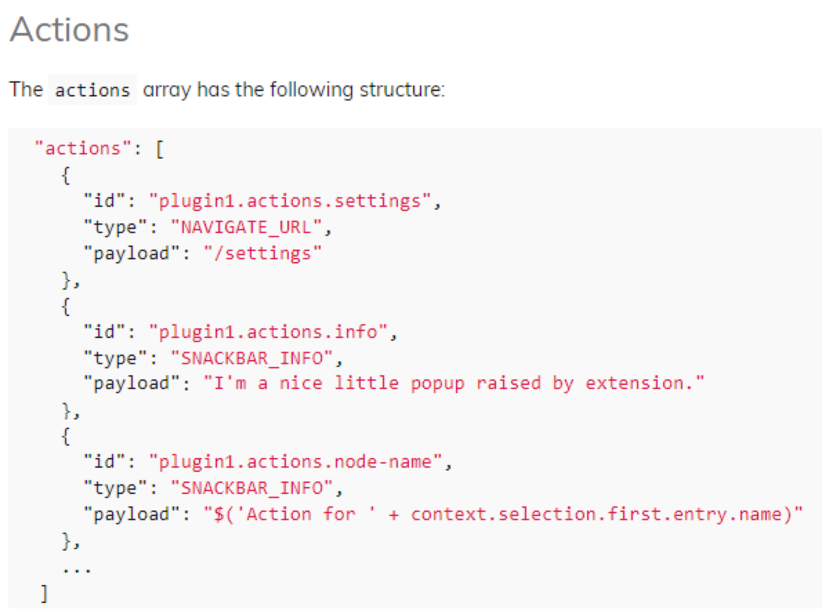
Adds a context menu to a component.

Basic Usage

```
<my-component [adf-context-menu]="menuItems"></my-component>
<adf-context-menu-holder></context-menu-holder>
```

```
@Component({
    selector: 'my-component'
})
class MyComponent implements OnInit {

    menuItems: any[];

    constructor() {
        this.menuItems = [
            { title: 'Item 1', subject: new Subject() },
            { title: 'Item 2', subject: new Subject() },
            { title: 'Item 3', subject: new Subject() }
        ];
    }

    ngOnInit() {
        this.menuItems.forEach(l => l.subject.subscribe(item => this.commandCallback(item)));
    }

    commandCallback(item) {
        alert(`Executing ${item.title} command.`);
    }

}
```

(*See* https://www.alfresco.com/abn/adf/docs/core/directives/context-menu.directive/).

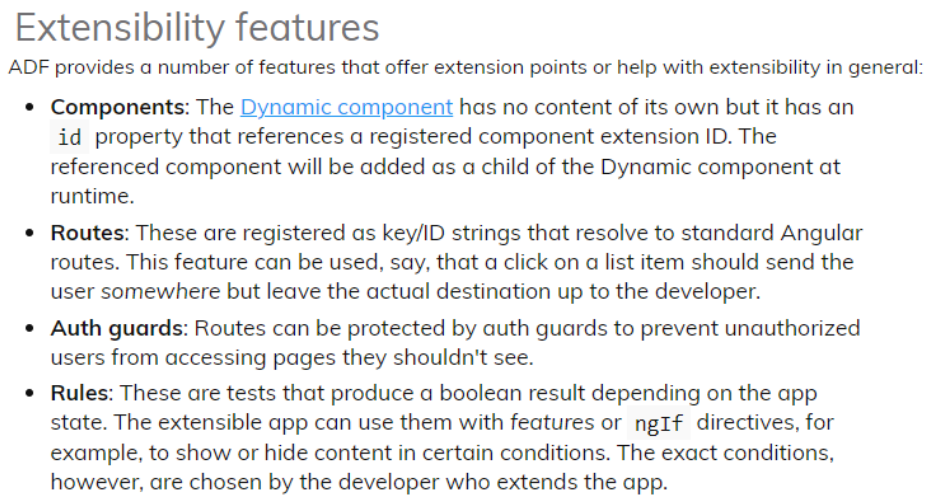


Actions

The `actions` array has the following structure:

```
"actions": [
  {
    "id": "plugin1.actions.settings",
    "type": "NAVIGATE_URL",
    "payload": "/settings"
  },
  {
    "id": "plugin1.actions.info",
    "type": "SNACKBAR_INFO",
    "payload": "I'm a nice little popup raised by extension."
  },
  {
    "id": "plugin1.actions.node-name",
    "type": "SNACKBAR_INFO",
    "payload": "$('Action for ' + context.selection.first.entry.name)"
  },
  ...
]
```

(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/).

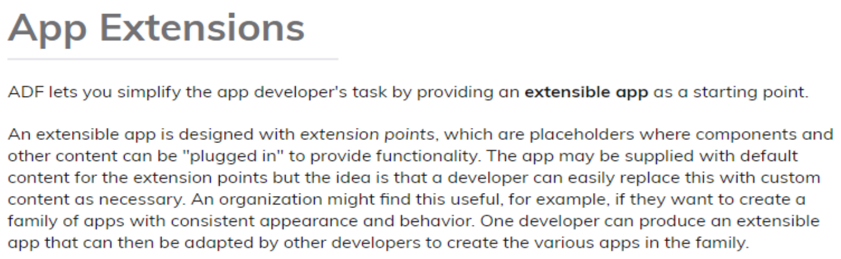


Extensibility features

ADF provides a number of features that offer extension points or help with extensibility in general:

- **Components**: The Dynamic component has no content of its own but it has an `id` property that references a registered component extension ID. The referenced component will be added as a child of the Dynamic component at runtime.
- **Routes**: These are registered as key/ID strings that resolve to standard Angular routes. This feature can be used, say, that a click on a list item should send the user somewhere but leave the actual destination up to the developer.
- **Auth guards**: Routes can be protected by auth guards to prevent unauthorized users from accessing pages they shouldn't see.
- **Rules**: These are tests that produce a boolean result depending on the app state. The extensible app can use them with features or `ngIf` directives, for example, to show or hide content in certain conditions. The exact conditions, however, are chosen by the developer who extends the app.

(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/).



App Extensions

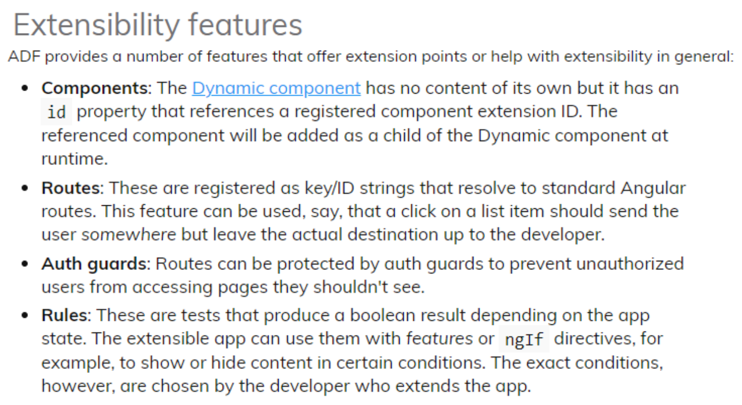
ADF lets you simplify the app developer's task by providing an **extensible app** as a starting point.

An extensible app is designed with extension points, which are placeholders where components and other content can be "plugged in" to provide functionality. The app may be supplied with default content for the extension points but the idea is that a developer can easily replace this with custom content as necessary. An organization might find this useful, for example, if they want to create a family of apps with consistent appearance and behavior. One developer can produce an extensible app that can then be adapted by other developers to create the various apps in the family.

(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/)

180. The application code generated by the Accused Products is also used *to create an invisible object that is associated with an application page during execution of the application page, wherein the invisible object provides, to the context menu, information with which the context menu is updated during execution of the application page, wherein the context menu is displayed in response to detection of a predetermined event during execution of the application page.* For example, each context menu is loaded into an invisible "Dynamic component" container that is used to add the context menu "as a child of the Dynamic component at runtime." Similarly, an invisible "feature" container can also be used to load in, by referencing its "ID," a particular context menu and to specify certain actions for that menu to perform when interacted with by a user. These containers also specify predetermined conditions that must be satisfied before the context menu is rendered "visible." In the example below, the

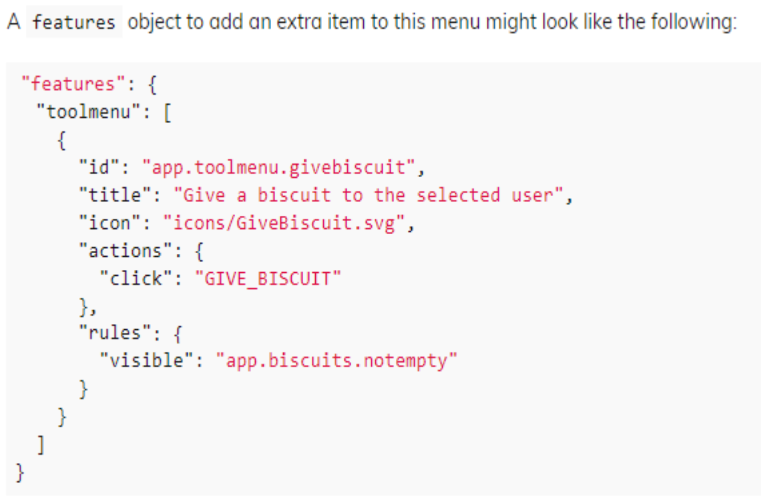
context menu "tool" is rendered visible if the "biscuits" feature within the app is "not empty."


Extensibility features

ADF provides a number of features that offer extension points or help with extensibility in general:

- **Components**: The Dynamic component has no content of its own but it has an `id` property that references a registered component extension ID. The referenced component will be added as a child of the Dynamic component at runtime.
- **Routes**: These are registered as key/ID strings that resolve to standard Angular routes. This feature can be used, say, that a click on a list item should send the user somewhere but leave the actual destination up to the developer.
- **Auth guards**: Routes can be protected by auth guards to prevent unauthorized users from accessing pages they shouldn't see.
- **Rules**: These are tests that produce a boolean result depending on the app state. The extensible app can use them with features or `ngIf` directives, for example, to show or hide content in certain conditions. The exact conditions, however, are chosen by the developer who extends the app.

(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/).


A `features` object to add an extra item to this menu might look like the following:

```
"features": {
  "toolmenu": [
    {
      "id": "app.toolmenu.givebiscuit",
      "title": "Give a biscuit to the selected user",
      "icon": "icons/GiveBiscuit.svg",
      "actions": {
        "click": "GIVE_BISCUIT"
      },
      "rules": {
        "visible": "app.biscuits.notempty"
      }
    }
  ]
}
```

(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/).

181. Each claim in the '786 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the '786 Patent.

182. On information and belief, there has been significant effort by Hyland to imitate OpenText's patent-protected products to compete with OpenText in the ECM and EIM markets and to increase Hyland's share of that market at the expense of OpenText's market share. (*See, e.g.*, Exhibit B (2020.09.09 - Hyland enters definitive agreement to acquire Alfresco, hyland.com), Exhibit C (2020.10.22 - Hyland completes

acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com).) Hyland's efforts have resulted in the Accused Products, which infringe at least claim 1 of the '786 patent as described above, and those efforts would have exposed Hyland to the '786 patent prior to the filing of the original Complaint in this action.

183. Hyland has known of the '786 Patent since receiving a letter identifying the patent and the infringement on September 2, 2022. At the very least, Hyland has been aware of the '786 patent and of its infringement based on the Accused Products since at least the filing and/or service of this Complaint. Further, OpenText marks its products with the '786 patent.

184. On information and belief, at least as of the filing of the Complaint in this action, Hyland has knowingly and actively induced and is knowingly and actively inducing at least its customers and partners to directly infringe at least claim 1 of the '786 patent, and has done so with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b), by activities relating to selling, marketing, advertising, promoting, supporting, installing, and distributing its Accused Products in the United States. (Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com).) On information and belief, those activities continue.

185. On information and belief, Hyland deliberately and knowingly encourages, instructs, directs, and/or requires third parties—including its partners, customers, and/or end users—to use the Accused Products in a way that infringes at least claim 1 of the '726 patent as described above.

186. Hyland's partners, customers, and end users of its Accused Products directly infringe at least claim 1 of the '786 patent, at least by using the Accused Products, as described above.

187. For example, on information and belief, Hyland knowingly and intentionally shares instructions, guides, and manuals, including through its website, training programs, and/or YouTube, which advertise and instruct third parties on how to use the Accused Products in a way that directly infringes at least claim 1 of the ’786 patent as described above, including at least Hyland’s customers. On further information and belief, Hyland knowingly and intentionally provides customer service or technical support to purchasers of the infringing Accused Products, which directs and encourages Hyland’s customers to use the Accused Products in a way that directly infringes at least claim 1 of the ’786 patent as described above.

188. On information and belief, the infringing actions of each customer and/or end-user of the Accused Products are attributable to Hyland.

