AtlasMap Developer Guide

The AtlasMap Team

Version 1.42.0-SNAPSHOT, 2019-08-07

AtlasMap Developer Guide

Chapter 1. Introduction

This document provides some information for the developers who is willing to contribute to AtlasMap code.

Chapter 2. Quickstart

2.1. Running AtlasMap Data Mapper UI within Syndesis

Data Mapper is primarily designed to be embeded and run in Syndesis as a Data Mapper step. Simply follow the Syndesis Developer Handbook to install, and run Syndesis UI. You will find the Data Mapper UI under the integrations panel after selecting or adding an integration with a data mapping step involved in the integration.

2.2. Running AtlasMap Data Mapper UI standalone

Here is the shortest path to run standalone AtlasMap.

1. Download AtlasMap standalone jar

```
wget http://central.maven.org/maven2/io/atlasmap/atlasmap-
standalone/${VERSION}/atlasmap-standalone-${VERSION}.jar
```

2. Run AtlasMap standalone

```
$ java -jar atlasmap-standalone-${VERSION}.jar
```

Now AtlasMap Data Mapper UI is available at http://127.0.0.1:8585/

2.3. Running AtlasMap build

Building everything for standalone usage

1. Clone AtlasMap repository

```
$ git clone https://github.com/atlasmap/atlasmap ${ATLASMAP}
```

2. Build AtlasMap runtime

```
$ cd ${ATLASMAP}
$ ./mvnw clean install
```

2.4. Tips for UI developer

2.4.1. Developing Within Syndesis UI

Data Mapper UI is referenced by Syndesis as a dependency. When Syndesis UI's dependencies are installed during **yarn install** step, Data Mapper UI will be cloned from the NPM package repository into **${SYNDESIS}/app/ui/node_modules/@atlasmap/atlasmap-data-mapper** directory.

You can point your local Syndesis UI's Data Mapper UI reference to your working copy of the Data Mapper by yarn link. You'll do something like this:

```
$ cd ${ATLASMAP}/ui
$ yarn build:lib
$ cd dist/lib
$ yarn link
$ cd ${SYNDESIS}/app/ui
$ yarn link @atlasmap/atlasmap-data-mapper
```

Note that running `yarn install` in the Syndesis UI directory **will remove and redownload the ${SYNDESIS}/app/ui/node_modules/@atlasmap/atlasmap-data-mapper directory**, so you'd want to avoid doing it while you have `yarn link`. Instead you'd change ${ATLASMAP}/ui/src and run `yarn build:lib` to make a change in AtlasMap UI and consume it within Syndesis UI.

