

AtlasMap Developer Guide

The AtlasMap Team

Version 2.4.0-SNAPSHOT, 2022-01-24

AtlasMap Developer Guide

| | |
|---|----|
| 1. Introduction | 2 |
| 2. Quickstart | 3 |
| 2.1. Running AtlasMap Data Mapper UI within Synthesis | 3 |
| 2.2. Running AtlasMap Data Mapper UI standalone | 3 |
| 2.3. Running AtlasMap build | 3 |
| 2.4. Tips for UI developer | 3 |
| 3. Internal Design | 4 |
| 3.1. UI | 4 |
| 3.1.1. @atlasmap/core | 5 |
| 3.1.2. @atlasmap/atlasmap | 6 |
| 3.1.3. @atlasmap/standalone | 6 |
| 3.2. Mapping State | 6 |
| 3.3. Design Time Service | 6 |
| 3.3.1. Core Service | 6 |
| 3.3.2. Java Service | 6 |
| 3.3.3. JSON Service | 6 |
| 3.3.4. XML Service | 7 |
| 3.3.5. DFDL Service | 7 |
| 3.3.6. Kafka Connect Service | 7 |
| 3.4. Runtime Engine | 7 |
| 3.4.1. AtlasModule and mapping process | 8 |
| 3.4.2. TypeConverter | 8 |
| 3.4.3. FieldAction (Transformation) | 9 |
| 3.4.4. FieldReader | 9 |
| 3.4.5. FieldWriter | 9 |
| 3.4.6. Validation | 9 |
| 3.4.7. Audit | 9 |
| 4. camel-atlasmap component | 10 |
| 5. Reference | 11 |
| 5.1. Design Time Service API | 11 |
| 5.1.1. Core | 11 |
| 5.1.2. Java | 11 |
| 5.1.3. JSON | 11 |
| 5.1.4. XML | 11 |
| 5.1.5. DFDL | 11 |
| 5.2. Transformation (FieldAction) | 11 |
| 5.3. Java API Reference (Javadoc) | 22 |
| 5.4. Coverage Reports | 22 |

| | |
|---|----|
| 5.4.1. Java | 22 |
| 5.4.2. UI/atlasmap | 22 |
| 5.4.3. UI/atlasmap-core | 22 |
| 5.4.4. UI/atlasmap-standalone | 22 |
| 5.5. Terminology..... | 22 |

```
<a      href="https://maven-badges.herokuapp.com/maven-central/io.atlasmap/atlas-parent/"></a>      <a      href="<a
href="https://badge.fury.io/js/%40atlasmap%2Fatlasmap">">%40atlasmap%2Fatlasmap</a></a>
```

 Main   Checked  [Codacy Grade] [Codacy Coverage]  chat  [atlasmap]

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 embedded 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 https://repo1.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

TODO

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 a React 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 ADM archive 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/main/app/server/docs/design/datamapper.md>

3.1. UI

The AtlasMap Data Mapper UI is a web based user interface to define a data mapping, built with [React](#).

- <https://github.com/atlasmap/atlasmap/blob/main/ui/>

It consists of 3 sub-packages:

- [@atlasmap/core](#) - UI side business logic which is used by the presentation layer, [@atlasmap/atlasmap](#). Published in NPM repository - <https://www.npmjs.com/package/@atlasmap/core>
- [@atlasmap/atlasmap](#) - The presentation layer of AtlasMap Data Mapper UI built with React. `<AtlasmapProvider>` is meant to be placed in any React Web application. Published in NPM repository - <https://www.npmjs.com/package/@atlasmap/atlasmap>
- [@atlasmap/standalone](#) - A thin React bootstrapping component to spin up AtlasMap Data Mapper UI in standalone mode.

3.1.1. @atlasmap/core

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.

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.

3.1.2. @atlasmap/atlasmap

TODO

3.1.3. @atlasmap/standalone

TODO

3.2. Mapping State

As you add fields into mapping, mapping transitions to a different state. Here is a state machine diagram for current mapping state transition.

[AtlasMap Mapping State]

On the other hand, once you enable Conditional Mapping by pressing **f(x)** button upper left in the UI, it enters into an advanced **expression** mode and gets out of this state transition.

3.3. 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:

3.3.1. Core Service

- Code Location: <https://github.com/atlasmap/atlasmap/blob/main/lib/service>
- API Reference: [API](#)

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.