189. On information and belief, Hyland sells and offers for sale the Accused Products and provides technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. On information and belief, each customer enters into a contractual relationship with Hyland, which obligates each customer to perform certain actions as a condition to use the Accused Products. Further, in order to receive the benefit of Hyland’s continued technical support and its specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that infringes the ’786 patent. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Products remain operational for each customer through ongoing technical support, on information and belief, Hyland establishes the manner and timing of each customer’s performance of activities that infringe the ’786 patent.

190. On information and belief, Hyland forms a joint enterprise with its customers to engage in directly infringing the ’786 patent. On further information and belief, Hyland together with each customer operate under a contractual agreement; have

a common purpose to operate the Accused Products in a way that directly infringes the ’786 patent as outlined in the paragraphs above; have pecuniary interests in operating the Accused Products by directly profiting from the sale and/or maintenance of the Accused Products or by indirectly profiting from the increased efficiency resulting from use of the Accused Products; and have equal rights to a voice in the direction of the enterprise either by guiding and advising on the operation and capabilities of the Accused Products with product-specific know-how and expertise or by requesting that certain customer-specific capabilities be implemented in the Accused Products.

191. Hyland also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the inventions claimed, and not a staple article or commodity of commerce suitable for substantial non-infringing uses.

192. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including the development and implementation of context menus and invisible objects and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. These processes are part of the development framework and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’786 patent, that functionality could not be performed.

193. Additionally, the accused functionality, including the development and implementation of context menus and invisible objects and related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For

example, without the context menus and invisible objects, the Accused Products could not provide the dynamic components and features of a user interface. On information and belief, these features cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’786 Patent, that functionality could not be performed.

194. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Hyland constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionalities (including the functionality for developing and implementing context menus and invisible objects) constitute a material part of the inventions claimed because such functionality is integral to the processes identified above (such as to receive “definition of a context menu option and an associated action,” generate “code to implement the context menu at runtime” and create “an invisible object that is associated with an application page during execution of the application page”) as recited in the claims of the ’786 Patent. None of these products are staple goods—they are sophisticated and customized application development and ECM products, methods, and systems.

195. OpenText “consists of four revenue streams: license, cloud services and subscriptions, customer support, and professional service and other.” (Exhibit A at 9-10 (Aug. 6, 2020 10-K).) Each revenue stream relates directly to the ability of OpenText to acquire and retain customers for its software products in a market that is “highly competitive” and increasingly more competitive “as a result of ongoing software industry consolidation,” such as Hyland’s acquisition of Alfresco. (Exhibit A at 11 (Aug. 6, 2020 10-K); *see also* Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com); Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com); Exhibit F at 4 (“The Forrester

Wave: ECM Content Platforms, Q3 2019"); Exhibit E at 3 (2020.11.16 - Gartner Content Services Report 2020).) OpenText is an innovator in the market and has acquired multiple patents, including the Patents-in-Suit, to give it an advantage over such competition. Hyland's infringing activities have resulted and will continue to result in irreparable harm to OpenText because of the competitive threat that Hyland—including Hyland's acquisition of Alfresco—has to OpenText's share of the relevant "highly competitive" market, and the impact that Hyland's infringing activities have on each one of OpenText's four revenue streams. Further, public interest factors favor OpenText as the owner and assignee of government-issued patents, including the Patents-in-Suit, that serve to recognize OpenText's innovative contribution to the public knowledge in exchange for the patent protection that Hyland is now infringing.

196. For past infringement, OpenText has suffered damages, including lost profits, as a result of Hyland's infringement of the '786 patent. Hyland is therefore liable to OpenText under 35 U.S.C. § 284 for past damages in an amount that adequately compensates OpenText for Hyland's infringement, but no less than a reasonable royalty.

197. OpenText is entitled to a preliminary injunction to maintain the status quo between OpenText and Hyland, which, through its acquisition of Alfresco, is now one of OpenText's biggest competitors (*see, e.g.*, Exhibits 24, Exhibit B (2020.09.09 - Hyland enters definitive agreement to acquire Alfresco, hyland.com), Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com)), and is using OpenText's patented technology to compete with OpenText in the ECM and EIM markets.

198. For ongoing and future infringement, OpenText will continue to suffer irreparable harm, including without limitation, loss of market share, customers and/or convoyed sales and services which cannot be accurately quantified nor adequately compensated for by money damages, unless this Court preliminarily and permanently

enjoins Hyland, its agents, employees, representatives, and all others acting in concert with Hyland from infringing the ’786 patent.

199. In the alternative, OpenText is entitled to damages in lieu of an injunction, in an amount consistent with the facts, for future infringement. Hyland’s continued infringement, at least since it had notice of the ’786 patent, is knowing and willful. Hyland will be an adjudicated infringer of a valid patent and, thus, Hyland’s future infringement will be willful as a matter of law.

200. Hyland’s infringement is without license or other authorization.

201. This case is exceptional, entitling Plaintiffs to enhanced damages under 35 U.S.C. § 284 and an award of attorneys’ fees and costs incurred in prosecuting this action under 35 U.S.C. § 285.

FIFTH CAUSE OF ACTION

(INFRINGEMENT OF THE ’150 PATENT)

202. Plaintiffs reallege and incorporate by reference the allegations of the preceding paragraphs of this Complaint.

203. Alfresco has infringed and continues to infringe one or more claims of the ’786 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including features such as Alfresco’s Application Development Framework (“ADF”), at least when used for their ordinary and customary purposes, practice each element of at least claim 1 of the ’150 Patent as described below.

204. Claim 1 of the ’150 Patent recites:

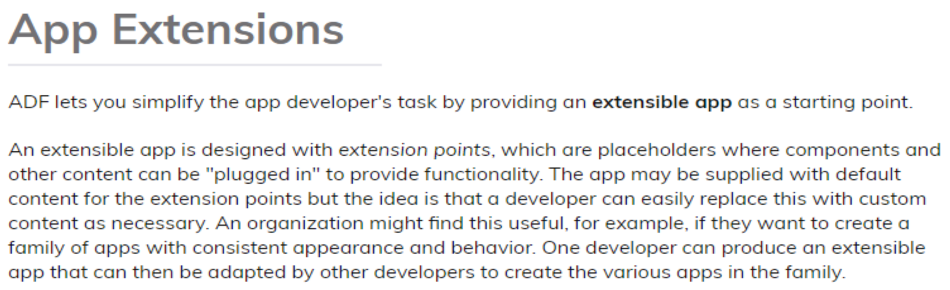
1. A method comprising:

receiving via a software development tool interface a context menu option definition; and

generating, using one or more processors, based at least in part on the context menu option definition, an application code to implement the context menu at runtime, including performing processing at runtime and to create an invisible object that is associated with an application page during execution of the application page,

wherein the invisible object is configured to consume an event during display of the application page, update a context menu related value responsive to the event, and provide, to the context menu, a current context menu related value with which a context menu option of the context menu is updated during display of the application page.

205. To the extent the preamble is construed to be limiting, the Accused Products perform a *method* as further explained below. For example, the Accused Products perform a method for plugging context menus into an "extensible app," wherein definitions for context menus and other components are incorporated into "extension points" within that app.

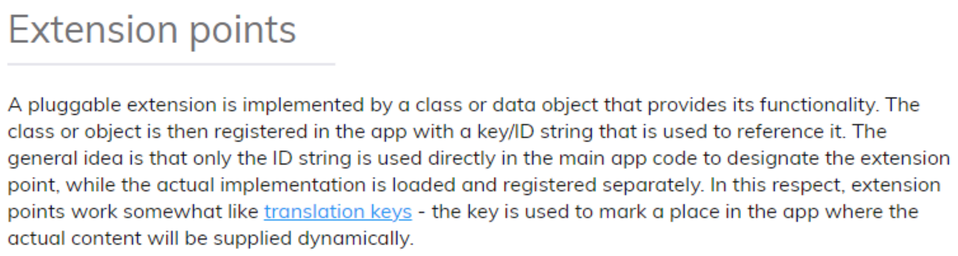


App Extensions

ADF lets you simplify the app developer's task by providing an **extensible app** as a starting point.

An extensible app is designed with *extension points*, which are placeholders where components and other content can be "plugged in" to provide functionality. The app may be supplied with default content for the extension points but the idea is that a developer can easily replace this with custom content as necessary. An organization might find this useful, for example, if they want to create a family of apps with consistent appearance and behavior. One developer can produce an extensible app that can then be adapted by other developers to create the various apps in the family.

(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/)



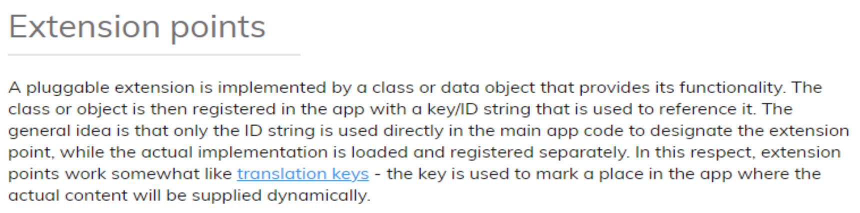
Extension points

A pluggable extension is implemented by a class or data object that provides its functionality. The class or object is then registered in the app with a key/ID string that is used to reference it. The general idea is that only the ID string is used directly in the main app code to designate the extension point, while the actual implementation is loaded and registered separately. In this respect, extension points work somewhat like translation keys - the key is used to mark a place in the app where the actual content will be supplied dynamically.

(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/.)

206. The Accused Products perform a method of *receiving via a software development tool interface a context menu option definition* and *generating, using one or more processors, based at least in part on the context menu option definition, an*

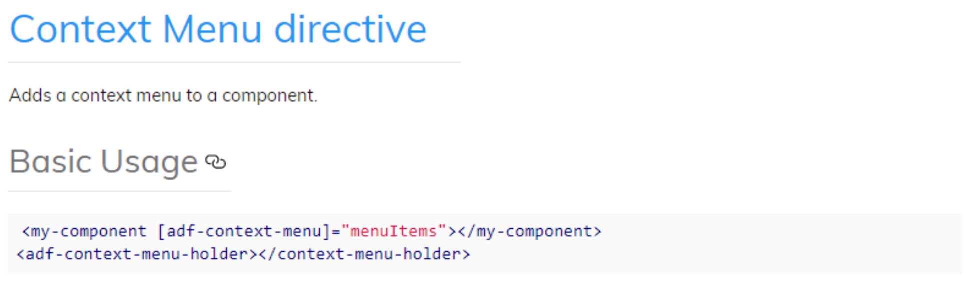
application code to implement the context menu at runtime, including performing processing at runtime. In the example shown below of one possible encoding of a "context menu" within the Accused Products, the "items" or options comprised by that menu and a set of actions associated with those options is shown as being loaded in as a "dynamic component" into an "extensible app." That context menu is plugged into the app at an "extension point" and referred to only by its "key" or "ID string" within the "main app code," "while the actual implementation is loaded and registered separately." An "extension point…is used to mark a place in the app where the actual content will be supplied dynamically." The Accused Products load in the "actual implementation" of that context menu "dynamically" at run time, including the actions to be performed that are associated with that menu's options.



Extension points

A pluggable extension is implemented by a class or data object that provides its functionality. The class or object is then registered in the app with a key/ID string that is used to reference it. The general idea is that only the ID string is used directly in the main app code to designate the extension point, while the actual implementation is loaded and registered separately. In this respect, extension points work somewhat like translation keys - the key is used to mark a place in the app where the actual content will be supplied dynamically.

(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/.)

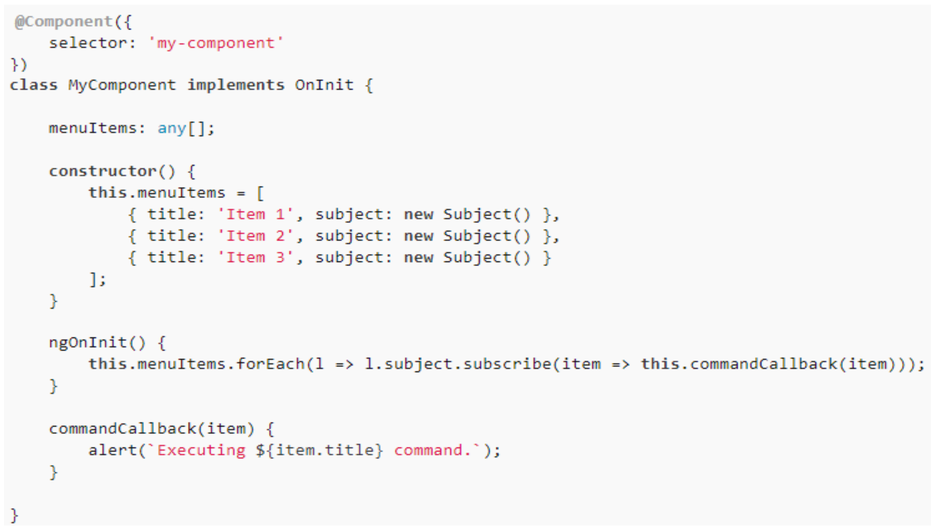


Context Menu directive

Adds a context menu to a component.

