medexter
clinical decision support

Medexter Healthcare

ACHIEVING FHIR CONNECTIVITY IN ARDEN SYNTAX USING THE ARDENSUITE FHIR CONNECTOR

Datum: 01.06.2017

Version: 1.0

Medexter Healthcare GmbH, Borschkegasse 7/5, A-1090 Vienna, Austria, Telephone +43-1-968 03 24, Facsimile +43-1-968 09 22 www.medexter.com, Handelsgericht Wien: FN 225564m, UID: ATU54901101, IBAN: AT602011128014988900, BIC: GIBAATWW

Summary

The aim of this *how-to* instruction manual is to show how to create, configure, and activate a FHIR server connection on an ARDENSUITE Server using a FHIR Connector. Furthermore, we show how to use an active FHIR server connection in Arden Syntax Medical Logic Modules (MLMs) in order to read data from the connected FHIR server.

Document Information

Target Audience

This instruction manual was created for ARDENSUITE users and developers interested in developing or using MLMs in combination with FHIR resources on an ARDENSUITE Server.

Impressum

Media owners, editors, publishers:
Medexter Healthcare GmbH, Borschkegasse 7/5, A-1090 Vienna, Austria
Telephone: +43-1-968 03 24, Fax: +43-1-968 09 22, Internet: www.medexter.com
Email: office@medexter.com
CEO: Klaus-Peter Adlassnig, PhD, MSc

Editorial, project management, coordination:
Klaus-Peter Adlassnig, PhD, MSc

Vienna, June 2017

Version: 1.0

Download at www.medexter.com

Introduction

In this *how-to* instruction manual, we will guide you step-by-step in the creation, configuration, and activation of a FHIR server connection on an ARDENSUITE Server. Furthermore, we illustrate how FHIR connections can be used to retrieve data in Arden Syntax via so-called *curly brace expressions*. Examples include:

- An MLM writing a FHIR patient resource to a connected FHIR server.
- An MLM searching for all FHIR patient resources with a given identifier.
- An MLM reading a FHIR patient resource by ID.
- An MLM reading a FHIR patient resource by ID, modifying a value within the MLM and updating the FHIR patient resource on the connected FHIR server.

Requirements

For optimal use of this *how-to*, please install the following software on your computer:

- The ARDENSUITE IDE and ARDENSUITE Server with FHIR Connector
- A FHIR server (public FHIR servers available for free)
- Web service client

The ARDENSUITE IDE and ARDENSUITE Server with FHIR Connector

In case you do not have access to the ARDENSUITE IDE or the ARDENSUITE Server with FHIR Connector yet, please contact us at support@medexter.com. A 30-day trial version of the ARDENSUITE IDE can also be downloaded here.

FHIR Server

Any publicly available FHIR server may be used for testing purposes. A list of public FHIR servers can be found on the HL7 website.

Web Service Client

MLMs deployed on the ARDENSUITE Server are called using web service communication protocols, e.g., Representational State Transfer (REST). For instructional and testing purposes, we illustrate these calls using a web browser. In this document, we recommend using Google Chrome, together with the Postman browser plugin for the visualization of REST communication.

Files

This *how-to* is accompanied by five files: Four MLM files (extension `.mlm`) and one valid FHIR patient resource XML file (extension `.xml`):

- `FHIRRead`: This MLM reads a FHIR patient resource from a connected FHIR server by patient ID and returns the patient name, identifier, status, etc. as a string.
- `FHIRWriteStatic`: This MLM contains a FHIR patient resource as a string and writes it to a connected FHIR server.
- `FHIRWriteDynamic`: This MLM retrieves a FHIR patient resource from a connected FHIR server by patient ID, modifies it, and executes an update on the FHIR server.
- `FHIRSearchQuery`: This MLM executes a search query on a connected FHIR server and retrieves the query result, which can be iterated inside the MLM.
- `FHIR_Patient_Resource`: A valid FHIR patient resource.

NOTE: *The MLM files can be opened using any standard text editor or viewer, but in order to compile and upload the MLMs, the ARDENSUITE IDE and ARDENSUITE Server are required.*

Preliminaries

Before you can start with the actual *how-to* part of this manual, the MLMs have to be compiled and deployed on the ARDENSUITE Server.

Compiling and Uploading MLMs

Before an MLM can be called, it first has to be compiled and uploaded onto the ARDENSUITE Server. To do this, we refer to the corresponding *how-to* document (*How to Compile, Test, and Deploy MLMs with the ARDENSUITE*), which can be found here. In order to use the *how-to* at hand, please compile and upload all four MLMs that accompany this document.