2.4.2. Debug unit tests with Chrome DevTools

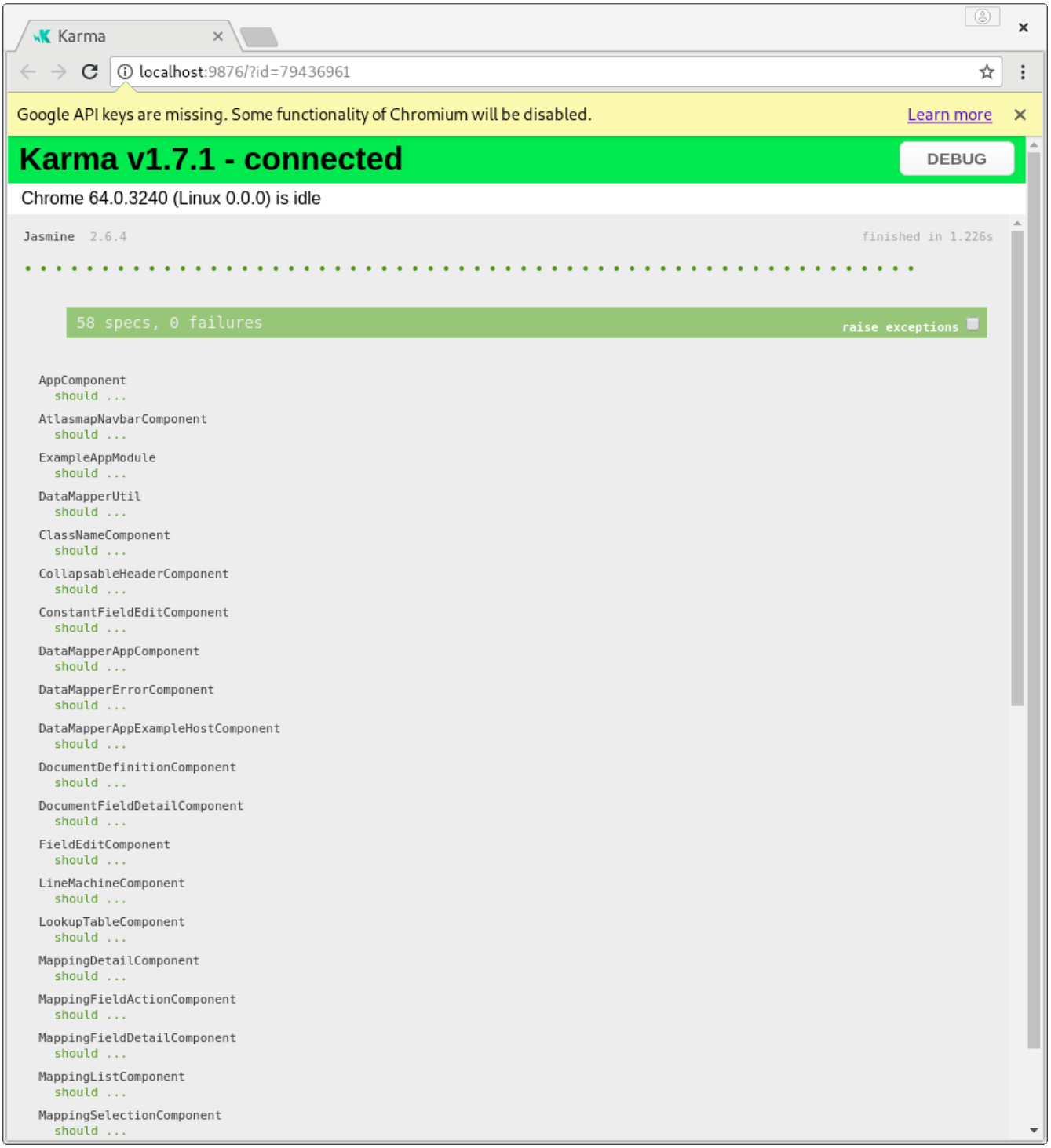
To debug AtlasMap Data Mapper UI unit tests with using Chrome DevTools:

```
$ cd ${ATLASMAP}/ui
$ yarn test:debug
```

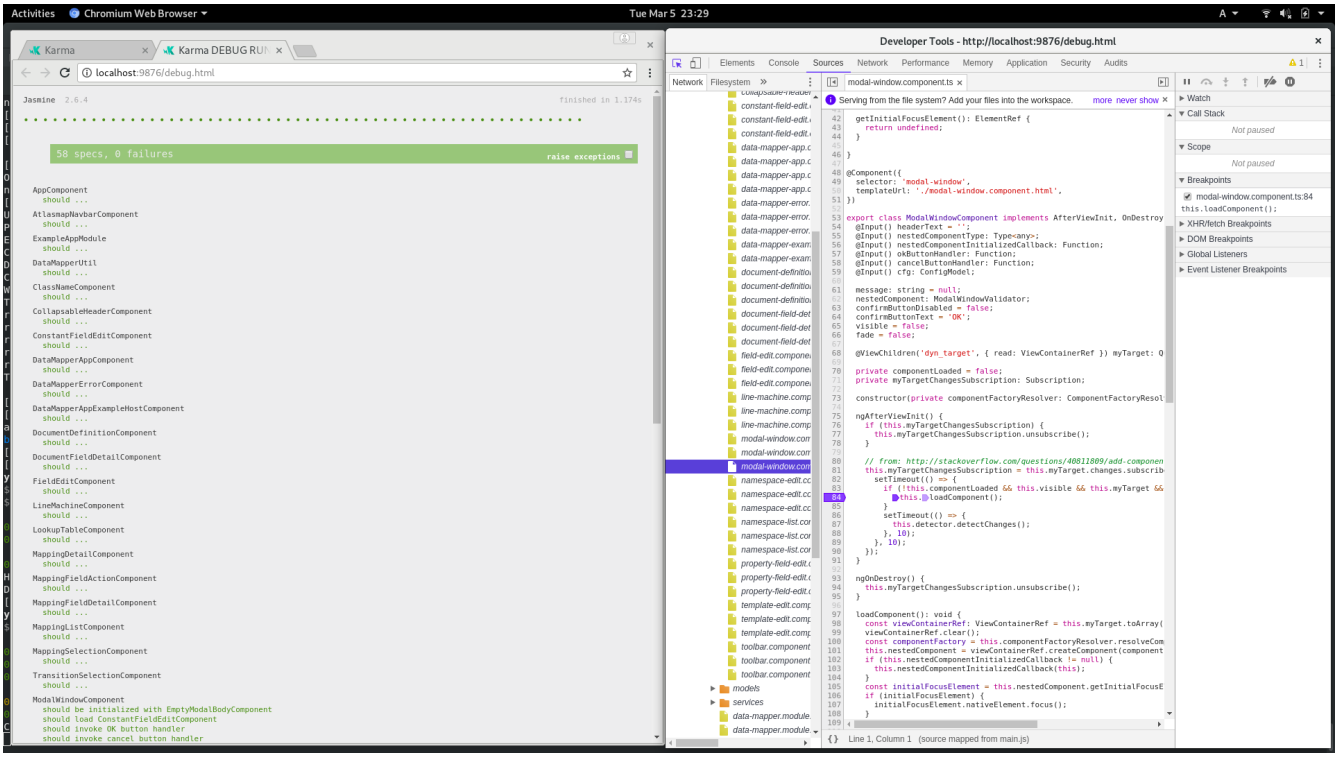
Or you can specify target test file alone

```
$ yarn test:debug --main src/app/lib/atlasmap-data-mapper/services/mapping-
serializer.service.spec.ts
```

