FHIR Mapper

Firely

Jun 08, 2021

FIRELY PRODUCTS

The FHIR Mapper is an implementation of the FHIR mapping language, developed by Firely and Healex, available as a plugin for Firely Server. With this add-on you're able to transform data from a variety of formats - HL7 v2 or your custom ones - to other formats such as FHIR/custom resources. It can even be used to map FHIR resources between different profiles and FHIR versions.

More general information and tutorials about the FHIR Mapping Language can be found on the HL7 wiki.

```
map "http://vonk.fire.ly/fhir/StructureMap/FHIRMapperTutorial" = FHIRMapperTutorial

uses "http://hl7.org/fhir/StructureDefinition/FakeInpatientDrugChart" as source
uses "http://hl7.org/fhir/StructureDefinition/Patient" as target
uses "http://hl7.org/fhir/StructureDefinition/Observation" as target

group MapFakeInpatientDrugChart(source src: FakeInpatientDrugChart, target bundle: Bundle)
{
  src -> bundle.id = uuid();
  src -> bundle.entry as entry, entry.resource = create('Patient') as patient then
    TransformPatient(src, patient), TransformPatientPostHandler(src, patient, bundle);
}

// ...
```

On these pages we provide you with the documentation you need to get up and running with the FHIR Mapper, as well as information on how to contact us when you have additional needs: such as a custom implementation or support contract.

CHAPTER

ONE

GETTING STARTED

The FHIR Mapper is available as a paid plugin to the Firely Server - FHIR server. Get in touch with us if you're interested in obtaining it for your needs. Alternatively, you can try the FHIR Mapper in our public test sandbox or by downloading Firely Server using an Evaluation license. All operations described in this documentation are accessible for testing purposes.

1.1 Configuration

The section *Configuring FHIR Mapper on Firely Server* explains how you can configure the plugin in your own Firely Server.

1.2 Running transformations

Once you have setup the FHIR Mapper, you can start *Setting up transformations*.

CHAPTER

TWO

CONFIGURING FHIR MAPPER ON FIRELY SERVER

To configure the FHIR Mapper in your Firely Server installation, follow the steps below. If you don't have a copy yet, get in touch with us if you're interested in purchasing it for your needs.

2.1 Load the engine

1. Ensure the `/plugins` directory exists in your Firely Server installation.

   1.1. The location of this folder is customisable via the `PluginDirectory` property, so if you chose a different directory, ensure it exists instead.

2. Verify that all required DLLs can be found in the plugin directory:

- <path to firely server>/plugins/vonk.plugin.mapping/<version>/Hl7.Fhir.Language.dll
- <path to firely server>/plugins/vonk.plugin.mapping/<version>/Hl7.Fhir.Mapping.dll
- <path to firely server>/plugins/vonk.plugin.mapping/<version>/Hl7.Fhir.Mapping.STU3.Poco.dll
- <path to firely server>/plugins/vonk.plugin.mapping/<version>/Hl7.Fhir.Mapping.R4.Poco.dll
- <path to firely server>/plugins/vonk.plugin.mapping/<version>/Superpower.dll
- <path to firely server>/plugins/vonk.plugin.mapping/<version>/Vonk.Plugin.Mapping.dll
- <path to firely server>/plugins/vonk.plugin.binarywrapper/<version>/Vonk.Plugin.BinaryWrapper.dll

3. Configure the Firely Server settings and check that `$convert` is added as a``WholeSystemInteractions`` to support the convert operation.
4. `text/fhir-mapping` must be added as a MIME type to the `Vonk.Plugin.BinaryWrapper` settings. Please add it as a new value to the `RestrictToMimeTypes` array.
5. Similarly, add `$transform` to `InstanceLevelInteractions` and `TypeLevelInteractions` to declare support for the transform operation. Sample configuration:

```
"SupportedInteractions": {
  "InstanceLevelInteractions": "read, vread, update, patch, delete, history,␣
↪conditional_delete, conditional_update, $validate, $validate-code, $expand,
↪$compose, $meta, $meta-add, $transform",
  "TypeLevelInteractions": "create, search, history, conditional_create,␣
↪compartment_type_search, $validate, $snapshot, $validate-code, $expand, $lookup,
↪$compose, $transform",
  "WholeSystemInteractions": "capabilities, batch, transaction, history, search,␣
↪compartment_system_search, $validate, $convert"
},
```

6. Review the `/administration` path of `PipelineOptions` and make sure that the mapping engine plugin are included using the following namespaces:

```
"Vonk.Plugin.BinaryWrapper",
"Vonk.Plugin.Mapping"
```

Sample configuration:

```
"PipelineOptions": {
  "PluginDirectory": "./plugins",
  "Branches": [
    {
      "Path": "/administration",
      "Include": [
        "Vonk.Core",
        "Vonk.Fhir.R3",
        "Vonk.Fhir.R4",
        //"Vonk.Fhir.R5",
        "Vonk.Repository.Sql.SqlAdministrationConfiguration",
        "Vonk.Repository.Sqlite.SqliteAdministrationConfiguration",
        "Vonk.Repository.MongoDb.MongoDbAdminConfiguration",
        "Vonk.Repository.Memory.MemoryAdministrationConfiguration",
        "Vonk.Subscriptions.Administration",
        "Vonk.Plugins.Terminology",
        "Vonk.Administration",
        "Vonk.Plugin.BinaryWrapper",
        "Vonk.Plugin.Mapping"
      ],
    },
```

