jProg®

Jeff Murray's Programming Shop, Inc.

Using the cwFhirPath Language

Introduction and Documentation

October 25, 2022

Contents

Prerequisites 2

Introduction 2

The cwFhirPath language 3

- cwFhirPath outputType 3
- note freeFormText 3
- column columnName { 3
- column names with # 4
- column names with # and forEachObj 4
- found varName 4
  - When found is used in a column function 4
  - When found is used in a forEachObj function 4
- } 5
- findResult varName = pathToFhirResourceValue 5
- Writing a path to a FHIR Resource value 5
  - All paths lead to a string value 5
  - The FHIR Resource Base Property 5
  - Properties with object values 5
  - Properties with array values 6
  - Using index to retrieve an array value 6
- forEachObj funcName { 6
- if booleanOperation { 6
  - Using if with varName.Found 7
  - Using *if* with varName.StringValue and varName.StringValue.ToLower and '=' 7
  - Using *if* with varName.StringValue and varName.StringValue.ToLower and 'contains' 7

Using *if* with *not*..... 7
rows {..... 7
rowArrayPath pathToRowObjs..... 8
rowCwFhirPath filename.cwfp..... 8
Automatic .reference resource fetching in findResult paths..... 8

Prerequisites

To make sense of this document you will need general programming knowledge and an understanding of the mechanics of the FHIR R4 API and the JSON format that FHIR Resources take. Here are a couple of links where you can start: https://www.hl7.org/fhir/resourcelist.html and https://www.json.org/json-en.html. You will also need familiarity with the CAREWare PDI and its FHIR Data Source. Documentation for these can be found here (this is the main link to the WIKI PDI FHIR Datasource page).

Introduction

The cwFhirPath language is used by CAREWare FHIR Data Sources to find data in the highly nested JSON format used by HL7's FHIR R4 API and to format that data as PDI CSV rows and columns. During the planning of the CAREWare FHIR Client, we quickly realized that hardcoding the data extraction of PDI columns from FHIR JSON Resources would be a build bottleneck, and tweaking differences across CAREWare's many installations could be tedious and costly. To avoid this issue we came up with the cwFhirPath language to enable jProg staff, or others, to update PDI CSV table and row specs without making a new build of CAREWare.

Masked behind cwFhirPath are a lot of complicated behind-the-scenes FHIR patterns like fetching a 'reference' resource and scanning an array for the desired element and unpackaging FHIR Bundles into rows and executing the PDI's cs_1, cs_2 pattern when multiple codes or coding systems are used. A nice byproduct of this language is that it speeded up the internal development of the first RSR data pull that CAREWare is distributed with. Using this language also provides better transparency in that the exact path within FHIR resources is available to those who wish to check CAREWare's FHIR Datasource implementation.

This documentation is meant to be distributed in a cwFhirPathDocs.zip file with a 'source' and a 'resource' folder inside. Files in these folders are included to provide example source files and example FHIR Resource JSON files that are referred to in this documentation.

We recommend unzipping the documentation with its folder structure intact and using a line numbered text editor like notepad++ to find files in the 'source' and 'resource'

folder. The following documentation refers to these files by name and references example code locations with line numbers.

The example FHIR Resources like 'EpicSandboxPatient.txt' are actual responses received from R4 queries to Epic's public sandbox at https://fhir.epic.com/.

The example *.cwfp files are early proof-of-concept examples of how to approach patterns when pathing through FHIR Resource JSON Objects and converting them to PDI data rows; they are not an examples of what our final, tested cwFhirPath *.cwb pathing will look like. Those examples will be in the CAREWare PDI FHIR Resource Connector.

The cwFhirPath language

The cwFhirPath language has a handful of keywords that are described below. We were able to keep the language small because its sole mission is to pluck data from FHIR R4 Resources for PDI columns. While that task is somewhat complicated, when boiled down to its essence, the intersection of FHIR and PDI patterns results in a finite space. Note that keywords, properties and data operations are case-sensitive, and all required elements of a keyword must be on the same line, including the '{' if specified.

cwFhirPath outputType

The *cwFhirPath* keyword is required to be the first word on the first line, followed by a space and then the type of output expected. There is only one type of output: 'pdi'.

Example on line 1 of exp_client.cwfp:

note freeFormText

> ***note*** is the place for hard coded comments that explain key points about the source.
>
> **freeFormText** must be on the same line.

Example on lines 3 and 4 of exp_client.cwfp:

column columnName {

> ***column*** is a cwFhirPath function used to find a value in its FHIR JSON Resource.
>
> **columnName** is the name of the PDI column you are finding within the FHIR Resource.
>
> ***{*** signifies the opening block of code that finds the data for this column

column names with

In JSON formatted FHIR Resources, a single record can supply an array of codes and coding systems for entities of the same type. The PDI CSV specification handles multiple coding systems through a column name pattern that inserts a counter starting with 1 into the column name. You can control where this counter is inserted by placing a single # in the column name.