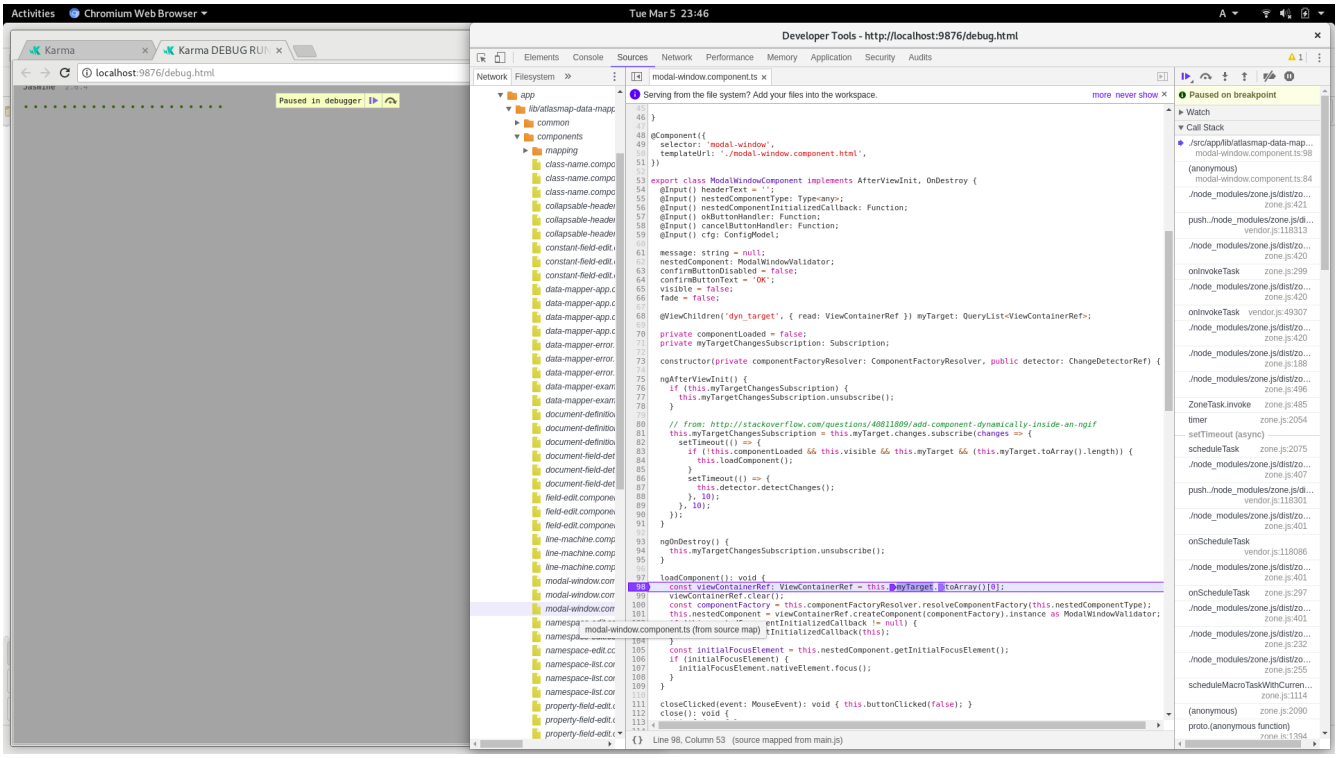
Then Chromium window opens up. Push the `DEBUG` button. New DEBUG tab will open.



Then open Chrome DevTools, open TypeScript source file and put a breakpoint.



Once you reload DEBUG tab, it stops at the breakpoint and you can start step debugging.



2.4.3. Debug Configuration

The Data Mapper UI features several developer-friendly debug configuration options. These configuration fields are specified on the ConfigModel class.