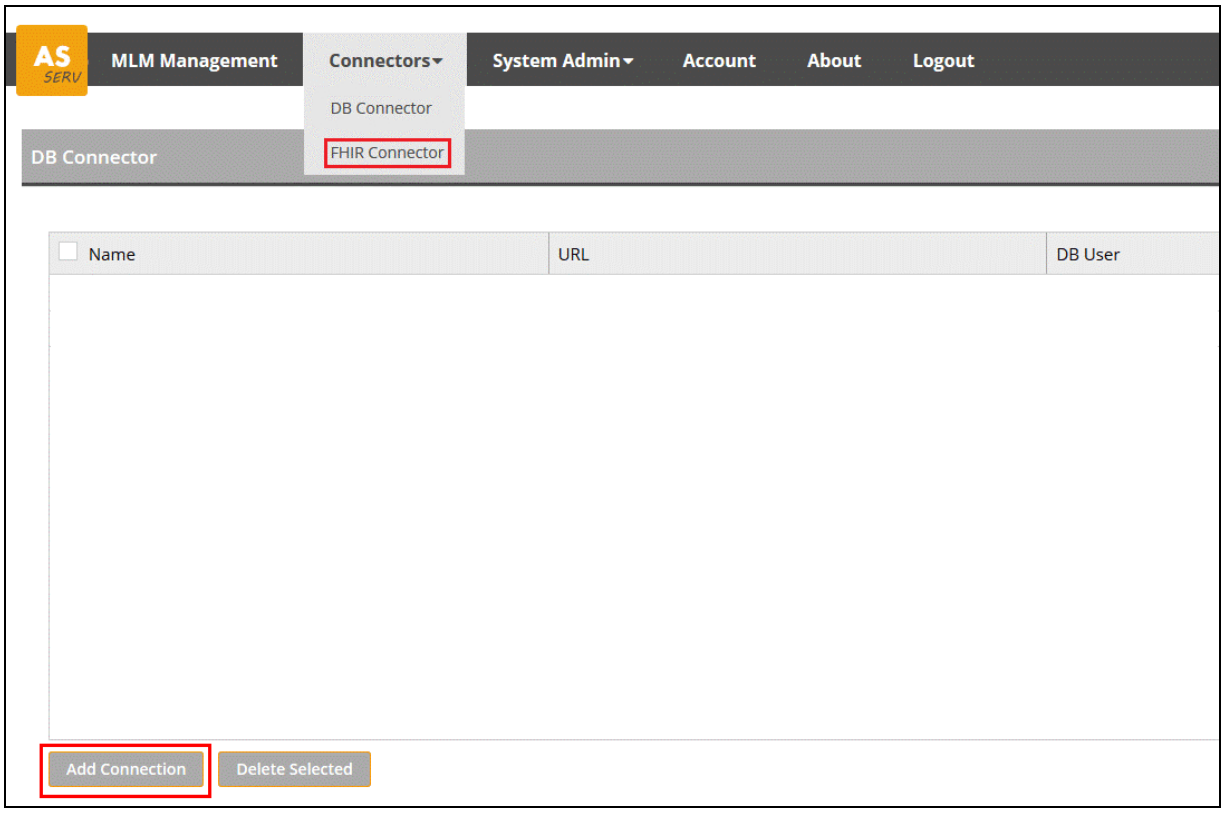
FHIR Connectivity

In this *how-to* we guide you step-by-step in the creation, configuration, and activation of a FHIR server connection on an ARDENSUITE Server as well as the use of a FHIR server connection in Arden Syntax MLMs.

FHIR Connection Setup

In order for MLMs on the ARDENSUITE Server to retrieve data from a FHIR server, a valid FHIR server