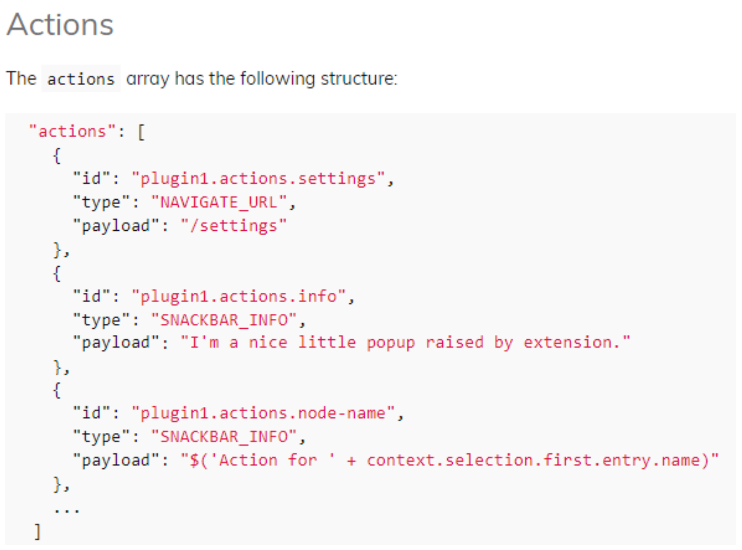
Basic Usage

```
<my-component [adf-context-menu]="menuItems"></my-component>
<adf-context-menu-holder></context-menu-holder>
```



```
@Component({
    selector: 'my-component'
})
class MyComponent implements OnInit {

    menuItems: any[];

    constructor() {
        this.menuItems = [
            { title: 'Item 1', subject: new Subject() },
            { title: 'Item 2', subject: new Subject() },
            { title: 'Item 3', subject: new Subject() }
        ];
    }

    ngOnInit() {
        this.menuItems.forEach(l => l.subject.subscribe(item => this.commandCallback(item)));
    }

    commandCallback(item) {
        alert(`Executing ${item.title} command.`);
    }

}
```

(*See* https://www.alfresco.com/abn/adf/docs/core/directives/context-menu.directive/).

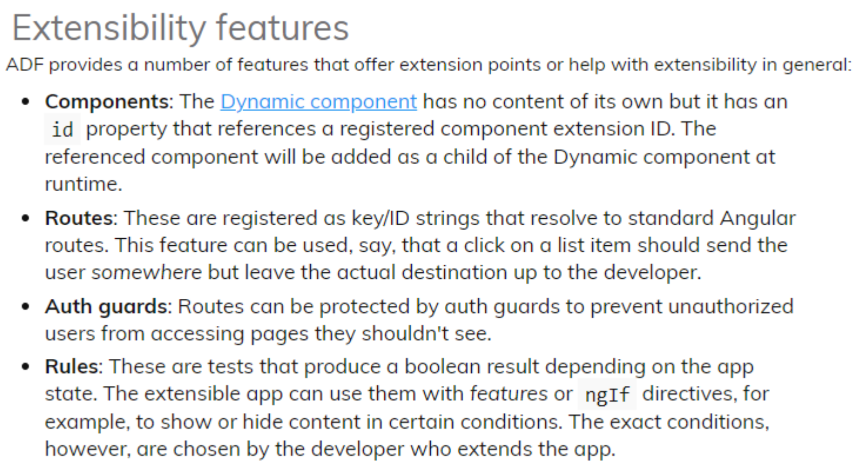


Actions

The `actions` array has the following structure:

```
"actions": [
  {
    "id": "plugin1.actions.settings",
    "type": "NAVIGATE_URL",
    "payload": "/settings"
  },
  {
    "id": "plugin1.actions.info",
    "type": "SNACKBAR_INFO",
    "payload": "I'm a nice little popup raised by extension."
  },
  {
    "id": "plugin1.actions.node-name",
    "type": "SNACKBAR_INFO",
    "payload": "$('Action for ' + context.selection.first.entry.name)"
  },
  ...
]
```

(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/).

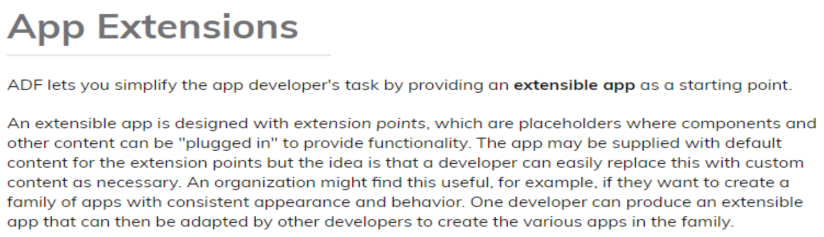


Extensibility features

ADF provides a number of features that offer extension points or help with extensibility in general:

- **Components**: The Dynamic component has no content of its own but it has an `id` property that references a registered component extension ID. The referenced component will be added as a child of the Dynamic component at runtime.
- **Routes**: These are registered as key/ID strings that resolve to standard Angular routes. This feature can be used, say, that a click on a list item should send the user somewhere but leave the actual destination up to the developer.
- **Auth guards**: Routes can be protected by auth guards to prevent unauthorized users from accessing pages they shouldn't see.
- **Rules**: These are tests that produce a boolean result depending on the app state. The extensible app can use them with features or `ngIf` directives, for example, to show or hide content in certain conditions. The exact conditions, however, are chosen by the developer who extends the app.

(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/).



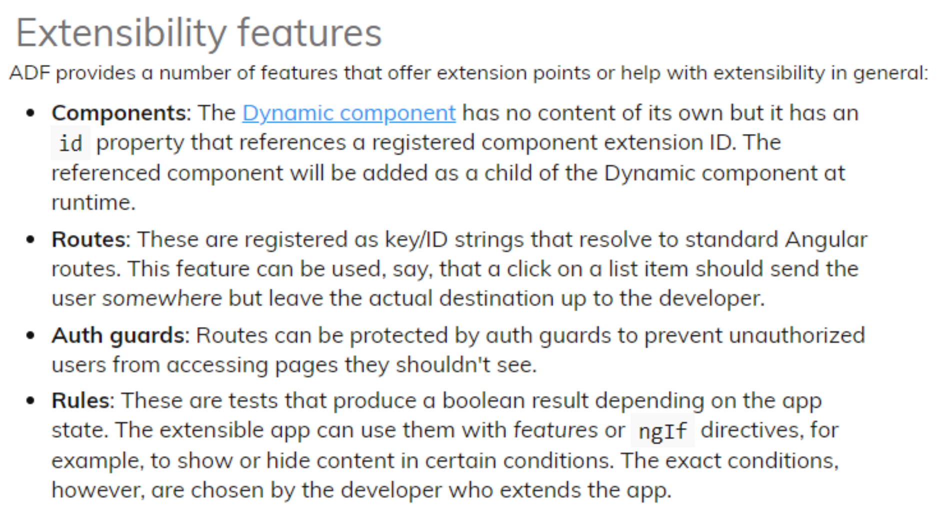
App Extensions

ADF lets you simplify the app developer's task by providing an **extensible app** as a starting point.

An extensible app is designed with extension points, which are placeholders where components and other content can be "plugged in" to provide functionality. The app may be supplied with default content for the extension points but the idea is that a developer can easily replace this with custom content as necessary. An organization might find this useful, for example, if they want to create a family of apps with consistent appearance and behavior. One developer can produce an extensible app that can then be adapted by other developers to create the various apps in the family.

(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/)

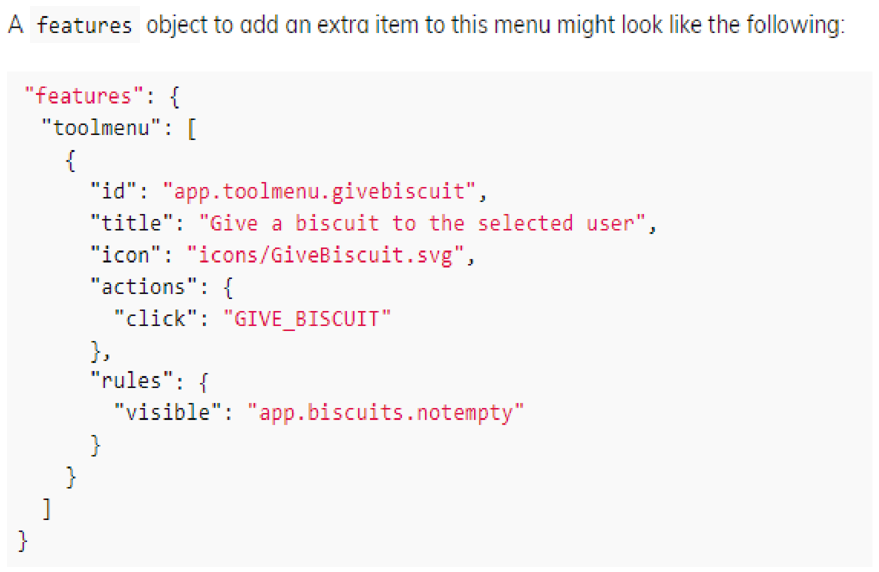
207. The application code generated by the Accused Products is also used *to create an invisible object that is associated with an application page during execution of the application page* and *wherein the invisible object is configured to consume an event during display of the application page, update a context menu related value responsive to the event, and provide, to the context menu, a current context menu related value with which a context menu option of the context menu is updated during display of the application page.* For example, each context menu is loaded into an invisible "Dynamic component" container that is used to add the context menu "as a child of the Dynamic component at runtime." Similarly, an invisible "feature" container can also be used to load in, by referencing its "ID," a particular context menu and to specify certain actions for that menu to perform when interacted with by a user. These containers also specify events that must be satisfied before the context menu is rendered "visible." In the example below, the context menu "tool" is rendered visible if the "biscuits" feature within the app is "not empty."


Extensibility features

ADF provides a number of features that offer extension points or help with extensibility in general:

- **Components**: The Dynamic component has no content of its own but it has an `id` property that references a registered component extension ID. The referenced component will be added as a child of the Dynamic component at runtime.
- **Routes**: These are registered as key/ID strings that resolve to standard Angular routes. This feature can be used, say, that a click on a list item should send the user somewhere but leave the actual destination up to the developer.
- **Auth guards**: Routes can be protected by auth guards to prevent unauthorized users from accessing pages they shouldn't see.
- **Rules**: These are tests that produce a boolean result depending on the app state. The extensible app can use them with features or `ngIf` directives, for example, to show or hide content in certain conditions. The exact conditions, however, are chosen by the developer who extends the app.

(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/).


A `features` object to add an extra item to this menu might look like the following:

```
"features": {
  "toolmenu": [
    {
      "id": "app.toolmenu.givebiscuit",
      "title": "Give a biscuit to the selected user",
      "icon": "icons/GiveBiscuit.svg",
      "actions": {
        "click": "GIVE_BISCUIT"
      },
      "rules": {
        "visible": "app.biscuits.notempty"
      }
    }
  ]
}
```

(*See* https://www.alfresco.com/abn/adf/docs/user-guide/app-extensions/).

208. Each claim in the ’150 Patent recites an independent invention. Neither claim 1, described above, nor any other individual claim is representative of all claims in the ’150 Patent.

209. On information and belief, there has been significant effort by Hyland to imitate OpenText’s patent-protected products to compete with OpenText in the ECM and EIM markets and to increase Hyland’s share of that market at the expense of OpenText’s market share. (*See, e.g.*, Exhibit B (2020.09.09 - Hyland enters definitive

agreement to acquire Alfresco, hyland.com), Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com).) Hyland's efforts have resulted in the Accused Products, which infringe at least claim 1 of the '150 patent as described above, and those efforts would have exposed Hyland to the '150 patent prior to the filing of the original Complaint in this action.

210. Defendant has known of the '150 Patent since receiving a letter identifying the patent and the infringement on September 2, 2022. At the very least, Hyland has been aware of the '150 patent and of its infringement based on the Accused Products since at least the filing and/or service of this Complaint.

211. On information and belief, at least as of the filing of the Complaint in this action, Hyland has knowingly and actively induced and is knowingly and actively inducing at least its customers and partners to directly infringe at least claim 1 of the '150 patent, and has done so with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b), by activities relating to selling, marketing, advertising, promoting, supporting, installing, and distributing its Accused Products in the United States. (Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com).) On information and belief, those activities continue.

212. On information and belief, Hyland deliberately and knowingly encourages, instructs, directs, and/or requires third parties—including its partners, customers, and/or end users—to use the infringing Accused Products in a way that infringes at least claim 1 of the '150 patent as described above.

213. Hyland's partners, customers, and end users of its Accused Products directly infringe at least claim 1 of the '150 patent, at least by using the Accused Products, as described above.