Service Endpoint Configuration

1. **baseJavaInspectionServiceUrl** - URL for the Java Inspection Service provided by the AtlasMap Services.

2. **baseXMLInspectionServiceUrl** - URL for the XML Inspection Service provided by the AtlasMap Services.
3. **baseJSONInspectionServiceUrl** - URL for the JSON Inspection Service provided by the AtlasMap Services.
4. **baseMappingServiceUrl** - URL for the Mapping Service provided by the AtlasMap Services.

Mock Source/Target Document Configuration

These flags control the UI automatically adding the specified mock documents to the system when the UI initializes.

1. **addMockJavaSingleSource** - Add single Java source document.
2. **addMockJavaSources** - Add multiple Java source documents.
3. **addMockXMLInstanceSources** - Add multiple XML instance-based source documents.
4. **addMockXMLSchemaSources** - Add multiple XML schema-based source documents.
5. **addMockJSONSources** - Add multiple JSON source documents.
6. **addMockJavaTarget** - Add a Java target document.
7. **addMockXMLInstanceTarget** - Add a XML instance target document.
8. **addMockXMLSchemaTarget** - Add a XML schema target document.
9. **addMockJSONTarget** - Add a JSON target document.

The code that initializes these mock documents is in the InitializationService. That service calls various static methods in the DocumentManagementService to create example XML instance, XML schema, and JSON documents. Mock Java documents referenced are from the AtlasMap Services' Atlas Java Test Model Maven Module.

Additional Debug Configuration

1. **discardNonMockSources** - Automatically discard all user-specified (or Syndesis UI-specified) source/target documents before initializing. This is helpful if you're trying to test with mock documents alone.
2. **addMockJSONMappings** - This flag bootstraps the UI's mappings from the provided JSON mapping definition. Useful for repeatedly debugging a particular scenario.
3. **debugClassPathServiceCalls** - Log details about JSON request/responses to/from the class path resolution service.
4. **debugDocumentServiceCalls** - Log details about JSON request/responses to/from the Java/XML/JSON inspection services.
5. **debugMappingServiceCalls** - Log details about JSON request/responses to/from the mapping service.
6. **debugValidationServiceCalls** - Log details about JSON request/responses to/from the mapping validation service.
7. **debugFieldActionServiceCalls** - Log details about JSON request/responses to/from the mapping field action configuration service.

8. **debugDocumentParsing** - Log details about parsing JSON responses from the inspection services.

Data Mapper Debug Configuration within the Syndesis UI is defined within your **${SYNDESIS}/ui/src/config.json** file's data mapper section:

```
{
  "apiEndpoint": "https://syndesis-staging.b6ff.rh-idev.openshiftapps.com/api/v1",
  "title": "Syndesis",
  "datamapper": {
    "baseJavaInspectionServiceUrl": "http://localhost:8585/v2/atlas/java/",
    "baseXMLInspectionServiceUrl": "http://localhost:8585/v2/atlas/xml/",
    "baseJSONInspectionServiceUrl": "http://localhost:8585/v2/atlas/json/",
    "baseMappingServiceUrl": "http://localhost:8585/v2/atlas/",
    "discardNonMockSources": true,
    "addMockJSONMappings": false,
    "addMockJavaSingleSource": true,
    "addMockJavaSources": false,
    "addMockXMLInstanceSources": true,
    "addMockXMLSchemaSources": true,
    "addMockJSONSources": true,
    "addMockJavaTarget": false,
    "addMockXMLInstanceTarget": false,
    "addMockXMLSchemaTarget": false,
    "addMockJSONTarget": true,
    "debugDocumentServiceCalls": true,
    "debugMappingServiceCalls": true,
    "debugClassPathServiceCalls": false,
    "debugValidationServiceCalls": false,
    "debugFieldActionServiceCalls": false,
    "debugDocumentParsing": false
  },
  "oauth": {
    "clientId": "syndesis-ui",
    "scopes": ["openid"],
    "oidc": true,
    "hybrid": true,
    "issuer": "https://syndesis-staging.b6ff.rh-
idev.openshiftapps.com/auth/realms/syndesis",
    "auto-link-github": true
  }
}
```

If you're running the Data Mapper UI locally outside of the Syndesis UI, the debug configuration is specified within the DataMapperAppExampleHostComponent.

Chapter 3. Internal Design

AtlasMap consists of following 3 parts:

1. Data Mapper UI
2. Design Time Service
3. Runtime Engine

Data Mapper UI is an Angular based web browser application. Design Time Service is a set of REST API which provide background services to be used by the Data Mapper UI behind the scene. Data Mapper UI eventually produces data mapping definition file in XML or JSON format. Then Runtime Engine consumes that mapping definition as well as actual data payload and perform mapping.