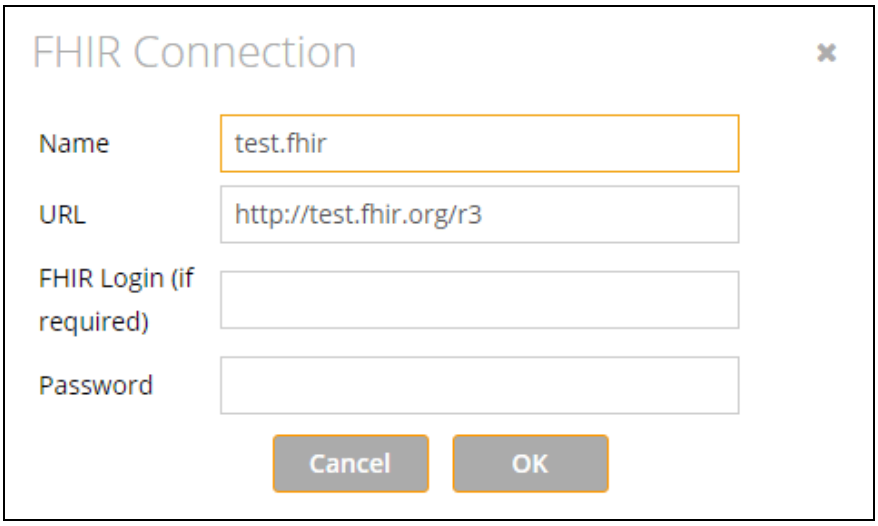
endpoint (REST) has to be configured. After logging into the ARDENSUITE Server frontend, select `FHIR Connector` in the menu bar, and click on the `Add Connection` button on the bottom left (see figure below).



Upon clicking this button, a dialog window will open. Here, please enter the following information:

- **Name:** This name is solely used for displaying purposes in the FHIR Connector frontend in order to make it easier for the user to identify a connection. Thus, any name may be chosen.
- **URL:** The actual FHIR server URL, e.g.: `http://test.fhir.org/r3`
- **FHIR Login (if required):** The FHIR server account username for authentication. On publicly available FHIR servers, no username or password is required (leave this field empty).
- **Password (if required):** The FHIR server account password for authentication. On publicly available FHIR servers, no username or password is required (leave this field empty).

For this *how-to*, an example FHIR server connection in the dialog window could look like this:



Once a connection has been created, it has to be activated. The connection can be activated by clicking on the icon in the `Active` column. This does not verify if the specified server URL indeed provides a valid REST API. If a FHIR connection is active, the icon in the `Active` column turns green (see figure below).

Name	URL	FHIR Login (if required)	Last Update	Active	Edit	Delete
connection1	http://fhir2.healthintersections.com.au/open	n	Mon May 15 10:00:16 CEST 2017			
connection2	http://fhirtest.uhn.ca/baseDstu2		Mon May 15 15:11:52 CEST 2017			
test.fhir	http://test.fhir.org/r3		Tue May 16 11:40:52 CEST 2017			

FHIR Resource Use in MLMs

FHIRWriteStatic MLM

In this MLM (`FHIRWriteStatic.mlm`) a static FHIR resource is created on the connected FHIR server. The JSON data that needs to be supplied in the `Body` segment of the REST call are two strings, specifying the patient's given name and family name:

```
{
"type": "list",
"primaryTime": null,
"applicability": 1,
"values":
[
  {
    "type": "string",
    "value": "hans"
  },
  {
    "type": "string",
    "value": "mayer"
```

```
  }
]
}
```

A FHIR patient resource is defined inside the MLM as a string in xml format, setting the specified given name and family name:

```
FHIRPatient:= "<Patient xmlns=""http://hl7.org/fhir"">
  <text>
    <status value=""generated""></status>
    <div xmlns=""http://www.w3.org/1999/xhtml"">
      <p>" || givenname || " " || familyname || "</p>
    </div>
  </text>

  <identifier>
    <use value=""usual"" />
    <value value=""" || givenname || "." || familyname || """ />
  </identifier>

  <name>
    <family value=""" || givenname || """ />
    <given value=""" || familyname || """ />
  </name>

  <telecom>
    <system value=""phone"" />
    <value value=""(03) 5555 6789"" />
    <use value=""home"" />
  </telecom>

  <gender value=""male"" />

  <birthDate value=""1988-02-18"" />

  <address>
    <line value=""3300 Washtenaw"" />
    <city value=""Ann Harbor"" />
    <state value=""MI"" />
    <postalCode value=""48104"" />
    <country value=""US"" />
  </address>
</Patient>";
```

This resource can be written to a connected FHIR server using the following statements in the data and action slots:

Data Slot: `destinationFHIRCreate := destination {fhir:create:Patient};`

Action Slot: `write FHIRPatient at destinationFHIRCreate;`

NOTE: *The `fhir:create` keyword specifies that a FHIR resource should be created. `Patient` (in the data slot) specifies the FHIR server endpoint for this resource.*

The FHIR server will assign an ID to the newly created FHIR resource. As the Arden Syntax `write` operator is not specified to return anything, the FHIR server response cannot be read within the MLM.