6. Start Firely Server :)

2.2 Verifying

To verify that the mapping engine is loaded, do check the metadata with `http(s)://<firely-server-endpoint>/metadata`. If it mentions this in the response:

```
{
  "name": "transform",
  "definition": {
    "reference": "http://hl7.org/fhir/OperationDefinition/StructureMap-transform"
   }
 }
```

That means the plugin is loaded and working.

(alternatively, run `curl -s http://localhost:4080/metadata | jq ".rest[].operation[] | select (.name == \"transform\")"`)

2.3 Troubleshooting

If the verification didn't work for some reason, check the Firely Server logs for the following:

1. In the `Looking for Configuration in these assemblies` section, ensure the dll's are loaded:

```
- <path to firely server>/plugins/vonk.plugin.mapping/<version>/Hl7.Fhir.Language.
↪dll
- <path to firely server>/plugins/vonk.plugin.mapping/<version>/Hl7.Fhir.Mapping.dll
- <path to firely server>/plugins/vonk.plugin.mapping/<version>/Hl7.Fhir.Mapping.
↪STU3.Poco.dll
- <path to firely server>/plugins/vonk.plugin.mapping/<version>/Hl7.Fhir.Mapping.R4.
↪Poco.dll
- <path to firely server>/plugins/vonk.plugin.mapping/<version>/Superpower.dll
- <path to firely server>/plugins/vonk.plugin.mapping/<version>/Vonk.Plugin.Mapping.
↪dll
- <path to firely server>/plugins/vonk.plugin.binarywrapper/<version>/Vonk.Plugin.
↪BinaryWrapper.dll
```

If they're not listed, check that the dll files are available in your `PluginDirectory` directory (`./plugins` by default).

2. Ensure the plugins are being registered with the Firely Server pipeline:

```
Configuration:
/administration
    [...]
    BinaryEncodeConfiguration               [1112] | Services: V | Pipeline: V
    BinaryDecodeConfiguration               [1122] | Services: V | Pipeline: V
    MappingToStructureMapConfiguration      [1500] | Services: V | Pipeline: V
    TransfromOperationConfiguration         [4560] | Services: V | Pipeline: V
```

If they're not listed, double-check your that your `PipelineOptions` are loading the engine plugins.

CHAPTER

THREE

SETTING UP TRANSFORMATIONS

The FHIR Mapper was designed to handle two kinds of transformations:

- Converting a FHIR Mapping Language file into a FHIR StructureMap resource
- Transforming an instance of a custom format into FHIR or vice-versa

Prior to running the data mapping $transform operation, you need to have two things in place:

1. A definition of the data you'll be working with in the StructureDefinition format.

   1.1. If you're converting from FHIR resources, then this is already available in Firely Server. Otherwise:

   1.2. Create a logical model in Forge, see: Create Logical Models.

2. Your mapping file as a StructureMap resource.

   2.1. Once you've written your mapping file, you can use Firely Server's $convert operation to convert it to a StructureMap for you.