214. For example, on information and belief, Hyland knowingly and intentionally shares instructions, guides, and manuals, including through its website, training programs, and/or YouTube, which advertise and instruct third parties on how to use the Accused Products in a way that directly infringes at least claim 1 of the ’150 patent as described above, including at least Hyland’s customers. On further information and belief, Hyland knowingly and intentionally provides customer service or technical support to purchasers of the infringing Accused Products, which directs and encourages Hyland’s customers to use the Accused Products in a way that directly infringes at least claim 1 of the ’150 patent as described above.

215. On information and belief, the infringing actions of each customer and/or end-user of the Accused Products are attributable to Hyland.

216. On information and belief, Hyland sells and offers for sale the Accused Products and provides technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. On information and belief, each customer enters into a contractual relationship with Hyland, which obligates each customer to perform certain actions as a condition to use the Accused Products. Further, in order to receive the benefit of Hyland’s continued technical support and its specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that infringes the ’150 patent. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Products remain operational for each customer through ongoing technical support, on information and belief, Hyland establishes the manner and timing of each customer’s performance of activities that infringe the ’150 patent.

217. On information and belief, Hyland forms a joint enterprise with its customers to engage in directly infringing the ’150 patent. On further information and belief, Hyland together with each customer operate under a contractual agreement; have

a common purpose to operate the Accused Products in a way that directly infringes the ’150 patent as outlined in the paragraphs above; have pecuniary interests in operating the Accused Products by directly profiting from the sale and/or maintenance of the Accused Products or by indirectly profiting from the increased efficiency resulting from use of the Accused Products; and have equal rights to a voice in the direction of the enterprise either by guiding and advising on the operation and capabilities of the Accused Products with product-specific know-how and expertise or by requesting that certain customer-specific capabilities be implemented in the Accused Products.

218. Hyland also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the inventions claimed, and not a staple article or commodity of commerce suitable for substantial non-infringing uses.

219. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including the development and implementation of context menus and invisible objects and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. These processes are part of the development framework and, on information and belief, cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’150 patent, that functionality could not be performed.

220. Additionally, the accused functionality, including the development and implementation of context menus and invisible objects and related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. For

example, without the context menus and invisible objects, the Accused Products could not provide the dynamic components and features of a user interface. On information and belief, these features cannot be removed or disabled (or, if they could, the system would no longer function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’150 Patent, that functionality could not be performed.

221. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Hyland constitute a material part of the invention—indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionalities (including the functionality for developing and implementing context menus and invisible objects) constitute a material part of the inventions claimed because such functionality is integral to the processes identified above (such as generating "application code to implement the context menu at runtime" and to "create an invisible object that is associated with an application page during execution of the application page") as recited in the claims of the ’150 Patent. None of these products are staple goods—they are sophisticated and customized application development and ECM products, methods, and systems.

222. OpenText "consists of four revenue streams: license, cloud services and subscriptions, customer support, and professional service and other." (Exhibit A at 9-10 (Aug. 6, 2020 10-K).) Each revenue stream relates directly to the ability of OpenText to acquire and retain customers for its software products in a market that is "highly competitive" and increasingly more competitive "as a result of ongoing software industry consolidation," such as Hyland's acquisition of Alfresco. (Exhibit A at 11 (Aug. 6, 2020 10-K); *see also* Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com); Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com); Exhibit F at 4 ("The Forrester

Wave: ECM Content Platforms, Q3 2019"); Exhibit E at 3 (2020.11.16 - Gartner Content Services Report 2020).) OpenText is an innovator in the market and has acquired multiple patents, including the Patents-in-Suit, to give it an advantage over such competition. Hyland's infringing activities have resulted and will continue to result in irreparable harm to OpenText because of the competitive threat that Hyland—including Hyland's acquisition of Alfresco—has to OpenText's share of the relevant "highly competitive" market, and the impact that Hyland's infringing activities have on each one of OpenText's four revenue streams. Further, public interest factors favor OpenText as the owner and assignee of government-issued patents, including the Patents-in-Suit, that serve to recognize OpenText's innovative contribution to the public knowledge in exchange for the patent protection that Hyland is now infringing.

223. For past infringement, OpenText has suffered damages, including lost profits, as a result of Hyland's infringement of the '150 patent. Hyland is therefore liable to OpenText under 35 U.S.C. § 284 for past damages in an amount that adequately compensates OpenText for Hyland's infringement, but no less than a reasonable royalty.

224. OpenText is entitled to a preliminary injunction to maintain the status quo between OpenText and Hyland, which, through its acquisition of Alfresco, is now one of OpenText's biggest competitors (*see, e.g.*, Exhibit A (2020.09.09 - Hyland enters definitive agreement to acquire Alfresco, hyland.com), Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com)), and is using OpenText's patented technology to compete with OpenText in the ECM and EIM markets.

225. For ongoing and future infringement, OpenText will continue to suffer irreparable harm, including without limitation, loss of market share, customers and/or convoyed sales and services which cannot be accurately quantified nor adequately compensated for by money damages, unless this Court preliminarily and permanently

enjoins Hyland, its agents, employees, representatives, and all others acting in concert with Hyland from infringing the ’150 patent.

226. In the alternative, OpenText is entitled to damages in lieu of an injunction, in an amount consistent with the facts, for future infringement. Hyland’s continued infringement, at least since it had notice of the ’150 patent, is knowing and willful. Hyland will be an adjudicated infringer of a valid patent and, thus, Hyland’s future infringement will be willful as a matter of law.

227. Hyland’s infringement is without license or other authorization.

228. This case is exceptional, entitling Plaintiffs to enhanced damages under 35 U.S.C. § 284 and an award of attorneys’ fees and costs incurred in prosecuting this action under 35 U.S.C. § 285.

<u>SIXTH CAUSE OF ACTION</u>

(INFRINGEMENT OF THE ’761 PATENT)

229. Plaintiffs reallege and incorporate the preceding paragraphs of this complaint.

230. Defendants have infringed and continue to infringe one or more claims of the ’761 Patent in violation of 35 U.S.C. § 271 in this District and elsewhere in the United States and will continue to do so unless enjoined by this Court. The Accused Products, including features of the Alfresco Content Services, at least when used for their ordinary and customary purposes, practice each element of at least claim 24 of the ’761 Patent as described below.

231. For example, claim 24 of the ’761 patent recites:

> 24. A method for providing an action flow definition, including:
>
> providing, to a client device, an action flow definition which includes a first association between a user interface page and a first state during which the user interface page is displayed and a second association between a business service associated with a content management server

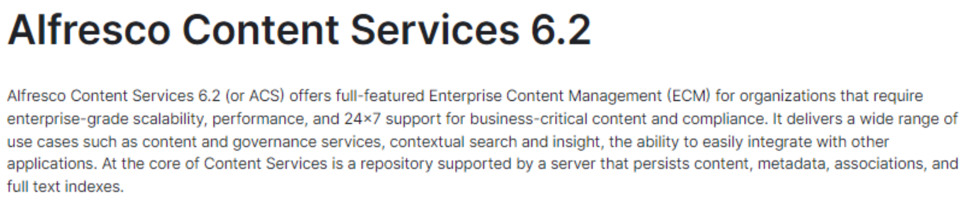
and a second state during which the business service is performed on the content management server, wherein:

the action flow definition is agnostic with respect to user interface technology, on the client device, associated with displaying; and

the client device is configured to perform the action flow definition, including by (1) displaying the user interface page during the first state based on the action flow definition provided to the client device by the front-end server and (2) triggering the business service to be performed on the content management server during the second state based on the action flow definition provided to the client device by the front-end server; and

performing, in response to the trigger from the client, the business service on the content management server.

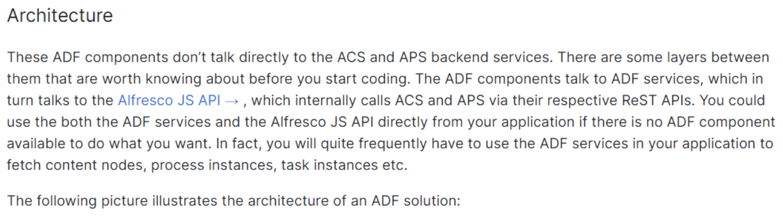
232. The Accused Products perform the method of claim 24 of the '761 Patent. To the extent the preamble is construed to be limiting, the Accused Products perform *a method for providing an action flow definition*, as further explained below. For example, the "Alfresco Content Services 6.2 (or ACS) offers full-featured Enterprise Content Management (ECM) for organizations" and "delivers a wide range of use cases such as content and governance services, contextual search and insight, the ability to easily integrate with other applications." (*See* https://docs.alfresco.com/content-services/6.2/develop/repo-ext-points/repo-actions/; https://docs.alfresco.com/content-services/6.2/develop/share-ext-points/form-processors/.)



Alfresco Content Services 6.2

Alfresco Content Services 6.2 (or ACS) offers full-featured Enterprise Content Management (ECM) for organizations that require enterprise-grade scalability, performance, and 24×7 support for business-critical content and compliance. It delivers a wide range of use cases such as content and governance services, contextual search and insight, the ability to easily integrate with other applications. At the core of Content Services is a repository supported by a server that persists content, metadata, associations, and full text indexes.

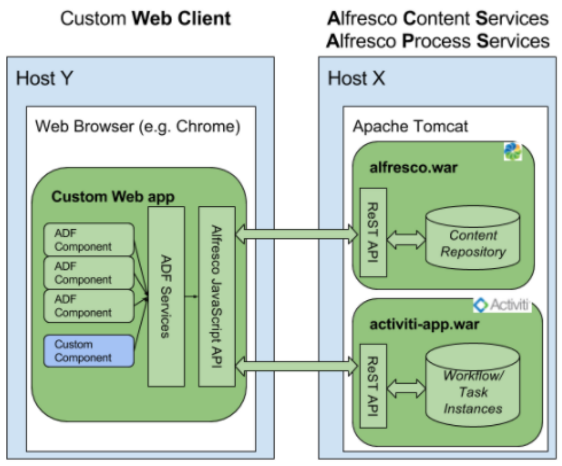
(*See* https://docs.alfresco.com/content-services/6.2/)



Architecture

These ADF components don't talk directly to the ACS and APS backend services. There are some layers between them that are worth knowing about before you start coding. The ADF components talk to ADF services, which in turn talks to the Alfresco JS API →, which internally calls ACS and APS via their respective ReST APIs. You could use the both the ADF services and the Alfresco JS API directly from your application if there is no ADF component available to do what you want. In fact, you will quite frequently have to use the ADF services in your application to fetch content nodes, process instances, task instances etc.

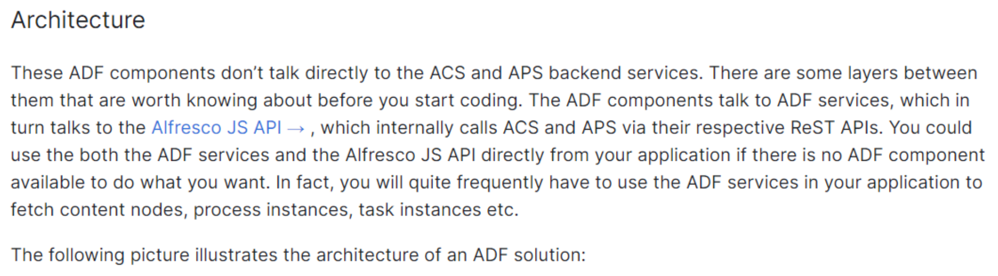
The following picture illustrates the architecture of an ADF solution:



(*See* https://docs.alfresco.com/content-services/6.2/develop/software-architecture/)

233. The Accused Products perform a method that further includes *providing, to a client device, an action flow definition which includes a first association between a user interface page and a first state during which the user interface page is displayed and a second association between a business service associated with a content management server and a second state during which the business service is performed on the content management server*. For example, Alfresco Content Services comprises an Apache Tomcat application server that provides actions to a Custom Web Client. The actions "contain both a back-end part (business logic) and a front-end part (UI widgets)." *See* https://docs.alfresco.com/content-services/6.2/develop/software-

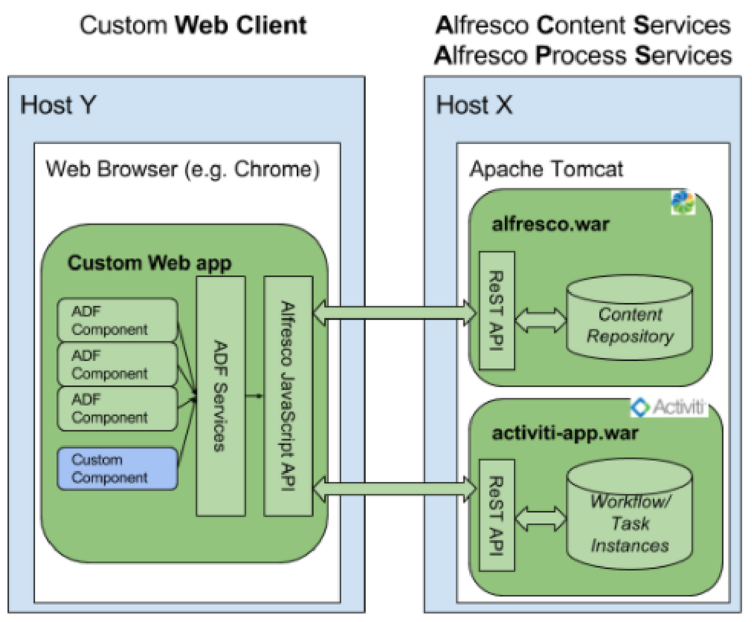
architecture/; ttps://docs.alfresco.com/content-services/6.2/develop/software-architecture/; https://docs.alfresco.com/content-services/6.2/develop/software-architecture/)