When data formats are provided into Data Mapper UI, it requests an inspection to Design Time Service and receives a unified metadata, called Document. For example, if the data format is provided as a JSON schema, Data Mapper UI makes a JSON schema inspection request and receive a Document object. While JSON schema is a JSON specific schema to represent message payload, Document object is an AtlasMap internal, but data format agnostic metadata so that the Data Mapper UI can consume and provide an unified mapping experience.

AtlasMap Document object defines a tree of fields. Defining a set of field-to-field mapping is all about the mappings in AtlasMap.

Another thing to know for AtlasMap internal is the modules located here in repository. Module in AtlasMap is a facility to plug-in an individual data format support like Java, JSON and XML. Each module implements roughly following 2 parts:

- Inspection Design Time Service to convert the format specific metadata into AtlasMap Document object
- Runtime SPI to achieve actual mappings
  - AtlasModule: Several methods to be invoked during processing mappings. We'll look into deeper in AtlasModule section.
  - AtlasFieldReader: Read a field value from source payload
  - AtlasFieldWriter: Write a field value into target payload

There is also an overview document for the Data Mapper step in Syndesis, which might help if you look into AtlasMap within Syndesis context. https://github.com/syndesisio/syndesis/blob/master/app/server/docs/design/datamapper.md

3.1. UI

Data Mapper UI is a web based user interface to define a data mapping, built with Angular.

- https://github.com/atlasmap/atlasmap/blob/master/ui/

3.1.1. BOOTSTRAPPING

Bootstrapping the Data Mapper UI requires a bit of configuration. An example bootstrapping component is provided within the project:

https://github.com/atlasmap/atlasmap/blob/master/ui/src/app/lib/atlasmap-data-mapper/components/data-mapper-example-host.component.ts

3.1.2. MODEL

All application data and configuration is stored in a centralized ConfigModel object.

The ConfigModel contains:

- initialization data such as service URLs and source/target document information
- references to our angular2 services that manage retrieving and saving our documents and mapping data
- document / mapping model objects

There are two document models contained within the ConfigModel object, both of type DocumentDefinition. A DocumentDefinition contains information about a source or target document such as the document's name, and fields for that document. Fields are represented by our Field model.

A single MappingDefinition model in the ConfigModel object stores information about field mappings and related lookup tables. Individual mappings are represented in instances of MappingModel, and lookup tables are represented by the LookupTable model.

3.1.3. SERVICE

When the Data Mapper UI Bootstraps, a series of service calls are made to the mapping service (MappingManagementService) and document service (DocumentManagementService).

The document service is used to fetch our source/target document information (name of doc, fields). After these are parsed from the service, they are stored in the ConfigModel's inputDoc and outputDoc DocumentDefinition models.

The mapping service is used to fetch our mappings for the fields mapped from the source to the target document. These mappings (and related lookup tables) are parsed by the management service and stored in the ConfigModel's mappings MappingDefinition model.

3.2. Design Time Service

Design Time Service is a set of REST API which provide background services to be used by the Data Mapper UI behind the scene. There are 4 types of Design Time Service:

3.2.1. Core Service

- Code Location: https://github.com/atlasmap/atlasmap/blob/master/lib/service

- API Reference: Core Service

Core Service provides basic operations which is not specific to the individual data formats, Create/Get/Update/Remove mapping definition stored in Design Time Service local storage, validate mapping, retrieve metadata for available field actions and etc.

3.2.2. Java Service

- Code Location: https://github.com/atlasmap/atlasmap/blob/master/lib/modules/java/service
- API Reference: Java Service

Java Service provides Java inspection service which generate an AtlasMap Document object from Java class name.

3.2.3. JSON Service

- Code Location: https://github.com/atlasmap/atlasmap/blob/master/lib/modules/json/service
- API Reference: JSON Service

JSON Service provides JSON inspection service which generate an AtlasMap Document object from JSON instance or JSON schema.

3.2.4. XML Service

- Code Location: https://github.com/atlasmap/atlasmap/blob/master/lib/modules/xml/service
- API Reference: XML Service

XML Service provides XML inspection service which generate an AtlasMap Document object from XML instance or XML schema.

3.3. Runtime Engine

- Code Location: https://github.com/atlasmap/atlasmap/blob/master/lib/

AtlasMap runtime engine consumes mapping definition file created via Data Mapper UI as well as actual data payload and perform mapping.