See example in 'sources/exp_medication_from_bundle.cwfp' on lines 147 and 161. See PDI documentation for 'exp_medication.csv' for the definition of 'mdc_cs_#' and 'mdc_cs_#_def_code' in the example.

column names with # and forEachObj

It is possible to design forEachObj functions so that more than one JSON object in the scanned array is 'found'. If **columnName** does not have a '**#**', then **findResult** will always use the first object found from the **forEachObj** in its path. If **columnName** has a '**#**', then every object found in the array will be sequenced using the PDI pattern at the '**#**'.

See example in 'sources/exp_medication_from_bundle.cwfp' on lines 151 and 165. In this example, the **forEachObj** returns **found** for any JSON object in the array that has a property named 'system' resulting in all the meds codes and coding systems in the example array.

found varName

> ***found*** is the end-goal of the column and ***forEachObj*** functions indicating at least one value has been found.
> **varName** holds a value from a previous ***findResult*** path

When found is used in a column function

When ***found*** is returned in a *column* function, the string value of the column is set to the value held in the ***findResult*** variable **varName** and all further processing in the ***column*** function is halted.

When found is used in a forEachObj function

When ***found*** is used in a ***forEachObj*** function, even though it looks like you are returning the value held in the ***findResult*** variable **varName**, the ***forEachObj*** function instead returns the object the ***forEachObj*** is evaluating, to be used in the rest of the ***findResult*** path.

}

} Ends its companion opening block of code. The example closing bracket ending a column block started on line 97 is on line 119 in 'source/exp_client.cwfp'

findResult varName = pathToFhirResourceValue

findResult attempts to find a value in a FHIR Resource, and if one is found, stores it in **varName** as a string. If a value is not found, varName will return a zero-length string. **=** signifies the result of **pathToFhirResourceValue** will be assigned to **varName** **pathToFhirResourceValue** is a javascript-style '.'-delineated path that navigates through the various keys, arrays of objects and referenced FHIR Resources and retrieves a value. See 'Writing a path to a FHIR Resource Value' below.

See example on line 101 of 'source/cw_client.cwfp'.

Writing a path to a FHIR Resource value

The ***findResult*** keyword provides a powerful way to navigate to values in the JSON structures of a FHIR Resource.

All paths lead to a string value

findResult will convert any non-string value (like a JSON number) into a string value.

If for some reason your syntactically correct path is unable to find a value, the value will be set to '' (a zero length string). See the ***if*** keyword for how to apply Boolean logic to ***findResult*** variables.

The FHIR Resource Base Property

The PDI's FHIR Datasource will be configured to associate an R4 Resource Query with a cwFhirPath source file that turns the FHIR Resource into PDI columns. The result stored in **varName** will be what **pathToFhirResourceValue** finds on the right side of the '='. A successful FHIR Patient query will return a JSON structure like that in 'resources/EpicSandboxPatient.txt'. A path starts with a base property of its FHIR Resource. Sometimes the base property is the entire path. On line 116 in 'sources/exp_client.cwfp', the base property 'birthDate' has the string/value "1973-06-03" stored with it on line 166 in 'resources/EpicSandboxPatient.txt'.

Properties with object values

In some cases the value of a property is an object. When this is the case, a path can be formed to a property in that object by placing a '.' and then the name of the property. For

example, if you were making a path to get the marital status of the patient, the path ‘maritalStatus.text’ would return the ‘Married’ string/value on line 196 of ‘resources/EpicSandboxPatient.txt’.

Properties with array values

In some cases the value of a property is an array. Since a single PDI column can only contain one value, you will need to decide on a strategy for selecting the element you want. The two methods are providing an index like ‘[**0**]’ or putting the name of a ***forEachObj*** like ‘findOfficialName’.

Using index to retrieve an array value

cwFhirPath uses a zero-based index value enclosed in brackets to specify an element. For example, the path name[**0**]given[**0**] on line 76 in ‘sources/exp_client.cwfp’ would find the value “Us” on line 133 in ‘resources/EpicSandboxPatient.txt’. Notice that the elements in the ‘name’ array are JSON objects and ‘given’ holds a JSON array of strings.

forEachObj funcName {

forEachObj is a function you create to find an element or elements in an array of JSON objects; if ***found*** is invoked, the function will use the object in the rest of the path that uses it. The ***forEachObj*** function must be declared in the file before it is used in a ***findResult*** path.

funcName is the name you can use in your ***findResult*** path(s)
{ is the start of the function block.

For example, the ***forEachObj*** function named ‘findFirstOfficialName’ on line 6 in ‘sources/exp_client.cwfp’ looks for the property ‘use’ to see if its value is ‘official', and if so uses the keyword ***found*** to indicate that this is the object to use in an array of objects. An example of using ‘findFirstOfficialName’ to use the ‘given’ value of the ***found*** object in a path can is on line 42. In the example ‘resources/EpicSandboxPatient.txt you can follow the FHIR Patient resource path ‘name[findFirstOfficialName]family’. The property ‘name’ on line 127 holds an array of three JSON objects, each containing a ‘use’ property with different values. You can see that the second JSON object on lines 136 through 143 is what ‘findFirstOfficialName’ would find, and its ‘family’ property on line 139 has a value ‘Lin’ in the property ‘lastName’ on line 42 of ‘sources/exp_client.cwfp’.