Architecture

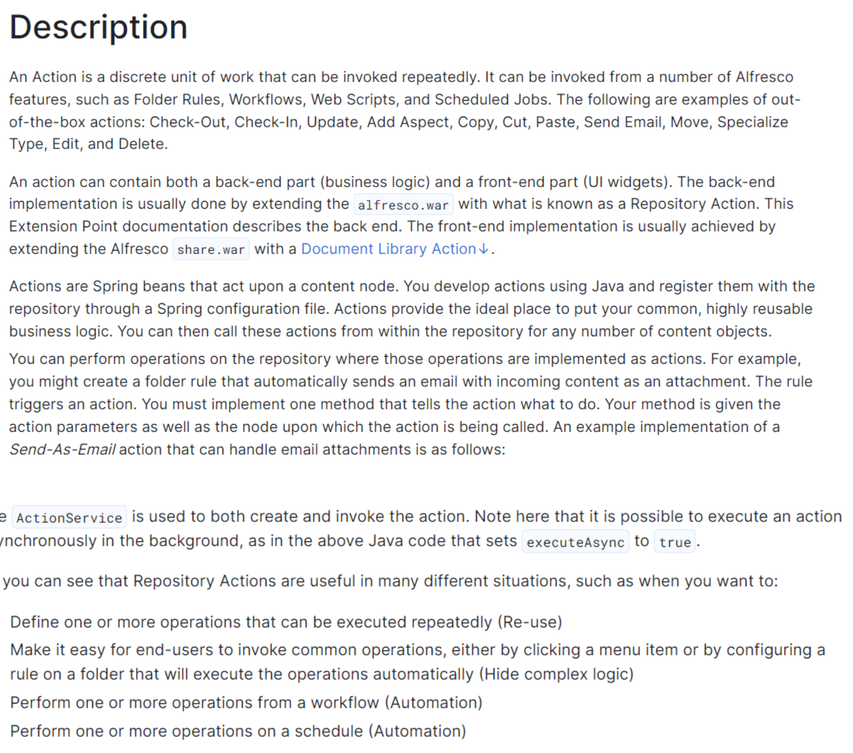
These ADF components don't talk directly to the ACS and APS backend services. There are some layers between them that are worth knowing about before you start coding. The ADF components talk to ADF services, which in turn talks to the Alfresco JS API →, which internally calls ACS and APS via their respective ReST APIs. You could use the both the ADF services and the Alfresco JS API directly from your application if there is no ADF component available to do what you want. In fact, you will quite frequently have to use the ADF services in your application to fetch content nodes, process instances, task instances etc.

The following picture illustrates the architecture of an ADF solution:



(*See* https://docs.alfresco.com/content-services/6.2/develop/software-architecture/)

234. In particular, the Accused Products implement two types of actions: UI action and rule action. The UI action is "called from menu items in the Alfresco Share user interface." The rule action is used to describe "a discrete, reusable unit of work that can be performed against an object in the repository and can optionally be configured at run-time by the user," where the repository stores content files and implements services on a server.



Description

An Action is a discrete unit of work that can be invoked repeatedly. It can be invoked from a number of Alfresco features, such as Folder Rules, Workflows, Web Scripts, and Scheduled Jobs. The following are examples of out-of-the-box actions: Check-Out, Check-In, Update, Add Aspect, Copy, Cut, Paste, Send Email, Move, Specialize Type, Edit, and Delete.

An action can contain both a back-end part (business logic) and a front-end part (UI widgets). The back-end implementation is usually done by extending the `alfresco.war` with what is known as a Repository Action. This Extension Point documentation describes the back end. The front-end implementation is usually achieved by extending the Alfresco `share.war` with a Document Library Action↓.

Actions are Spring beans that act upon a content node. You develop actions using Java and register them with the repository through a Spring configuration file. Actions provide the ideal place to put your common, highly reusable business logic. You can then call these actions from within the repository for any number of content objects.

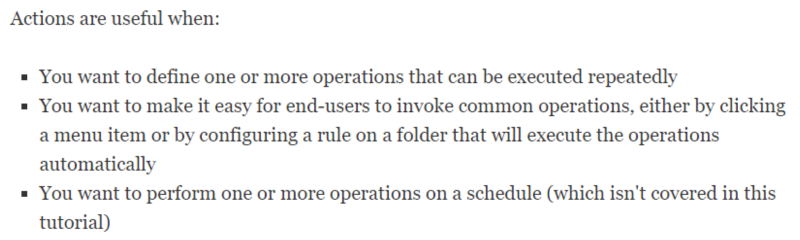
You can perform operations on the repository where those operations are implemented as actions. For example, you might create a folder rule that automatically sends an email with incoming content as an attachment. The rule triggers an action. You must implement one method that tells the action what to do. Your method is given the action parameters as well as the node upon which the action is being called. An example implementation of a *Send-As-Email* action that can handle email attachments is as follows:

The `ActionService` is used to both create and invoke the action. Note here that it is possible to execute an action asynchronously in the background, as in the above Java code that sets `executeAsync` to `true`.

So you can see that Repository Actions are useful in many different situations, such as when you want to:

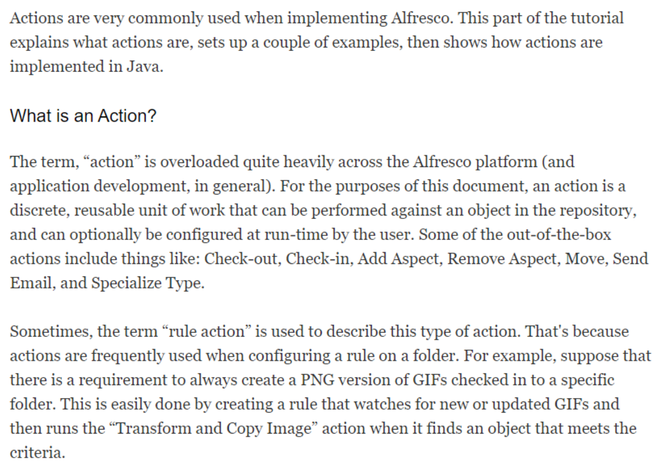
- Define one or more operations that can be executed repeatedly (Re-use)
- Make it easy for end-users to invoke common operations, either by clicking a menu item or by configuring a rule on a folder that will execute the operations automatically (Hide complex logic)
- Perform one or more operations from a workflow (Automation)
- Perform one or more operations on a schedule (Automation)

(*See* https://docs.alfresco.com/content-services/6.2/develop/repo-ext-points/repo-actions/)



Actions are useful when:

- You want to define one or more operations that can be executed repeatedly
- You want to make it easy for end-users to invoke common operations, either by clicking a menu item or by configuring a rule on a folder that will execute the operations automatically
- You want to perform one or more operations on a schedule (which isn't covered in this tutorial)

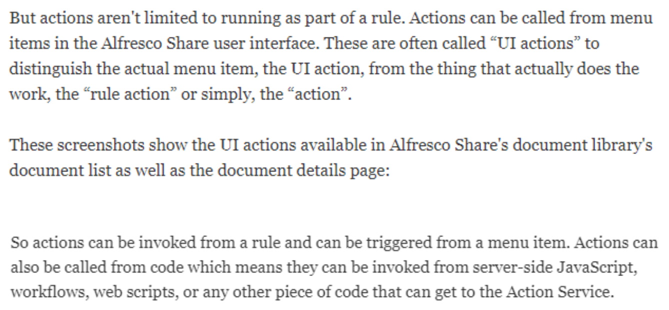


Actions are very commonly used when implementing Alfresco. This part of the tutorial explains what actions are, sets up a couple of examples, then shows how actions are implemented in Java.

What is an Action?

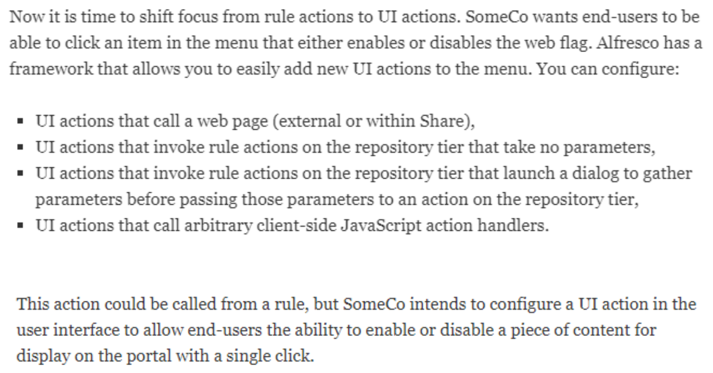
The term, "action" is overloaded quite heavily across the Alfresco platform (and application development, in general). For the purposes of this document, an action is a discrete, reusable unit of work that can be performed against an object in the repository, and can optionally be configured at run-time by the user. Some of the out-of-the-box actions include things like: Check-out, Check-in, Add Aspect, Remove Aspect, Move, Send Email, and Specialize Type.

Sometimes, the term "rule action" is used to describe this type of action. That's because actions are frequently used when configuring a rule on a folder. For example, suppose that there is a requirement to always create a PNG version of GIFs checked in to a specific folder. This is easily done by creating a rule that watches for new or updated GIFs and then runs the "Transform and Copy Image" action when it finds an object that meets the criteria.


But actions aren't limited to running as part of a rule. Actions can be called from menu items in the Alfresco Share user interface. These are often called "UI actions" to distinguish the actual menu item, the UI action, from the thing that actually does the work, the "rule action" or simply, the "action".

These screenshots show the UI actions available in Alfresco Share's document library's document list as well as the document details page:

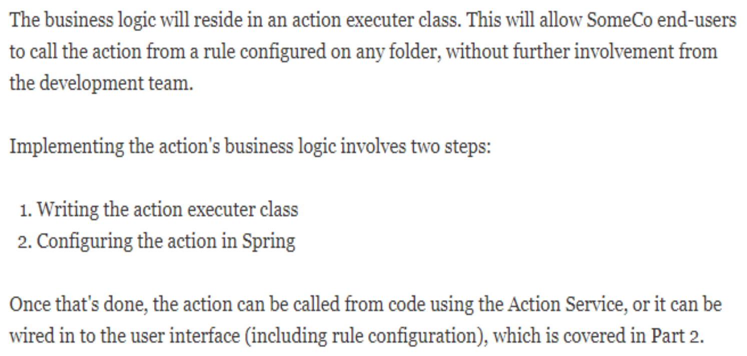
So actions can be invoked from a rule and can be triggered from a menu item. Actions can also be called from code which means they can be invoked from server-side JavaScript, workflows, web scripts, or any other piece of code that can get to the Action Service.


Now it is time to shift focus from rule actions to UI actions. SomeCo wants end-users to be able to click an item in the menu that either enables or disables the web flag. Alfresco has a framework that allows you to easily add new UI actions to the menu. You can configure:

- UI actions that call a web page (external or within Share),
- UI actions that invoke rule actions on the repository tier that take no parameters,
- UI actions that invoke rule actions on the repository tier that launch a dialog to gather parameters before passing those parameters to an action on the repository tier,
- UI actions that call arbitrary client-side JavaScript action handlers.

This action could be called from a rule, but SomeCo intends to configure a UI action in the user interface to allow end-users the ability to enable or disable a piece of content for display on the portal with a single click.

(*See* http://ecmarchitect.com/alfresco-developer-series-tutorials/actions/tutorial/tutorial.html)


The business logic will reside in an action executer class. This will allow SomeCo end-users to call the action from a rule configured on any folder, without further involvement from the development team.

Implementing the action's business logic involves two steps:

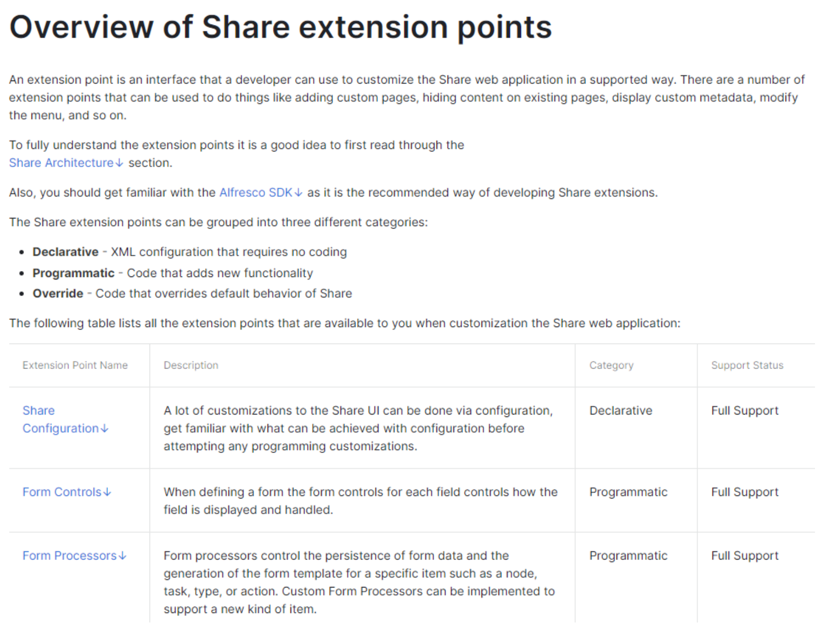
1. Writing the action executer class
2. Configuring the action in Spring

Once that's done, the action can be called from code using the Action Service, or it can be wired in to the user interface (including rule configuration), which is covered in Part 2.