Here is a shortest code to process a mapping using AtlasMap runtime engine:

```
AtlasContextFactory factory = atlasContextFactory = DefaultAtlasContextFactory
.getInstance(); ①
AtlasContext context = factory.createContext(new File("./my-mapping.xml")); ②
AtlasSession session = context.createSession(); ③
session.setSourceDocument("myJsonSourceDoc", "{...}"); ④
context.process(session); ⑤
Object targetDoc = session.getTargetDocument("myXmlTargetDoc"); ⑥
```

① AtlasContextFactory is a singleton instance on JVM, which holds global configuration for the

AtlasMap.

② `AtlasContextFactory#createContext(File)` creates AtlasContext which represents an AtlasMap mapping context for each mapping definitions by feeding a mapping definition file.

③ AtlasSession represents a mapping processing session. `AtlasSession` should be created for each execution and should **NOT** be shared among multiple threads.

④ Put a source Document with a corresponding Document ID. Make sure Document ID matches with what is specified in the mapping definition.

⑤ Process a mapping by `AtlasContext#process(AtlasSession)`. This invocation also triggers mapping validation prior to actually perform a mapping.

⑥ Finally take the transformed document out from `AtlasSession` by specifying target Document ID.

3.3.1. AtlasModule and mapping process

AtlasModule is a SPI to be implemented by each modules like Java, JSON and XML. The methods defined in `AtlasModule` are invoked from `AtlasContext` while `AtlasContext#process(AtlasSession)` is in progress.

`AtlasContext#process(AtlasSession)` goes on in following order:

1. `AtlasContext#processValidation(AtlasSession)`
   a. `AtlasValidationService.validateMapping(AtlasMapping)` validates mapping definition
   b. `AtlasModule#processPreValidation(AtlasSession)` for each modules participated in the mapping, validates data format specific things
2. `AtlasModule#processPreSourceExecution(AtlasSession)` for each source modules
3. `AtlasModule#processPreTargetExecution(AtlasSession)` for each target modules
4. for each mapping entries:
   a. `AtlasModule#processSourceFieldMapping(AtlasSession)` for each source fields
      i. Read source field values from source payload with using `FieldReader`
      ii. Apply FieldAction if it's specified for the source field
   b. `AtlasModule#processTargetFieldMapping(AtlasSession)` for each target fields
      i. Convert source field values into target field type with using TypeConverter if needed
      ii. Copy source field values into target fields
      iii. Apply FieldAction if it's specified for the target field
      iv. Write target field values into target payload with using `FieldWriter`
5. `AtlasModule#processPostValidation(AtlasSession)` for each modules
6. `AtlasModule#processPostSourceExecution(AtlasSession)` for each source modules
7. `AtlasModule#processPostTargetExecution(AtlasSession)` for each target modules

3.3.2. TypeConverter

TypeConverter converts one field value to the expected field type. This is automatically invoked during mapping when the actual value is not in expected type. AtlasMap runtime provides OOTB converters for the AtlasMap primitive types here.

3.3.3. FieldAction (Transformation)

FieldAction is a function you can apply on a field value as a part of mapping. AtlasMap provides a variety of FieldActions you can apply in the middle of processing mappings here. Also There is a Reference for all available FieldAction.

3.3.4. FieldReader

Each module implements its own FieldReader to read a field value from document specific payload.

3.3.5. FieldWriter

Each module implements its own FieldWriter to write a field value into document specific payload.

3.3.6. Validation

`AtlasContext#processValidation(AtlasSession)` validates a mapping definition associated with this context. After it's completed, you can retrieve a collection of `Validation` object which represents a validation log.

`processValidation(AtlasSession)` is also invoked as a part of `AtlasContext#process(AtlasSession)` prior to actually perform a mapping. In this case, validation results are converted to Audit.

3.3.7. Audit

`Audit` represents an audit log which is emitted from runtime engine during processing a mapping. After `AtlasContext#process(AtlasSession)` is completed, you can retrieve a collection of `Audit` object by invoking `AtlasSession.getAudits()`.

Chapter 4. camel-atlasmap component

camel-atlasmap is an Apache Camel Component for AtlasMap. This component executes AtlasMap mapping as a part of Camel route processing.