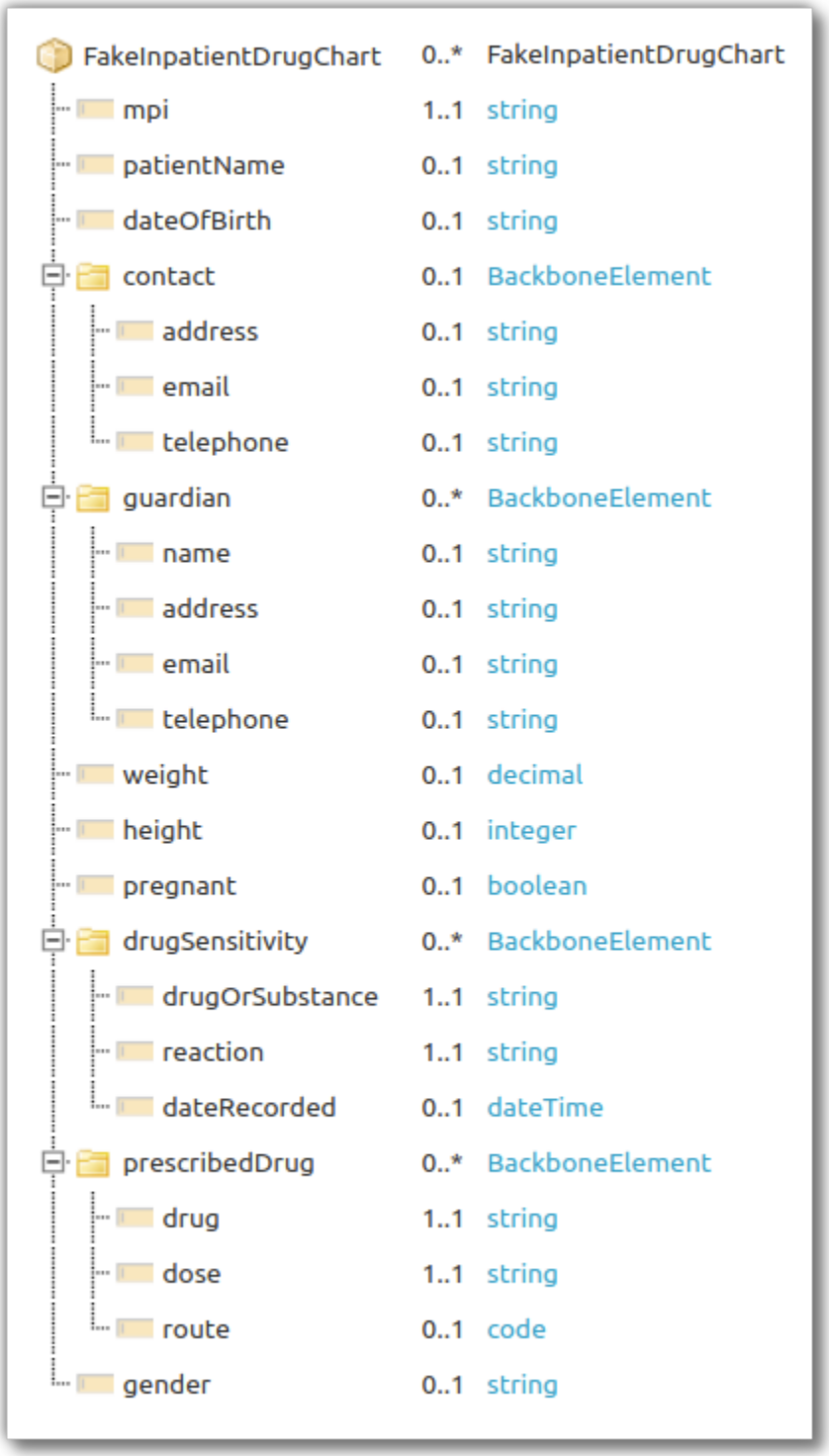
With the definition of data(1) and the mapping file(2) available and uploaded to Firely Server, you can start transforming your data!

The following sections will guide you through all of the steps to setup your transformation and then run it.

3.1 Logical Model + Custom Resource

If you're working with a custom format that you want to transform, you need create a model of your data to describe it to the FHIR Mapper. If you're working with FHIR resources as your source data, Firely Server already has the models available, so you can skip this step.

1. Describe your logical model in Forge: see Create Logical Models. An example model is available and we'll use it in this documentation.



2. Next, convert the logical model you've made to a custom resource:

   2.1. Ensure the `.url` starts with `http://hl7.org/fhir/StructureDefinition` (this is a temporary limitation).

      2.1.1. In our example, change from `http://example.org/mappingengine/fhir/StructureDefinition/FakeInpatientDrugChart` to `http://hl7.org/fhir/`

`StructureDefinition/FakeInpatientDrugChart`.

2.2. Set `.kind` to `resource`.

2.3. Remove the URL from `.type` and set it to just a name.

2.3.1. In our example, change from `http://example.org/fhir/StructureDefinition/FakeInpatientDrugChart` to `FakeInpatientDrugChart`.

2.4. Add a `.id` element at the root level with the name of the custom resource.

2.4.1. In our example, set `.id` to `FakeInpatientDrugChart`.

2.5. Set the `.baseDefinition` to `http://hl7.org/fhir/StructureDefinition/DomainResource`.

2.6. Delete the `type` field from the first `.differential.element` (and `.snapshot.element` if you have it):

2.7. If you only have a `.snapshot` in your model and no `.differential`, rename the `.snapshot` to `.differential` (this is a temporary limitation - basically, ensure that you have a `.differential`).

If you'd like to double-check, this is how our example custom resource looks like now with all the changes applied.

3. Finally, upload your custom resource to Firely Server's `/administration` endpoint:

3.1. `PUT` the resource to `http(s)://<firely-server-endpoint>/administration/StructureDefinition/<custom resource name>`.

3.1.1 In our example, `PUT http://localhost:4080/administration/StructureDefinition/FakeInpatientDrugChart` with the resource in the body.

With the custom resource uploaded to Firely Server's administration point, we have now taught Firely Server about a new resource type!

You can verify this by running `GET http(s)://localhost:4080/<custom resource name>` (in our example `GET http://localhost:4080/FakeInpatientDrugChart`). The query will return 0 search results instead of an error message "Request for not-supported ResourceType(s)".

Next we'll create a mapping between our custom resource and FHIR STU3 resources.

3.2 Mapping File

The mapping files give purpose to our FHIR Mapper: with them, we're able to $transform data from one format to another. These transformation rules can be represented in two formats:

- FHIR Mapping Language syntax (human-readable)
- FHIR StructureMap resource (machine-readable)

These two formats are isomorphic and can be converted between each other.