(*See* http://ecmarchitect.com/alfresco-developer-series-tutorials/actions/tutorial/tutorial.html)

235. As another example, Alfresco Content Services includes an extension point named "Form Processors" that is used by a developed to "customize the Share web application in a supported way." The Shared web application generates and displays a form template on Share user interface (UI). When a user inputs and submits form data, a business logic executes in the content repository for persisting the

submitted form data, creating, and updating "an item of a certain kind (for example, node, type, task) based on a form submission."



Overview of Share extension points

An extension point is an interface that a developer can use to customize the Share web application in a supported way. There are a number of extension points that can be used to do things like adding custom pages, hiding content on existing pages, display custom metadata, modify the menu, and so on.

To fully understand the extension points it is a good idea to first read through the Share Architecture↓ section.

Also, you should get familiar with the Alfresco SDK↓ as it is the recommended way of developing Share extensions.

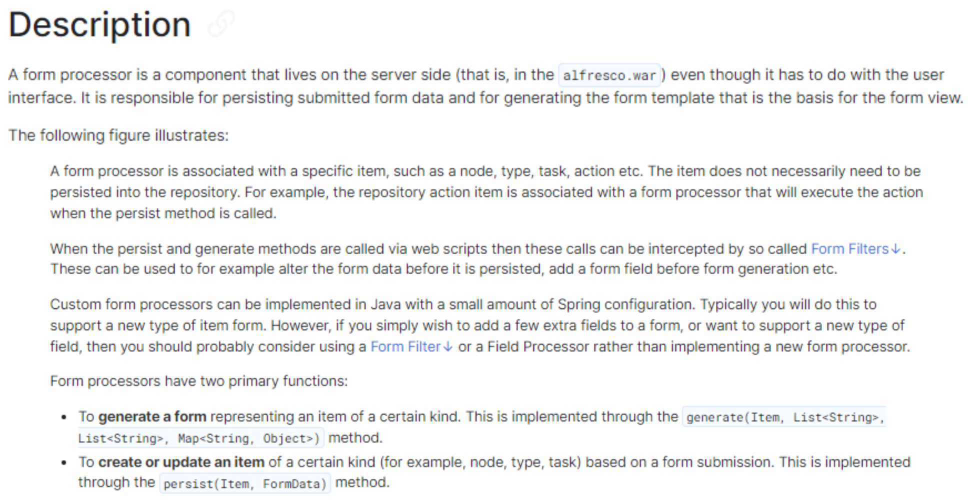
The Share extension points can be grouped into three different categories:

- **Declarative** - XML configuration that requires no coding
- **Programmatic** - Code that adds new functionality
- **Override** - Code that overrides default behavior of Share

The following table lists all the extension points that are available to you when customization the Share web application:

Extension Point Name	Description	Category	Support Status
Share Configuration↓	A lot of customizations to the Share UI can be done via configuration, get familiar with what can be achieved with configuration before attempting any programming customizations.	Declarative	Full Support
Form Controls↓	When defining a form the form controls for each field controls how the field is displayed and handled.	Programmatic	Full Support
Form Processors↓	Form processors control the persistence of form data and the generation of the form template for a specific item such as a node, task, type, or action. Custom Form Processors can be implemented to support a new kind of item.	Programmatic	Full Support

(*See* https://docs.alfresco.com/content-services/6.2/develop/share-ext-points/)



Description

A form processor is a component that lives on the server side (that is, in the `alfresco.war`) even though it has to do with the user interface. It is responsible for persisting submitted form data and for generating the form template that is the basis for the form view.

The following figure illustrates:

A form processor is associated with a specific item, such as a node, type, task, action etc. The item does not necessarily need to be persisted into the repository. For example, the repository action item is associated with a form processor that will execute the action when the persist method is called.

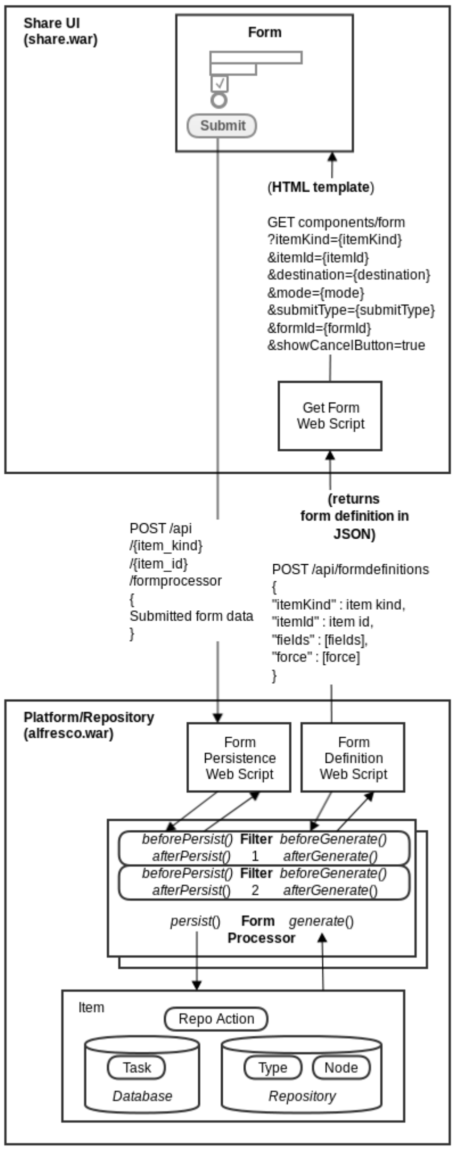
When the persist and generate methods are called via web scripts then these calls can be intercepted by so called Form Filters↓. These can be used to for example alter the form data before it is persisted, add a form field before form generation etc.

Custom form processors can be implemented in Java with a small amount of Spring configuration. Typically you will do this to support a new type of item form. However, if you simply wish to add a few extra fields to a form, or want to support a new type of field, then you should probably consider using a Form Filter↓ or a Field Processor rather than implementing a new form processor.

Form processors have two primary functions:

- To **generate a form** representing an item of a certain kind. This is implemented through the `generate(Item, List<String>, List<String>, Map<String, Object>)` method.
- To **create or update an item** of a certain kind (for example, node, type, task) based on a form submission. This is implemented through the `persist(Item, FormData)` method.

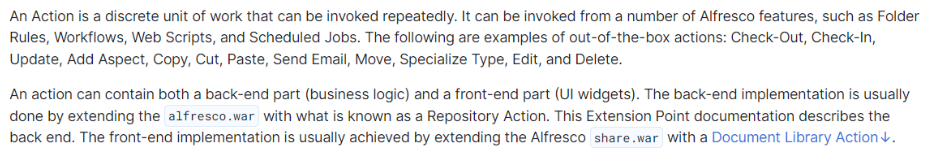
(*See* https://docs.alfresco.com/content-services/6.2/develop/share-ext-points/form-processors/)



(*See* https://docs.alfresco.com/content-services/6.2/develop/share-ext-points/form-processors/)

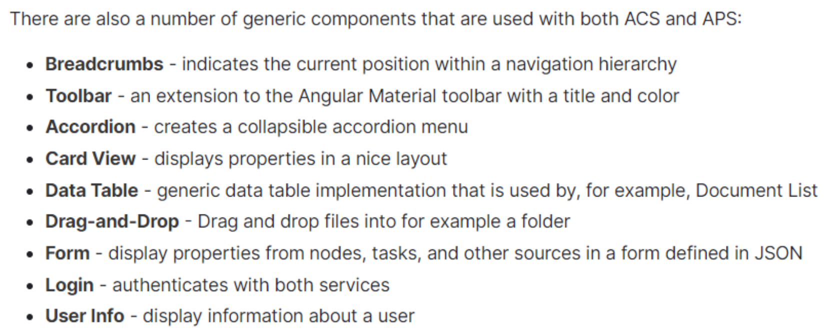
236. The Accused Products perform a method that further includes *the action flow definition is agnostic with respect to user interface technology, on the client device, associated with displaying*. As explained above, in Alfresco Content Services, an "action can contain both a back-end part (business logic) and a front-end part (UI widgets)," where the front-end part (UI widgets). Alfresco Content Services "offers a web-based client called Alfresco Share, built entirely with the web script technology."

"Share gets the content that it should display in pages and dashlets by calling repository web scripts, which returns JSON or XML that can be incorporated into the presentation. The presentation is actually put together with two different kinds of JavaScript frameworks, Yahoo UI library (YUI) and Aikau, which is based on Dojo."

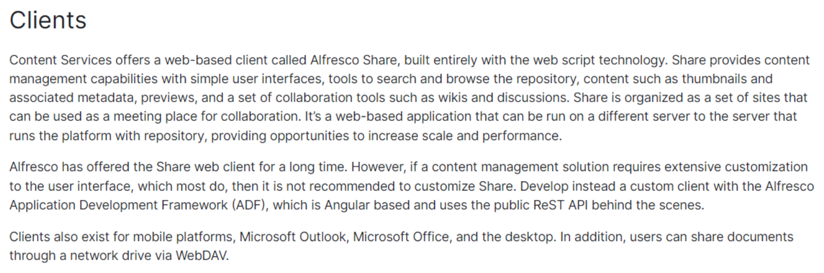

An Action is a discrete unit of work that can be invoked repeatedly. It can be invoked from a number of Alfresco features, such as Folder Rules, Workflows, Web Scripts, and Scheduled Jobs. The following are examples of out-of-the-box actions: Check-Out, Check-In, Update, Add Aspect, Copy, Cut, Paste, Send Email, Move, Specialize Type, Edit, and Delete.

An action can contain both a back-end part (business logic) and a front-end part (UI widgets). The back-end implementation is usually done by extending the `alfresco.war` with what is known as a Repository Action. This Extension Point documentation describes the back end. The front-end implementation is usually achieved by extending the Alfresco `share.war` with a Document Library Action↓.

(*See* https://docs.alfresco.com/content-services/6.2/develop/repo-ext-points/repo-actions/)


There are also a number of generic components that are used with both ACS and APS:

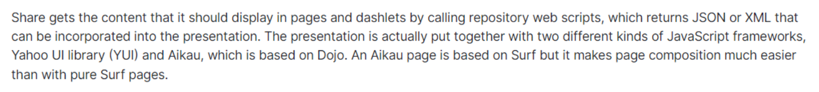
- **Breadcrumbs** - indicates the current position within a navigation hierarchy
- **Toolbar** - an extension to the Angular Material toolbar with a title and color
- **Accordion** - creates a collapsible accordion menu
- **Card View** - displays properties in a nice layout
- **Data Table** - generic data table implementation that is used by, for example, Document List
- **Drag-and-Drop** - Drag and drop files into for example a folder
- **Form** - display properties from nodes, tasks, and other sources in a form defined in JSON
- **Login** - authenticates with both services
- **User Info** - display information about a user

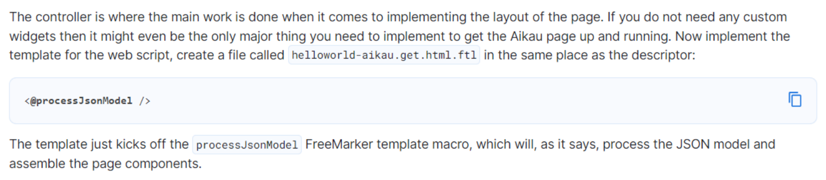

Clients

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 platform with repository, providing opportunities to increase scale and performance.

Alfresco has offered the Share web client for a long time. However, if a content management solution requires extensive customization to the user interface, which most do, then it is not recommended to customize Share. Develop instead a custom client with the Alfresco Application Development Framework (ADF), which is Angular based and uses the public ReST API behind the scenes.

Clients also exist for mobile platforms, Microsoft Outlook, Microsoft Office, and the desktop. In addition, users can share documents through a network drive via WebDAV.


Share gets the content that it should display in pages and dashlets by calling repository web scripts, which returns JSON or XML that can be incorporated into the presentation. The presentation is actually put together with two different kinds of JavaScript frameworks, Yahoo UI library (YUI) and Aikau, which is based on Dojo. An Aikau page is based on Surf but it makes page composition much easier than with pure Surf pages.