Example usage:

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
    <route>
        <from uri="direct:start" />
        <to uri="atlas:atlasmapping.xml" />
        <to uri="mock:result" />
    </route>
</camelContext>
```

Chapter 5. Reference

5.1. Design Time Service API

5.2. Transformation (FieldAction)

Transformation is a function you can apply on a field value as a part of mapping. There are a variety of transformations you can apply in the middle of processing mappings. This is called FieldAction internally.

TODO: Generate this list automatically from annotation - https://github.com/atlasmap/atlasmap/issues/173

Field Action	Input Type	Output Type	Parameter(s) (*=required)	Description
AbsoluteValue	Number	Number	n/a	Return the absolute value of a number.
Add	Collection/Array/Map	Number	n/a	Add the numbers in a collection, array, or map's values.
AddDays	Date	Date	days	Add days to a date. The default days is 0.
AddSeconds	Date	Date	seconds	Add seconds to a date. The default seconds is 0.
Append	String	String	string	Append a string to the end of a string. The default is to append nothing.
Average	Collection/Array/Map	Number	n/a	Return the average of the numbers in a collection, array, or map's values.

Field Action	Input Type	Output Type	Parameter(s) (*=required)	Description
Camelize	String	String	n/a	Convert a phrase to a camelized string by: Removing whitespaceMaking the first word lowercaseCapitalizing the first letter of each subsequent word
Capitalize	String	String	n/a	Capitalize the first character of a string.
Ceiling	Number	Number	n/a	Return the whole number ceiling of a number.
Concatenate	Collection/Array/Map	String	delimiter	Concatenate a collection, array, or map's values, separating each entry by the delimiter if supplied.
Contains	Any	Boolean	value	Return true if a field contains the supplied value.
ConvertAreaUnit	Number	Number	fromUnit *toUnit *	Convert a number representing an area to another unit. The fromUnit and toUnit parameters can be one of the following: Square FootSquare MeterSquare Mile

Field Action	Input Type	Output Type	Parameter(s) (*=required)	Description
ConvertDistanceUnit	Number	Number	fromUnit *toUnit *	Convert a number representing a distance to another unit. The fromUnit and toUnit parameters can be one of the following: FootInchMeterMileYard
ConvertMassUnit	Number	Number	fromUnit *toUnit *	Convert a number representing a mass to another unit. The fromUnit and toUnit parameters can be one of the following: KilogramPound
ConvertVolumeUnit	Number	Number	fromUnit *toUnit *	Convert a number representing a volume to another unit. The fromUnit and toUnit parameters can be one of the following: Cubic footCubic meterGallonLiter
CurrentDate	n/a	Date	n/a	Return the current date.
CurrentDateTime	n/a	Date	n/a	Return the current date/time.
CurrentTime	n/a	Date	n/a	Return the current time.

Field Action	Input Type	Output Type	Parameter(s) (*=required)	Description
DayOfWeek	Date	Number	n/a	Return the day of the week for a date, from 1 (Monday) to 7 (Sunday).
DayOfYear	Date	Number	n/a	Return the day of the year for a date, from 1 to 365 (or 366 in a leap year).
Divide	Collection/Array/Map	Number	n/a	Divide each entry in a collection, array, or map's values by its subsequent entry (A "normal" division would only involve 2 entries).
EndsWith	String	Boolean	string	Return true if a string ends with the supplied string (including case).
Equals	Any	Boolean	value	Return true if a field is equal to the supplied value (including case).
FileExtension	String	String	n/a	Retrieve the extension, without the dot ('.'), of a string representing a file name.
Floor	Number	Number	n/a	Return the whole number floor of a number.

Field Action	Input Type	Output Type	Parameter(s) (*=required)	Description
Format	Any	String	template *	Return a string that is the result of substituting a field's value within a template containing placeholders like %s, %d, etc., similar to mechanisms available in programming languages like Java and C.
GenerateUUID	n/a	String	n/a	Create a string representing a random UUID.
IndexOf	String	Number	string	Return the first index, starting at 0, of the supplied string within a string, or -1 if not found.
IsNull	Any	Boolean	n/a	Return true if a field is null.
LastIndexOf	String	Number	string	Return the last index, starting at 0, of the supplied string within a string, or -1 if not found.
Length	Any	Number	n/a	Return the length of the field, or -1 if null. For collections, arrays, and maps, this means the number of entries.

Field Action	Input Type	Output Type	Parameter(s) (*=required)	Description
Lowercase	String	String	n/a	Convert a string to lowercase.
Maximum	Collection/Array/Map	Number	n/a	Return the maximum number from the numbers in a collection, array, or map's values.
Minimum	Collection/Array/Map	Number	n/a	Return the minimum number from the numbers in a collection, array, or map's values.
Multiply	Collection/Array/Map	Number	n/a	Multiply the numbers in a collection, array, or map's values.
Normalize	String	String	n/a	Replace consecutive whitespace characters with a single space and trim leading and trailing whitespace from a string.
PadStringLeft	String	String	padCharacter *padCount *	Insert the supplied character to the beginning of a string the supplied count times.