The FHIR Mapper operates on a StructureMap resource, so let's convert the mapping file to a StructureMap resource and upload to our Firely Server.

1. `POST http(s)://<firely-server-endpoint>/administration/$convert` with the body as your mapping file and the `Content-Type` header set to `text/fhir-mapping;charset=utf-8` to convert your mapping file to a StructureMap. The operation parses your mapping file to check it is valid. You will receive an OperationOutcome if an syntax error is encountered, including a hint on how to fix it.

   1.1. In our example, `POST http://localhost:4080/administration/$convert` with our sample map as the body.

2. `POST http(s)://<firely-server-endpoint>/administration/StructureMap` with the resulting `StructureMap`, or `PUT` to a unique ID. Make sure you don't make duplicates of the StructureMap on the server - so always use `PUT` to update the existing one afterwards. Note down logical ID of your map.

   2.1. In our example, add `"id": "tutorial",` to the StructureMap received in step 1 and upload it to `http://localhost:4080/administration/StructureMap/FHIRMapperTutorial`. Thus `FHIRMapperTutorial` is the logical ID we're working with.

Note: The $convert operation supports a persist option. By adding ?persist=true to the $convert URL, the FHIR Mapper will automatically store the StructureMap. StructureDefinition.id and hence the location of the StructureMap will be automatically chosen based on StructureMap.name.

With the structure map and the logical model uploaded to Firely Server, you are now ready to run your transformations (*Running transformations*).

CHAPTER

FOUR

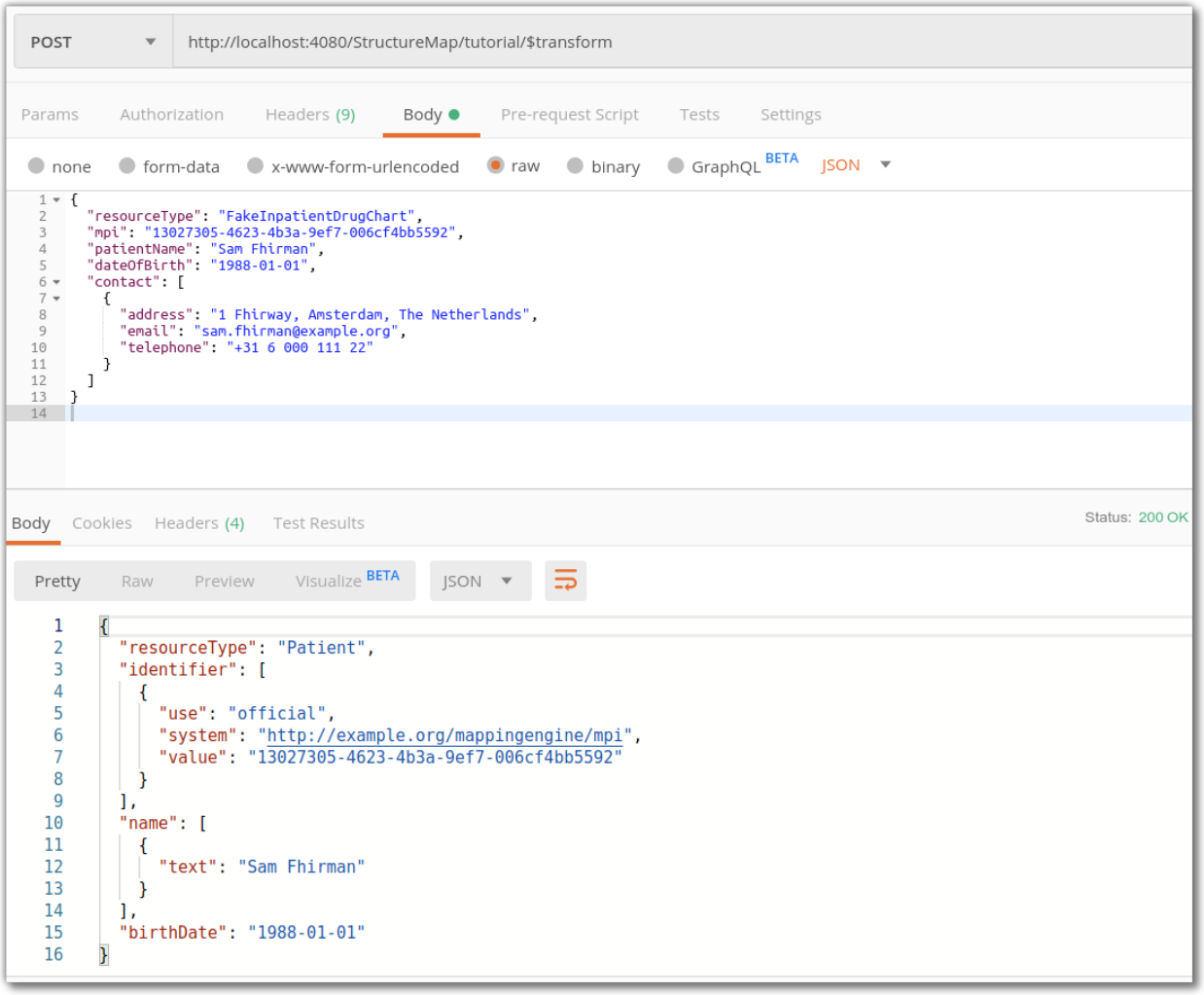
RUNNING TRANSFORMATIONS

Once your transforms are setup (*Setting up transformations*), you can run your data conversion!