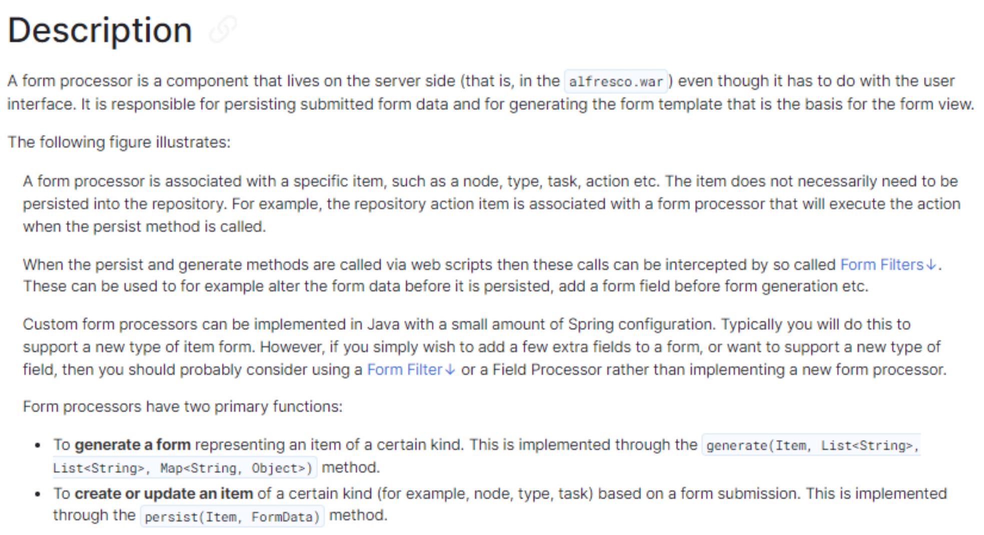

The controller is where the main work is done when it comes to implementing the layout of the page. If you do not need any custom widgets then it might even be the only major thing you need to implement to get the Aikau page up and running. Now implement the template for the web script, create a file called `helloworld-aikau.get.html.ftl` in the same place as the descriptor:

```
<@processJsonModel />
```

The template just kicks off the `processJsonModel` FreeMarker template macro, which will, as it says, process the JSON model and assemble the page components.

(*See* https://docs.alfresco.com/content-services/6.2/develop/software-architecture/#web-ui-architecture)

237. The Accused Products perform a method that further includes *the client device is configured to perform the action flow definition, including by (1) displaying the user interface page during the first state based on the action flow definition provided to the client device by the front-end server and (2) triggering the business service to be performed on the content management server during the second state based on the action flow definition provided to the client device by the front-end server;* and *performing, in response to the trigger from the client, the business service on the content management server*. As explained above, Alfresco Content Services "offers a web-based client called Alfresco Share, built entirely with the web script technology." Alfresco Share generates and displays a form template on Share user interface (UI). When a user inputs and submits form data, a business logic is triggered to execute in the content repository for persisting the submitted form data, creating, and updating "an item of a certain kind (for example, node, type, task) based on a form submission."


Description

A form processor is a component that lives on the server side (that is, in the `alfresco.war`) even though it has to do with the user interface. It is responsible for persisting submitted form data and for generating the form template that is the basis for the form view.

The following figure illustrates:

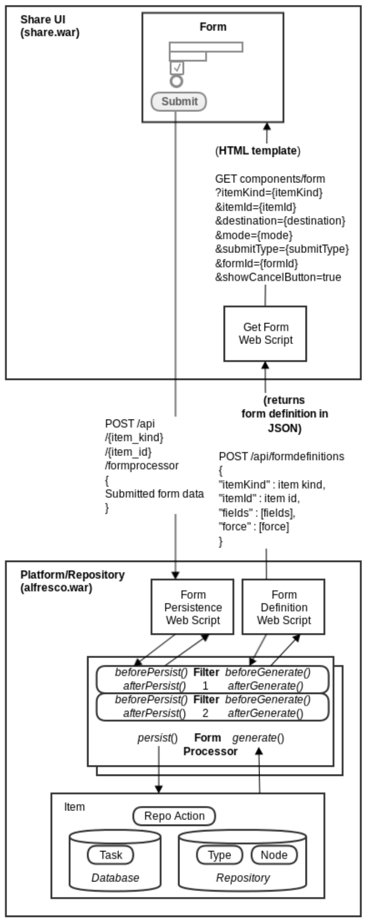
A form processor is associated with a specific item, such as a node, type, task, action etc. The item does not necessarily need to be persisted into the repository. For example, the repository action item is associated with a form processor that will execute the action when the persist method is called.

When the persist and generate methods are called via web scripts then these calls can be intercepted by so called Form Filters↓. These can be used to for example alter the form data before it is persisted, add a form field before form generation etc.

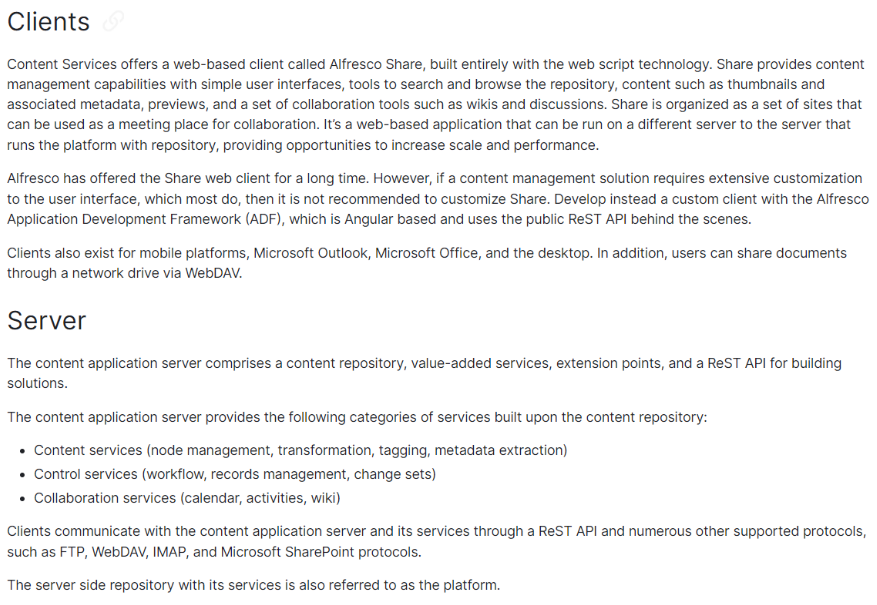
Custom form processors can be implemented in Java with a small amount of Spring configuration. Typically you will do this to support a new type of item form. However, if you simply wish to add a few extra fields to a form, or want to support a new type of field, then you should probably consider using a Form Filter↓ or a Field Processor rather than implementing a new form processor.

Form processors have two primary functions:

- To **generate a form** representing an item of a certain kind. This is implemented through the `generate(Item, List<String>, List<String>, Map<String, Object>)` method.
- To **create or update an item** of a certain kind (for example, node, type, task) based on a form submission. This is implemented through the `persist(Item, FormData)` method.


Share UI (share.war)
Form
Submit
(HTML template)
GET components/form ?itemKind={itemKind} &itemId={itemId} &destination={destination} &mode={mode} &submitType={submitType} &formId={formId} &showCancelButton=true
Get Form Web Script
(returns form definition in JSON)
POST /api /{item_kind} /{item_id} /formprocessor { Submitted form data }
POST /api/formdefinitions { "itemKind" : item kind, "itemId" : item id, "fields" : [fields], "force" : [force] }
Platform/Repository (alfresco.war)
Form Persistence Web Script
Form Definition Web Script
beforePersist() Filter 1 beforeGenerate()
afterPersist() afterGenerate()
beforePersist() Filter 2 beforeGenerate()
afterPersist() afterGenerate()
persist() Form Processor generate()
Item
Repo Action
Task
Database
Type
Node
Repository

(*See* https://docs.alfresco.com/content-services/6.2/develop/share-ext-points/form-processors/)



Clients

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 platform with repository, providing opportunities to increase scale and performance.

Alfresco has offered the Share web client for a long time. However, if a content management solution requires extensive customization to the user interface, which most do, then it is not recommended to customize Share. Develop instead a custom client with the Alfresco Application Development Framework (ADF), which is Angular based and uses the public ReST API behind the scenes.

Clients also exist for mobile platforms, Microsoft Outlook, Microsoft Office, and the desktop. In addition, users can share documents through a network drive via WebDAV.

Server

The content application server comprises a content repository, value-added services, extension points, and a ReST API for building solutions.

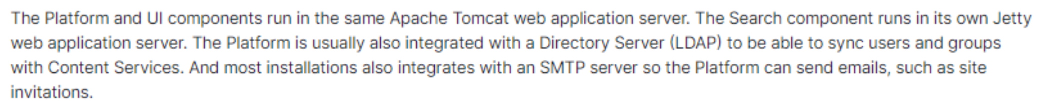
The content application server provides the following categories of services built upon the content repository:

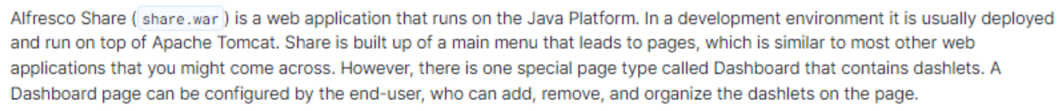
- Content services (node management, transformation, tagging, metadata extraction)
- Control services (workflow, records management, change sets)
- Collaboration services (calendar, activities, wiki)

Clients communicate with the content application server and its services through a ReST API and numerous other supported protocols, such as FTP, WebDAV, IMAP, and Microsoft SharePoint protocols.

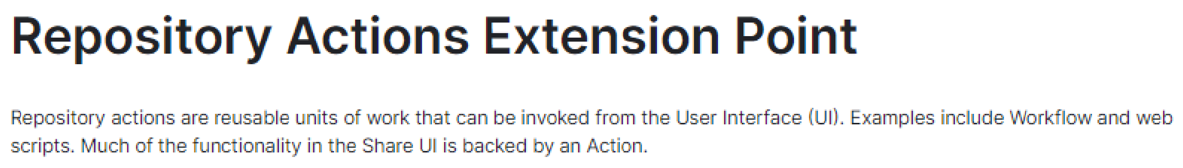
The server side repository with its services is also referred to as the platform.

(*See* https://docs.alfresco.com/content-services/6.2/develop/software-architecture/#web-ui-architecture)


The Platform and UI components run in the same Apache Tomcat web application server. The Search component runs in its own Jetty web application server. The Platform is usually also integrated with a Directory Server (LDAP) to be able to sync users and groups with Content Services. And most installations also integrates with an SMTP server so the Platform can send emails, such as site invitations.

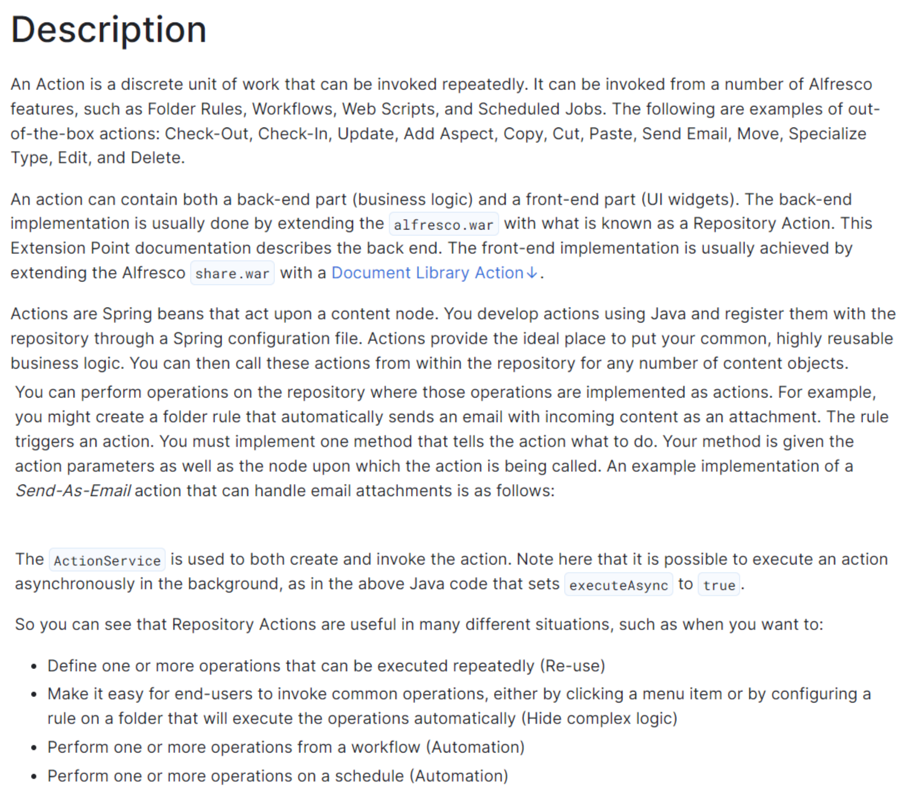

Alfresco Share (`share.war`) is a web application that runs on the Java Platform. In a development environment it is usually deployed and run on top of Apache Tomcat. Share is built up of a main menu that leads to pages, which is similar to most other web applications that you might come across. However, there is one special page type called Dashboard that contains dashlets. A Dashboard page can be configured by the end-user, who can add, remove, and organize the dashlets on the page.