PadStringRight	String	String	padCharacter *padCount *	Insert the supplied character to the end of a string the supplied count times.

Field Action	Input Type	Output Type	Parameter(s) (*=required)	Description
Prepend	String	String	string	Prepend a string to the beginning of a string. The default is to prepend nothing.
ReplaceAll	String	String	match *newString	Replace all occurrences of the supplied matching string in a string with the supplied newString. The default newString is an empty string.
ReplaceFirst	String	String	match *newString	Replace this first occurrence of the supplied matching string in a string with the supplied newString. The default newString is an empty string.
Round	Number	Number	n/a	Return the rounded whole number of a number.
SeparateByDash	String	String	n/a	Replace all occurrences of whitespace, colons (:), underscores (_), plus (+), or equals (=) with a dash (-) in a string.

Field Action	Input Type	Output Type	Parameter(s) (*=required)	Description
SeparateByUnderscore	String	String	n/a	Replace all occurrences of whitespace, colon (:), dash (-), plus (+), or equals (=) with an underscores (_) in a string.
StartsWith	String	Boolean	string	Return true if a string starts with the supplied string (including case).
Substring	String	String	startIndex *endIndex	Retrieve the segment of a string from the supplied inclusive startIndex to the supplied exclusive endIndex. Both indexes start at zero. The default endIndex is the length of the string.

Field Action	Input Type	Output Type	Parameter(s) (*=required)	Description
SubstringAfter	String	String	startIndex *endIndexmatch *	Retrieve the segment of a string after the supplied match string from the supplied inclusive startIndex to the supplied exclusive endIndex. Both indexes start at zero. The default endIndex is the length of the string after the supplied match string.
SubstringBefore	String	String	startIndex *endIndexmatch *	Retrieve the segment of a string before the supplied match string from the supplied inclusive startIndex to the supplied exclusive endIndex. Both indexes start at zero. The default endIndex is the length of the string before the supplied match string.

Field Action	Input Type	Output Type	Parameter(s) (*=required)	Description
Subtract	Collection/Array/Map	Number	n/a	Subtract each entry in a collection, array, or map's values from its previous entry (A "normal" subtraction would only involve 2 entries).
Trim	String	String	n/a	Trim leading and trailing whitespace from a string.
TrimLeft	String	String	n/a	Trim leading whitespace from a string.
TrimRight	String	String	n/a	Trim trailing whitespace from a string.
Uppercase	String	String	n/a	Convert a string to uppercase.

5.3. Java API Reference (Javadoc)

5.4. Terminology

AtlasMap terminology

Term	Definition	Example
AtlasMapping	The top-level mapping file containing the mapping instructions to execute at runtime	atlasmapping.xml, atlasmapping.json
Audit	An Audit is informational tracing data captured runtime execution in order to provide user feedback	Useful during testing cycles, or when user does not have direct access to the runtime logs
Collection	A data type used in conversion that repeats 0 or more times	Used to process Arrays and Lists
Data Source	The definition of an input or output data format within an AtlasMapping file	atlas:java, atlas:json, atlas:xml, etc

Term	Definition	Example
Field	The base type used to describe a data value used in a Mapping	JavaField, JsonField, XmlField, etc
Field Action	A function to perform on a given value of an input or output field	CurrentDate, Trim, SubString, etc
Field Type	Normalized convention to describe the type of data expected within the value of a field	String, Integer, Long, Number, Date Time, etc
Lookup Table	A cross reference table used at runtime to define a data conversion that varies based on the value of the input field	Used for mapping enumerations
Mapping	A grouping of input Field(s), output Field(s) and optionally Field Action(s) that defines the conversion to be performed at runtime within an AtlasMapping file	JavaField → JsonField, XmlField → JsonField + Uppercase, etc
Mapping Type	The specific type of conversion that should be performed at runtime	Map, Combine, Separate, Collection or Lookup
Validation	A Validation is a syntax check of a provide mapping file to ensure execution may proceeed	Useful during testing cycles, and at runtime to avoid complicated and misleading exception stack traces

AtlasMap Data Format terminology

Term	Definition	Example
Boxed Primitive	A type of primitive that have a 'null' value	Integer, Long, Boolean, etc
(Unboxed) Primitive	A type of primitive that cannot be 'null' and generally has a default initialized value	int, long, boolean, etc
Rooted	JSON documents that have a parent node identifying to contents of the JSON document	{ "Contact": { "firstName": "Yu", "lastName": "Darvish" } }
Unrooted	JSON documents that do not have a parent node identifying the contents of the JSON document	{ "firstName": "Yu", "lastName": "Darvish" }
Qualified	Xml elements and/or attributes that contain the namespace prefix	<ns1:Contact ns1:firstName="Yu" ns1:lastName="Darvish" />
Unqualified	Xml elements and/or attributes that do not contain the namespace prefix	<Contact firstName="Yu" lastName="Darvish" />