To do so, `POST http(s)://<firely-server-endpoint>/administration/StructureMap/<logical id>/$transform` with content to transform as the resource body. Make sure to set the `Content-Type` to either `application/json` or `application/xml` accordingly.

For the example we've been working with so far, `POST http://localhost:4080/administration/StructureMap/FHIRMapperTutorial/$transform` with a sample resource as the body.

Simplified sample call:



With this done, your custom resource and mapping file is registered in Firely Server, and your data transformation is working. Gefeliciteerd!

CHAPTER

FIVE

SUPPORTED MAPPING LANGUAGE FEATURES

The FHIR Mapper implements a broad set of features of the FHIR Mapping Language, allowing you write transformations to handle a variety of use cases. Please note that the FHIR Mapping Language is still under active development by the FHIR community. The following page provides an overview of all features that are supported by our engine.

5.1 Metadata

Metadata can be added to a StructureMap based on a FHIR Mapping Language script by using a /// comment at the beginning of the file. It's possible to set:

- StructureMap.title
- StructureMap.version
- StructureMap.status

```
/// version = 0.1
/// title = "FHIR Mapper Tutorial : FakeInpatientDrugChart"
/// status = draft

map "http://server.fire.ly/fhir/StructureMap/FHIRMapperTutorial" = FHIRMapperTutorial
```

Imported StructureDefinitions can be annotated with an alias that can be used instead of the type name throughout the mapping file:

```
uses "http://hl7.org/fhir/StructureDefinition/Patient" alias PatientAlias as source
group MapPatient(source src: PatientAlias, target bundle: Bundle) {...}
```

5.2 Flow control

In some cases it is necessary to control which order mapping groups are executed. It's possible to chain mapping groups in order for them to be evaluated subsequently:

```
group MapFakeInpatientDrugChart(source src: FakeInpatientDrugChart, target bundle:␣
↪Bundle)
{
  src -> bundle.id = uuid();
  src -> bundle.entry as entry, entry.resource = create('Patient') as patient then
    TransformPatient(src, patient), TransformPatientPostHandler(src, patient, bundle);
}
```

Here `TransformPatient` is called first followed by `TransformPatientPostHandler`, allowing to refine the created Patient resource or to use its content in another resource mapping.

5.3 Transformation functions

The FHIR Mapping language defines a series of transformation functions that can alter the source content before being copied to a target. The following functions are currently supported by the FHIR Mapper.

1. `create('<type>')` - explicitly create an element to pass it into a subsequent rule / mapping group:

```
src.guardian as guardian -> patient.contact = create('BackboneElement') as contact␣
↪collate then {...}
```

2. `dateOp('<input>', '<date | dateTime>')` - transform a string to a FHIR date or dateTime. The input must match the FHIR data type specification:

```
src.dateOfBirth as dateOfBirth -> patient.birthDate = dateOp(dateOfBirth, 'date');
```

Additional parameters are supported:

- `dateOp('<input>', '<inputFormat>', '<date | dateTime>')`
- `dateOp('<input>', '<inputFormat>', '<outputFormat>', '<outputType>')`. Custom types for other information models then FHIR are supported as the outputType. See Custom date and time format strings for available format strings.

3. `uuid()` / `uuid('<name>', '<3 | 5>')` - create a random UUID. See RFC4122 for more information.

```
src -> tgt.id = uuid();
```

4. `cc('<text>')` / `cc('<CodeSystemCanonical>', '<code>', '<DisplayValue>')` - create a CodeableConcept:

```
src -> observation.category = cc('http://hl7.org/fhir/observation-category', 'vital-
↪signs', 'Vital Signs');
```

5. `id('<CodeSystemCanonical>', '<identifier>')` - create an Identifier:

```
src.mpi as mpi -> patient.identifier = id('http://server.fire.ly/fhir/CodeSystem/mpi
↪', mpi) as identifier, identifier.use = 'official';
```

6. `c('<CodeSystemCanonical>', '<code>', '<DisplayValue>')` - create a Coding.
7. `cast(source, '<type>')` - cast source to a certain different type. The following cast options are supported:

	string	code	id	mark-down	uri	oid	uuid	canon-i-cal	url	base64-Bi-nary	in-te-ger	un-signed-Int	pos-i-tivInt	dec-i-mal	boolean	date-Time	time
string	X	X	X	X	X	X	X	X	X	X	X						
in-te-ger											X	X	X				
dec-i-mal														X			
boolean															X		
date-Time																X	
time																	X

8. `translate(source, map_uri, output)` - transform codes using a ConceptMap by its canonical URL. The ConceptMap must be available on the `/administration` endpoint. Note that only `equal` and `equivalent` equivalences are supported.

```
src.gender as gender -> patient.gender = translate(gender, 'http://server.fire.ly/
↪fhir/ConceptMap/MyFakePatientGender', 'code');
```