(*See* https://docs.alfresco.com/content-services/6.2/develop/software-architecture/#sharearchitecture)


Repository Actions Extension Point

Repository actions are reusable units of work that can be invoked from the User Interface (UI). Examples include Workflow and web scripts. Much of the functionality in the Share UI is backed by an Action.

(*See* https://docs.alfresco.com/content-services/6.2/develop/repo-ext-points/repo-actions/)


Description

An Action is a discrete unit of work that can be invoked repeatedly. It can be invoked from a number of Alfresco features, such as Folder Rules, Workflows, Web Scripts, and Scheduled Jobs. The following are examples of out-of-the-box actions: Check-Out, Check-In, Update, Add Aspect, Copy, Cut, Paste, Send Email, Move, Specialize Type, Edit, and Delete.

An action can contain both a back-end part (business logic) and a front-end part (UI widgets). The back-end implementation is usually done by extending the `alfresco.war` with what is known as a Repository Action. This Extension Point documentation describes the back end. The front-end implementation is usually achieved by extending the Alfresco `share.war` with a Document Library Action↓.

Actions are Spring beans that act upon a content node. You develop actions using Java and register them with the repository through a Spring configuration file. Actions provide the ideal place to put your common, highly reusable business logic. You can then call these actions from within the repository for any number of content objects.

You can perform operations on the repository where those operations are implemented as actions. For example, you might create a folder rule that automatically sends an email with incoming content as an attachment. The rule triggers an action. You must implement one method that tells the action what to do. Your method is given the action parameters as well as the node upon which the action is being called. An example implementation of a *Send-As-Email* action that can handle email attachments is as follows:

The `ActionService` is used to both create and invoke the action. Note here that it is possible to execute an action asynchronously in the background, as in the above Java code that sets `executeAsync` to `true`.

So you can see that Repository Actions are useful in many different situations, such as when you want to:

- Define one or more operations that can be executed repeatedly (Re-use)
- Make it easy for end-users to invoke common operations, either by clicking a menu item or by configuring a rule on a folder that will execute the operations automatically (Hide complex logic)
- Perform one or more operations from a workflow (Automation)
- Perform one or more operations on a schedule (Automation)

(*See* https://docs.alfresco.com/content-services/6.2/develop/repo-ext-points/repo-actions/)

238. Hyland has known of the ’761 Patent since receiving a letter identifying the patent and the infringement on September 2, 2022. At the very least, Hyland has been aware of the ’761 patent and of its infringement based on the Accused Products since at least the filing and/or service of this Complaint. Further, OpenText marks its products with the ’761 patent.

239. On information and belief, at least as of the filing of the Complaint in this action, Hyland has knowingly and actively induced and is knowingly and actively inducing at least its customers and partners to directly infringe at least claim 1 of the ’761 patent, and has done so with specific intent to induce infringement, and/or willful blindness to the possibility that its acts induce infringement, in violation of 35 U.S.C. § 271(b), by activities relating to selling, marketing, advertising, promoting, supporting, installing, and distributing its Accused Products in the United States. (Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com).) On information and belief, those activities continue.

240. On information and belief, Hyland deliberately and knowingly encourages, instructs, directs, and/or requires third parties—including its partners, customers, and/or end users—to use the Accused Products in a way that infringes at least claim 1 of the ’726 patent as described above.

241. Hyland’s partners, customers, and end users of its Accused Products directly infringe at least claim 1 of the ’761 patent, at least by using the Accused Products, as described above.

242. For example, on information and belief, Hyland knowingly and intentionally shares instructions, guides, and manuals, including through its website, training programs, and/or YouTube, which advertise and instruct third parties on how to use the Accused Products in a way that directly infringes at least claim 1 of the ’761

patent as described above, including at least Hyland's customers. On further information and belief, Hyland knowingly and intentionally provides customer service or technical support to purchasers of the infringing Accused Products, which directs and encourages Hyland's customers to use the Accused Products in a way that directly infringes at least claim 1 of the '761 patent as described above.

243. On information and belief, the infringing actions of each customer and/or end-user of the Accused Products are attributable to Hyland.

244. On information and belief, Hyland sells and offers for sale the Accused Products and provides technical support for the installation, implementation, integration, and ongoing operation of the Accused Products for each individual customer. On information and belief, each customer enters into a contractual relationship with Hyland, which obligates each customer to perform certain actions as a condition to use the Accused Products. Further, in order to receive the benefit of Hyland's continued technical support and its specialized knowledge and guidance of the operability of the Accused Products, each customer must continue to use the Accused Products in a way that infringes the '761 patent. Further, as the entity that provides installation, implementation, and integration of the Accused Products in addition to ensuring the Accused Products remain operational for each customer through ongoing technical support, on information and belief, Hyland establishes the manner and timing of each customer's performance of activities that infringe the '761 patent.

245. On information and belief, Hyland forms a joint enterprise with its customers to engage in directly infringing the '761 patent. On further information and belief, Hyland together with each customer operate under a contractual agreement; have a common purpose to operate the Accused Products in a way that directly infringes the '761 patent as outlined in the paragraphs above; have pecuniary interests in operating the Accused Products by directly profiting from the sale and/or maintenance of the Accused Products or by indirectly profiting from the increased efficiency resulting from

use of the Accused Products; and have equal rights to a voice in the direction of the enterprise either by guiding and advising on the operation and capabilities of the Accused Products with product-specific know-how and expertise or by requesting that certain customer-specific capabilities be implemented in the Accused Products.

246. Hyland also contributes to the infringement of its partners, customers, and end-users of the Accused Products by providing within the United States or importing the Accused Products into the United States, which are for use in practicing, and under normal operation practice, methods claimed in the Asserted Patents, constituting a material part of the inventions claimed, and not a staple article or commodity of commerce suitable for substantial non-infringing uses.

247. Indeed, as shown above, the Accused Products have no substantial non-infringing uses because the accused functionality, including providing an action flow definition and related functionality described above, is an integral part of the Accused Products and must be performed for the Accused Products to perform their intended purpose. On information and belief, these processes cannot be removed or disabled (or, if they could, the system would no longer suitably function for its intended purpose). Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’761 patent, that functionality could not be performed.

248. Additionally, the accused functionality, including the implementation of action flow definitions and related functionality described above, itself has no substantial non-infringing uses because the components, modules and methods identified above are a necessary part of that functionality. Moreover, for the same reasons, without performing each of the steps as described and shown above, or without the system and components identified above that practice the ’761 Patent, that functionality could not be performed.

249. In addition, as shown in the detailed analysis above, the products, systems, modules, and methods provided by Hyland constitute a material part of the invention—

indeed, they provide all the components, modules, and features that perform the claimed methods and systems. For example, the Accused Products and accused functionalities (including the action flow functionality) constitute a material part of the inventions claimed because such functionality is integral to the processes identified above (such as "providing, to a client device, an action flow definition which includes a first association between a user interface page and a first state during which the user interface page is displayed and a second association between a business service associated with a content management server and a second state during which the business service is performed on the content management server") as recited in the claims of the '761 Patent. None of these products are staple goods—they are sophisticated and customized ECM products, methods, and systems.

250. OpenText "consists of four revenue streams: license, cloud services and subscriptions, customer support, and professional service and other." (Exhibit A at 9-10 (Aug. 6, 2020 10-K).) Each revenue stream relates directly to the ability of OpenText to acquire and retain customers for its software products in a market that is "highly competitive" and increasingly more competitive "as a result of ongoing software industry consolidation," such as Hyland's acquisition of Alfresco. (Exhibit A at 11 (Aug. 6, 2020 10-K); *see also* Exhibit C_ (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com); Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com); Exhibit F at 4 ("The Forrester Wave: ECM Content Platforms, Q3 2019"); Exhibit E at 3 (2020.11.16 - Gartner Content Services Report 2020).) OpenText is an innovator in the market and has acquired multiple patents, including the Patents-in-Suit, to give it an advantage over such competition. Hyland's infringing activities have resulted and will continue to result in irreparable harm to OpenText because of the competitive threat that Hyland—including Hyland's acquisition of Alfresco—has to OpenText's share of the relevant "highly competitive" market, and the impact that Hyland's infringing activities have on each one of OpenText's four revenue streams. Further, public interest factors favor

OpenText as the owner and assignee of government-issued patents, including the Patents-in-Suit, that serve to recognize OpenText's innovative contribution to the public knowledge in exchange for the patent protection that Hyland is now infringing.

251. For past infringement, OpenText has suffered damages, including lost profits, as a result of Hyland's infringement of the '761 patent. Hyland is therefore liable to OpenText under 35 U.S.C. § 284 for past damages in an amount that adequately compensates OpenText for Hyland's infringement, but no less than a reasonable royalty.

252. OpenText is entitled to a preliminary injunction to maintain the status quo between OpenText and Hyland, which, through its acquisition of Alfresco, is now one of OpenText's biggest competitors (*see, e.g.*, Exhibit B (2020.09.09 - Hyland enters definitive agreement to acquire Alfresco, hyland.com), Exhibit C (2020.10.22 - Hyland completes acquisition of Alfresco, alfresco.com), Exhibit D (2020.12.02 - Hyland and Alfresco named Leaders in Content Services GMQ, hyland.com)), and is using OpenText's patented technology to compete with OpenText in the ECM and EIM markets.

253. For ongoing and future infringement, OpenText will continue to suffer irreparable harm, including without limitation, loss of market share, customers and/or convoyed sales and services which cannot be accurately quantified nor adequately compensated for by money damages, unless this Court preliminarily and permanently enjoins Hyland, its agents, employees, representatives, and all others acting in concert with Hyland from infringing the '761 patent.

254. In the alternative, OpenText is entitled to damages in lieu of an injunction, in an amount consistent with the facts, for future infringement. Hyland's continued infringement, at least since it had notice of the '761 patent, is knowing and willful. Hyland will be an adjudicated infringer of a valid patent and, thus, Hyland's future infringement will be willful as a matter of law.

255. Hyland's infringement is without license or other authorization.

256. This case is exceptional, entitling Plaintiffs to enhanced damages under 35 U.S.C. § 284 and an award of attorneys' fees and costs incurred in prosecuting this action under 35 U.S.C. § 285.

PRAYER FOR RELIEF

WHEREFORE, Plaintiffs respectfully requests the following relief:

a) That this Court adjudge and decree that Defendant has been, and is currently, infringing each of the Patents-in-Suit;

b) That this Court award damages to Plaintiffs to compensate them for Defendant's past infringement of the Patents-in-Suit, through the date of trial in this action;

c) That this Court award pre- and post-judgment interest on such damages to Plaintiffs;

d) That this Court order an accounting of damages incurred by Plaintiffs from six years prior to the date this lawsuit was filed through the entry of a final, non-appealable judgment;

e) That this Court determine that this patent infringement case is exceptional pursuant to 35 U.S.C. §§ 284 and 285 and award Plaintiffs their costs and attorneys' fees incurred in this action;

f) That this Court award increased damages under 35 U.S.C. § 284;

g) That this Court preliminarily and permanently enjoin Defendant from infringing any of the Patents-in-Suit;

h) That this Court order Defendant to:

(i) recall and collect from all persons and entities that have purchased any and all products found to infringe any of the Patents-in-Suit that were made, offered for sale, sold, or otherwise distributed in the United States by Defendant or anyone acting on its behalf;

(ii) destroy or deliver to OpenText all such infringing products;

(iii) revoke all licenses to all such infringing products;

(iv) disable all web pages offering or advertising all such infringing products;

(v) destroy all other marketing materials relating to all such infringing products;

(vi) disable all applications providing access to all such infringing software; and

(vii) destroy all infringing software that exists on hosted systems;

i) That this Court, if it declines to enjoin Defendant from infringing any of the Patents-in-Suit, award damages for future infringement in lieu of an injunction; and

j) That this award such other relief as the Court deems just and proper.

DATED: September 2, 2022 KING & SPALDING LLP

By: */s/Joseph N. Akrotirianakis*
JOSEPH N. AKROTIRIANAKIS

Attorney for Plaintiffs OPEN TEXT CORPORATION, OPEN TEXT SA ULC, and OPEN TEXT HOLDINGS INC.

DEMAND FOR JURY TRIAL

OpenText respectfully requests a trial by jury on all issues triable thereby.

DATED: September 2, 2022 KING & SPALDING LLP

By: */s/ Joseph N. Akrotirianakis*
JOSEPH N. AKROTIRIANAKIS

Attorney for Plaintiffs OPEN TEXT CORPORATION, OPEN TEXT SA ULC, and OPEN TEXT HOLDINGS INC.