If something went wrong (HTTP status of the REST call not 200), the FHIR server response is written into the FHIR Connector log file (`FHIRConnector.debug.log`), which can be found on the ARDENSUITE Server.

In order to obtain the ID of the newly created FHIR patient resource, the next MLM executes a search query to search for a patient identifier.

If the MLM call was executed successfully (which – however – does not automatically imply that the FHIR server successfully created the patient resource), a JSON/XML object like the following is returned:

```
{
  "type": "string",
  "applicability": 1,
  "value": "The write operation of the fhir ressource was executed for hans
mayer"
}
```

FHIRSearchQuery MLM

This MLM (`FHIRSearchQuery.mlm`) is executing a search query on a connected FHIR server, searching for a matching patient identifier. The JSON data that needs to be supplied in the `Body` segment of the REST call is a string, specifying the search term:

```
{
  "type": "string",
  "value": "hans.mayer"
}
```

The FHIR server response is returned to the MLM as an Arden Syntax object representing the exact same FHIR xml response. This result object is used inside the MLM to retrieve all patient IDs:

```
FHIRBundle := READ {fhir:Patient/_search?identifier=<serachTerm>};

patientO := FHIRBundle.entry.resource.Patient;
if patientO is list then
  for patient in patientO do
    fhirId      := patient.id.value;
    familyName  := patient.name.family.value;
    givenName   := patient.name.given.value;

    output := output || " id: " || fhirId || " name: " || familyName || " "
              || givenName || " | ";
  enddo;
else
  fhirId      := patientO.id.value;
  familyName  := patientO.name.family.value;
  givenName   := patientO.name.given.value;

  output := output || " id: " || fhirId || " name: " || familyName || " "
            || givenName;
```

```
endif;
```

The REST call result will look something like this (if there are multiple patients with the same identifier):

```
{
  "type": "string",
  "applicability": 1,
  "value": " id: 340 name: hans mayer |  id: 341 name: hans mayer | "
}
```

FHIRRead MLM

In this MLM (`FHIRRead.mlm`), a FHIR patient resource is read from a connected FHIR server. The JSON data that has to be supplied in the `Body` segment of the REST call is a number specifying the patient resource ID (e.g., use an ID that was found using the `FHIRSearchQuery.mlm`):

```
{
"type": "number",
"value": 340
}
```

The patient resource is read from the FHIR server and transformed into an Arden Syntax object.

```
testID:= Argument;
FHIRPatient:= READ {fhir:Patient/testID}; // Patient/30857
```

This object may be used in the MLM to process the retrieved FHIR resource data inside the MLM as any other Arden Syntax object (See the PDF file *FHIR_Connector_Concept* for further information on how to navigate through an Arden Syntax FHIR object):

```
familyName  := FHIRPatient.name.family.value;
givenName   := FHIRPatient.name.given.value;
identifiers := FHIRPatient.identifier;
birthday    := FHIRPatient.birthDate.value;
status      := FHIRPatient.text.status.value;
```

The patient name, identifier, status, and patient resource description text are returned as a JSON object like the following:

```
{
  "type": "string",
  "applicability": 1,
  "value": "Name: Max Muster | id text: max.muster | status: generated |
            pText: Gender:male"
}
```

FHIRWriteDynamic MLM

This MLM (`FHIRWriteDynamic.mlm`) executes a `read` operation on a connected FHIR server, trying to retrieve a patient by ID. After retrieving the patient object, it replaces one of the parameters with a new value; in our case we update the patient's given name. The JSON data that has to be supplied in the `Body` segment of the REST call is a list containing a number (patient ID) as the first, and

a string (edit value) as a second element:

```
{
"type": "list",
"primaryTime": null,
"applicability": 1,
"values":
[
          {
          "type": "number",
          "value": "340"
          },
          {
          "type": "string",
          "value": "russ"
          }
]
}
```

To update a FHIR resource, the following statements are used inside the data and action slots:

Data Slot: `destinationFHIRUpdate := destination {fhir:update:Patient/testID};`

Action Slot: `write FHIRPatient at destinationFHIRUpdate;`

The `fhir:update` keyword specifies that an existing FHIR resource should be updated. In the action slot, a `write` statement is executed the same way as it would be for a `fhir:create`. Confirm if the connected FHIR server did indeed update the FHIR patient resource by executing the `FHIRRead.mlm` with the corresponding FHIR patient resource ID.