9. `truncate(source, maxLength)` - shorten the source input - which must be a string - to maxLength by cutting it off.

```
src.name as name, name.text as text -> tgt.name as name, name.text = truncate(text,␣
↪10);
```

5.4 Target List modes

FHIR Mapper supports the `collate` target list mode: so if you have multiple rules that create elements within one backbone element, and you'd like all elements to go into one backbone element, you need to use collate - otherwise the engine will create multiple backbone elements with only one element each.

```
src.identifierPart1 as value -> tgt.identifier = create('Identifier') as identifier,␣
↪identifier.value = value;
src -> tgt.identifier as identifier collate, identifier.system = 'TestSystem';
```

5.5 Source Content

- type
- min..max
- default
- list-option

5.6 FHIRPath Checks

A mapping rule can be conditionally blocked from running by including a FHIRPath statement as a `where` selector:

```
src.weight where "weight.exists()" -> bundle.entry as entry,
          entry.resource = create('Observation') as observation
          then TransformObservationWeight(src, patient, observation);
```

Please note that the FHIRPath result set is selected on the source of the mapping rule. Even if you select src.<element> as your input for the target transformation, the FHIRPath is run on `src` and not on `<element>` - so in our example, you still have to say `weight.exists()`, not `$this.exists()`. It's even possible to use FHIRPath variables like `$this`.

Similar to `where`, FHIR mapper also supports `check` - using that will raise an error if the condition fails.

5.7 Logging

For debugging purposes source content can be dumped as an OperationOutcome via a `log` statement. A log statement can include an arbitrary FHIRPath statement and is executed on the source of the transformation rule:

```
patient.id as patientId log "$this" -> observation.subject = create('Reference') as␣
↪subject,
    subject.reference = evaluate(patientId, '\'Patient/\' + $this');
```

To see the debugging output StructureMap.experimental needs to be set to `true`.

5.8 Default mapping groups

In order to accommodate the fact that neither <<types>> or <<type+>> annotation are supported on a group level, the FHIR Mapper implements a default copy mechanism. A source element can be mapped to a target directly using the "Simple Form" `src.element -> tgt.element` if the source and target element consist of the same type. No casts are possible. In case of a type mismatch, the copy rule is silently ignored. For choice types, the target type is being derived from the src type.

5.9 Unsupported features

- Transformation functions:
  - escape
  - append
  - reference
  - pointer
  - qty
  - cp
- The following list modes on a target transform are not supported:
  - first
  - last

  - share
- <<stereotypes>> for mapping groups
- Extending groups
- conceptmaps embedded in the mapping file (they have to be uploaded to Firely Server instead)
- Using the “as queried” / “as produced” modes when importing a StructureDefinition

CHAPTER

SIX

SUPPORTED SOURCE DATA FORMATS

All components of the FHIR Mapper are designed to handle customs data formats as the source for a mapping to FHIR. By default the FHIR Mapper supports reading and writing 'Custom FHIR resources' (Custom Resources), however it is also possible to provide data in other formats besides the JSON / XML serialization format of FHIR. The mapping engine therefore supports different 'adapters' which can be used to read in other formats natively.

FHIR Version / Content format	Custom Resources (FHIR)	CSV	HL7 v2	VCF	HL7 (C)-CDA
FHIR STU3	X	X		X	X
FHIR R4	X	X	X	X	X

6.1 CSV

The FHIR Mapper supports mapping a comma-separated values (CSV) file without any prior setup (e.g. creating StructureDefinitions). You can POST the CSV file as the HTTP Body to the $transform operation and access its metadata and content within a StructureMap.

To enable CSV support, please adjust the BinaryWrapper settings in your appsettings.instance.json to allow Firely Server to accept the `text/csv` Content-Type header:

```
"Vonk.Plugin.BinaryWrapper":{
  "RestrictToMimeTypes": ["application/pdf", "text/plain", "image/png", "image/jpeg",
↪"text/fhir-mapping", "text/csv"]
}
```

In order to fully use the type-safety features of the FHIR Mapping Language, a StructureDefinition is generated automatically by the FHIR Mapper when a CSV file is received. Depending on the composition of the CSV, two different formats are used:

1. CSV file contains a header with column metadata:

For example, given a CSV file that contains the following rows:

```
ID, Given,        Family,     Gender
1,  Peter James,  Chalmers,   M
2,  Sandy,        Notsowell,  F
```

The following StructureDefinition would be produced in the background while executing $transform:

The .hasHeader child element indicates if the CSV mapping adapter was configured to interpret the first row of the CSV file as a header record. This behaviour can be indicated according to RFC 4180 by using the `header` parameter in the Content-Type header, ie: `text/csv; header=present`.