Note that if multiple objects in the array are found, the first found object will be used, unless the column name has ‘**#**’ in it–see ‘column names with #’ in this documentation.

if booleanOperation {

if is a keyword you can use to see if a path was found and use boolean operators to compare values. The cwFhirPath language does not use the ‘and’ or ‘or’ operators, nor

does it use ‘else’, ‘(‘ or ‘)’. Operators ‘=’ and ***contains*** only will compare string values and ***not*** can only be placed immediately after the ***if***.

booleanOperation represents the ways you can apply logic to a ***findResult*** variable.

Using if with varName.Found

The **varName** ‘.Found’ property is set to true in a ***findResult*** variable if the last property in the path was found. For example, on line 44 of ‘sources/exp_client.cwfp’, if the ***findResult*** path finds a value then ‘codingsys.Found’ will be set to true and ‘*if codingsys.Found {*‘ will execute the code inside its block. If the path did not find the desired property, ‘codingsys.Found’ would be false and the code block between lines 45 and 47 would be skipped.

Using *if* with varName.StringValue and varName.StringValue.ToLower and ‘=’

There are cases where you need to check the value of a ***findResult*** variable before executing a block of code. For example, on line 11 of ‘sources/exp_client.cwfp’ the statement ‘*if useType.StringValue.ToLower = 'official' {*‘ the ‘.StringValue’ of variable ‘useType’ is converted to lowercase before checking if it equals the hardcoded string ‘official’.

Note that ‘.Found’, ‘.StringValue’, and ‘.StringValue.ToLower’ are the only available properties that can be used with a ***findResult*** variable.

Using *if* with varName.StringValue and varName.StringValue.ToLower and ‘contains’

contains looks in a ***findResult*** variable and only returns true if the value contains a substring. For example, on line 63 of ‘sources/exp_client.cwfp’ the statement ‘*if codingsys.StringValue.ToLower contains 'us-core-birthsex' {*‘ the ‘.StringValue’ of ‘codingsys’ is converted to lowercase before checking if the value (in this case a URL) contains the substring ‘*us-core-birthsex*’.

Using *if* with *not*

At the time this document was written, we have not yet found a need for ***not***, but it can be used to reverse the Boolean value produced by the rest of the ***if*** statement. The placement of ***not*** can only be after the ***if*** and before the rest of the expression.

rows {

rows is a function to convert a FHIR Bundle into PDI rows by scanning the ‘entry’ array and invoking the code in another *.cwfb file for each element in the array. The ***rows*** function must be the only function in its own *.cwfb file, and it expects a FHIR Bundle Resource type to be what its FHIR PDI Resource query returns. For example, on line 5

of ‘medication_request_bundle.cwfp’, the ***rows*** function iterates through a FHIR Bundle of MedicationRequest resource objects stored in the ‘entry’ array on line 11 of ‘EpicSandboxMedicationRequestBundle.txt’.

rowArrayPath pathToRowObjs

> **rowArrayPath**
> **pathToRowObjs** for a FHIR Bundle you will want this to be ‘entry’.

rowCwFhirPath filename.cwfp

rowCwFhirPath specifies which *.cwfp file to use to convert each element in ***rowArrayPath***. **filename.cwfp** is the name of the cwFhirPath source file that can make PDI rows from elements in the FHIR Bundle ‘entry’ array. For example, in line 7 in ‘medicationOrders.cwfp’, the filename containing this source is ‘medicationOrder.cwfp’. If you look at line 77 in ‘medicationOrder.cwfp’, you will see that the base property is ‘resource’ for all *column* functions in the file. If you look on line 12 of ‘EpicSandboxMedicationRequestBundle.txt’, you can see the element is not a ‘MedicationRequest’ itself but an entry object with a ‘resource’ property (line 20) that contains a ‘MedicationRequest’.

Automatic .reference resource fetching in findResult paths

In line 77 in ‘medicationOrder.cwfp’ after the property ‘subject’, the ‘reference’ property is used, followed by the ‘name’ property. On line 58 of ‘EpicSandboxMedicationRequestBundle.txt’, the ‘subject’ property is an object that contains a ‘reference’ property (line 59) that rather than containing a Patient FHIR Resource, it contains an R4 Patient resource relative path that can be used to get a Patient resource like the one in ‘EpicSandboxPatient.txt’. cwFhirPath takes care of managing this query lookup behind-the-scenes. The referenced object can then be used like ‘name’ is on line 77 in ‘medicationOrder.cwfp’ as if the ‘reference’ property did contain the object itself.