3.3.2. Java Service

- Code Location: <https://github.com/atlasmap/atlasmap/blob/main/lib/modules/java/service>
- API Reference: [API](#)

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

3.3.3. JSON Service

- Code Location: <https://github.com/atlasmap/atlasmap/blob/main/lib/modules/json/service>
- API Reference: [API](#)

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

3.3.4. XML Service

- Code Location: <https://github.com/atlasmap/atlasmap/blob/main/lib/modules/xml/service>
- API Reference: [API](#)

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

3.3.5. DFDL Service

- Code Location: <https://github.com/atlasmap/atlasmap/blob/main/lib/modules/dfdl/service>
- API Reference: [API](#)

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

3.3.6. Kafka Connect Service

- Code Location: <https://github.com/atlasmap/atlasmap/blob/main/lib/modules/kafka-connect/service>
- API Reference: [API](#)

Kafka Connect Service provides Kafka Connect inspection service which generate an AtlasMap Document object from Kafka Connect schema such as JSON or AVRO.

3.4. Runtime Engine

- Code Location: <https://github.com/atlasmap/atlasmap/blob/main/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(); ❶  
AtlasContext context = factory.createContext(new File("./my-mapping.adm")); ❷  
AtlasSession session = context.createSession(); ❸  
session.setSourceDocument("myJsonSourceDoc", "{...}"); ❹  
context.process(session); ❺  
Object targetDoc = session.getTargetDocument("myXmlTargetDoc"); ❻
```

❶ [AtlasContextFactory](#) is a singleton instance on JVM, which holds global configuration for the AtlasMap.

❷ [AtlasContextFactory#createContext\(File\)](#) creates [AtlasContext](#) which represents an AtlasMap mapping context for each mapping definitions by feeding an ADM archive file exported from the AtlasMap Data Mapper UI.

- 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.

3.4.1. 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

3.4.2. 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](#).

3.4.3. 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.

3.4.4. FieldReader

Each module implements its own [FieldReader](#) to read a field value from document specific payload.

3.4.5. FieldWriter

Each module implements its own [FieldWriter](#) to write a field value into document specific payload.

3.4.6. 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](#).

3.4.7. 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



The latest version of `camel-atlasmap` component is now maintained in Apache Camel upstream <https://camel.apache.org/components/latest/atlasmap-component.html> <https://github.com/apache/camel/tree/main/components/camel-atlasmap>

We will keep maintaining Camel2 version of `camel-atlasmap` component in this repository for a while, however for Camel3 you need to go with the one from Apache Camel upstream.

`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.adm" />
    <to uri="mock:result" />
  </route>
</camelContext>
```

Chapter 5. Reference

5.1. Design Time Service API

5.1.1. Core

5.1.2. Java

5.1.3. JSON

5.1.4. XML

5.1.5. DFDL

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. |

| Field Action | Input Type | Output Type | Parameter(s) (*=required) | Description |
|--------------|----------------------|-------------|---------------------------|---|
| 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. |
| Camelize | String | String | n/a | Convert a phrase to a camelized string by: Removing whitespace Making the first word lowercase Capitalizing 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. |

| Field Action | Input Type | Output Type | Parameter(s) (*=-required) | Description |
|---------------------|------------|-------------|-------------------------------|---|
| 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 |
| 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: FootInchMeter MileYard |
| 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 |

| Field Action | Input Type | Output Type | Parameter(s) (* = required) | Description |
|-------------------|------------|-------------|--------------------------------|--|
| 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. |
| 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). |

| Field Action | Input Type | Output Type | Parameter(s) (*=required) | Description |
|---------------|----------------------|-------------|---------------------------|---|
| 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. |

| Field Action | Input Type | Output Type | Parameter(s) (* = required) | Description |
|--------------|----------------------|-------------|--------------------------------|---|
| 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. |
| 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. |

| Field Action | Input Type | Output Type | Parameter(s) (*=-required) | Description |
|----------------|------------|-------------|-------------------------------|--|
| 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. |
| 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. |

| Field Action | Input Type | Output Type | Parameter(s) (* = required) | Description |
|-----------------------|------------|-------------|--------------------------------|--|
| 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. |
| SeparateByUnder score | 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). |

| Field Action | Input Type | Output Type | Parameter(s) (* = required) | Description |
|----------------|------------|-------------|------------------------------------|---|
| 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. |
| SubstringAfter | String | String | startIndex *endIndex match * | 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. |

| Field Action | Input Type | Output Type | Parameter(s) (* = required) | Description |
|-----------------|----------------------|-------------|----------------------------------|---|
| SubstringBefore | String | String | startIndex *endIndex match | 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. |
| 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. |

| Field Action | Input Type | Output Type | Parameter(s) (* = required) | Description |
|--------------|------------|-------------|--------------------------------|--------------------------------|
| Uppercase | String | String | n/a | Convert a string to uppercase. |

5.3. Java API Reference (Javadoc)

5.4. Coverage Reports

5.4.1. Java

5.4.2. UI/atlasmap

5.4.3. UI/atlasmap-core

5.4.4. UI/atlasmap-standalone

5.5. 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 |
| 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 |

| Term | Definition | Example |
|--------------|---|---|
| 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 proceed | 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" /> |