All other column identifiers are exposed as child elements of the .record BackboneElement. They can directly be used in a mapping rule:

```
src.record as record then
{
  record.ID as id -> tgt.identifier = id('<system>', id);
  record.Given as given -> tgt.name as name, name.given = given;
  record.Family as family -> tgt.name as name collate, name.family = family;
  record.Gender as gender -> tgt.gender = translate('<ConceptMap>', gender, 'code');
};
```

2. CSV file contains no header metadata:

Regardless if the CSV file contains a header, all elements are accessible for the mapping to FHIR. For files not containing a metadata row, element names for the mapping source are generated dynamically as `field#` (starting with field0):

```
src.record as record then
{
  record.field0 as id -> tgt.identifier = id('<system>', id);
  record.field1 as given -> tgt.name as name, name.given = given;
  record.field2 as family -> tgt.name as name collate, name.family = family;
  record.field3 as gender -> tgt.gender = translate('<ConceptMap>', gender, 'code');
};
```

6.2 (C)-CDA

The FHIR Mapper supports mapping (C)-CDA XML documents to FHIR documents. All files needed to execute these mappings are provided as package in the Firely.FhirMapper.Examples Repository.

It contains:

1. StructureDefinitions respresenting (C)-CDA documents and all corresponding data types. All StructureDefinitions are based on the CDA-Core-2.0 Project from HL7 International. For more information see CDA-Core-2.0 ImplementationGuide.
2. StructureMap / FHIR Mapping Language files for executing the mapping. For C-CDA documents, the mappings are based on an open-source project provided by HL7 International. See ccda-to-fhir GitHub project. For more information about the scope of the mappings see HL7 CCDA Mapping Report. A high-level overview of the mapping can be found here as an Excel Sheet.

To enable (C)-CDA support, please adjust the BinaryWrapper settings in your appsettings.instance.json to allow Firely Server to accept the `application/hl7-sda+xml` Content-Type header:

```
"Vonk.Plugin.BinaryWrapper":{
  "RestrictToMimeTypes": ["application/pdf", "text/plain", "image/png", "image/jpeg",
↪"text/fhir-mapping", "application/hl7-sda+xml"]
}
```

CHAPTER

SEVEN

SPECIAL CONSIDERATIONS

The following special considerations - and features - are handy to keep in mind when working with the FHIR Mapper.

7.1 log statement

FHIR Mapper supports logging with `log`, but it is a bit different from the specification:

1. use `log` in the start source field
2. enclose the log statement in double quotes "
3. enclose plain text within the log statement in single quotes '
4. concatenate values with +

Example:

```
src.patientName as s log"'patient name is ' + $this.patientName" -> tgt.name as t, t.
↪text = s;
```

7.2 where/check statements

Enclose the `where` and `check` FHIRPath statements (documentation) in double quotes:

```
src.contact as s where "address.exists()" -> ...
```

7.3 bulk input

In order to transform data in bulk, submit it in CSV format. JSON format supports only individual transformations.

CHAPTER

EIGHT

RELEASE NOTES

All notable changes to this project will be documented in this file. This project adheres to Semantic Versioning.

8.1 Release 0.7.1 - 2021-05-12

- Fix: CSV entries that are enclosed with a double-qoute (e.g. values that contain a delimiter sign in the value itself) do not contain the surrounding quotes as part of their value anymore when mapped to FHIR

8.2 Release 0.7.0 - 2021-04-09

- Feature: Added an overload to the uuid() mapping function. It allows to create uuids in version 3 and version 5. See *Supported Mapping Language features*.
- Fix: Re-enabled support for (C)-CDA Transformations
- Fix: CSV parser preserves the casing of the CSV headers, uppercase headers don't need to be written as lowercase anymore
- Fix: Segment location is reported in OperationOutcome error message if segment contains too many components

8.3 Release 0.6.0 - 2021-01-27

- Known Issue: The (C)-CDA support is currently disabled due to an issue with accessing the .value element of a complex CDA element. We will publish a hotfix for this as soon as possible.
- Known Issue: CSV parsing lowercases all identifiers, meaning that CSV headers should be accessed lowercase in the map file.
- Feature: Upgraded to Firely SDK version 2.0.3. This new major release includes improved support for the normative version of FHIRPath. See https://github.com/FirelyTeam/firely-net-sdk/wiki/Breaking-changes-in-2.0#changes-to-the-fhirpath-engine for more details.
- Feature: ": <type>" (StructureDefinition.group.rule.source.type) can now be used on the source side of a mapping rule
- Feature: "(min..max)" (StructureDefinition.group.rule.source.min / max) can now be used on the source side of a mapping rule
- Feature: StructureMapSourceListMode (first | not_first | last | not_last | only_one) are now fully implemented
- Feature: $convert is now extracting /// experimental = <bool> from a StructureMap as a metadata annotation

- Experimental support for HL7 v2 has been added. See *Supported source data formats*.
- Feature: Suppport for <<types>> group annotations has been added. Please note that <<type+>> annotations are not yet supported.
- Feature: A "delimiter=<delimiter>" and "header = <absent|present>" parameter can be added to the Content-Type header for text/csv
- Fix: The internal LogicalModel type for CSV representation has been renamed from CSV_Transport to CSV
- Fix: A simple copy rule now checks if the source and target types are the same instance types
- Fix: Let statements with constants can now be used
- Fix: Line numbers have been added to error messages from the CSV parser to improve the debugging experience
- Fix: "/administration" was always included the .reference string of a Reference. Now all references stay as they are defined in the mapping. No absolute reference is being generated when executing $transform
- Fix: Include more details in the error message if the compilation step to IL fails when executing $transform

8.4 Release 0.5.0 - 2020-09-04

- Feature: Support for parsing Variant Call Format files. See *Supported source data formats*.
- Feature: External variables can now be used in a log FHIRPath statement using the %var syntax
- Feature: "default" FHIRPath statements can now be specified in mapping rules, the default FHIRPath statement will be executed if the src element is empty
- Feature: Upgrade to .NET FHIR API v1.9
- Fix: An error is now thrown if a mapping attempts to call a mapping function without a contex (e.g. "src.ele -> create('<resourceType>') as var", note the missing target assignment)

8.5 Release 0.4.0 - 2020-07-26

- Fetaure: Implement $transform as a Type level interaction using Parameter resources as input
- Feature: Upgrade the .NET FHIR API to 1.7
- Feature: $convert?$persist returns a location header
- Feature: External variables can now be used in a where FHIRPath check using the %var syntax
- Feature: Added support for the truncate transformation function
- Feature: Added support for casting a string to an integer
- Feature: Allow any kind of date type as the first parameter of dateOp
- Feature: Support the direct mapping of complex child elements
- Fix: Don't overwrite non-repeating elements if the collate option is being used
- Fix: The initial group selection did not account for type aliases

8.6 Release 0.3.6 - 2020-06-09

- Fix: Using $convert with the ?persist would try to persist the result of the conversion even if there was an error.

8.7 Release 0.3.5 - 2020-05-26

- Feature: As StructureMaps are conformance resources, they are now stored in the Vonk administration endpoint. Please check *Configuring FHIR Mapper on Firely Server* for enabling $convert and $transform on the /administration endpoint. Using the FHIR Mapper on the default branch ("{{BASE_URL}}/") is no longer supported.
- Feature: Support for natively mapping text/csv content to FHIR. See *Supported source data formats*.
- Feature: Adding ?persist=true to $convert will now automatically store the StructureMap in Vonk
- Fix: Improved the error message if an unknown / uninterpretable Content-Type header was sent to $transform
- Fix: $transform could be executed on a StructureMap using a different information model (e.g. executing $transform using FHIR R4 on a StructureMap stored in STU3). This could lead to unexpected behaviour in the mapping execution.
- Fix: If a FHIR Bundle was produced using $transform and the debug mode was enabled, a Bundle of Bundles would be returned. Now, the debug log is integrated into the result bundle.
- Fix: Using a FHIRPath statement in "check" mode always threw an error regardless of the statement
- Fix: A stacktrace is now included in error messages thrown during the execution of $transform
- Fix: Harmonized the dateOp parameters with FHIR data types.
- Fix: The source resource type of the initial mapping group is now checked against the provided resource type when calling $transform
- Fix: Return an exception if it is attempted to create a child of a choice[x] element without passing a concrete type.
- Fix: Circular 'using' statements could lead to a StackOverflow in Vonk

8.8 Release 0.3.4 - 2020-03-23

- Feature: "Cannot resolve symbol" error messages now include a 'GroupId' for improved debugging
- Feature: CCDA transformations are now supported for FHIR version STU3 in addition to R4
- Fix: "@primitivvalue@" is no longer printed when logging static text within a mapping statement
- Fix: "status" metadata information were not copied to the StructureMap by $convert if the mapping file included a comment between the first group and the metadata information

8.9 Release 0.3.3 - 2020-03-10

- Built against Vonk 3.3.0
- Upgraded .NET API to version 1.6
- Added support for reading HL7 (C)-CDA XML files natively. See *Supported source data formats*.

8.10 Release 0.3.2 - 2020-03-03

- Internal release.

8.11 Release 0.3.1 - 2020-02-11

- Internal release.

8.12 Release 0.3.0 - 2020-02-11

- Built against Vonk 3.2.0
- Fix: Error messages about empty groups now contain the corresponding group id
- Fix: Improved internal unit tests
- Fix: Improved handling of the 'collate' target list mode. In some cases the usage of collate resulted in too many repeating elements.
- Feature: "import" statements can now be used. All StructureMaps need to be uploaded first to the Administration Endpoint of Vonk.
- Fix: $transform was not showing up in the CapabilityStatement of Vonk when using FHIR R4
- Feature: Added support for different parameters for the dateOp function. See *Supported Mapping Language features*.
- Feature: Calls to evaluate() which return an empty result set result now in an error message to improve debugging
- Feature: $convert now uses the name of the StructureMap as its id

8.13 Release 0.2.0 - 2019-11-18

- Built against Vonk 3.0.0
- Compatible with Vonk 3.0.0, 3.1.0
- Upgrade to .NET API 1.4.0
- Initial public release

8.14 Release 0.1.0 - 2019-11-18

- Built against Vonk 2.1.0
- Initial internal release