GELLO R2 User Guide

1 Overview
2 Introduction
  2.1 GELLO
3 Getting Started
  3.1 Example 1 - very simple program
  3.2 Example 2 - some expressions
  3.3 Example 3 - using strings
  3.4 Example 4 - some real GELLO
  3.5 Wrap up

Overview

Note - this is a document currently undergoing new revisions.

You can also read **The GELLO R2 User Manual**

Introduction

The Medical-Objects implementation of GELLO was written as a response to the need for decision support within the range of HL7 products produced by Medical-Objects Pty Ltd. It was initially developed in 2006 and has been successfully integrated into Medical-Objects Explorer and other research related products. For more information on Medical-Objects products visit **www.medical-objects.com.au**

GELLO

GELLO is formally described as “An Object-Oriented Query and Expression Language for Clinical Decision Support”. In a nutshell, it is a language to write expressions and make queries on medical data. GELLO expressions are somewhat like mathematical formulas but include additional functions to query and organize the data that the expressions refer to.

The GELLO language has been targeted at a wide range of IT Health professionals, from IT savvy doctors to experienced IT specialists working with in the IT Health industry. It has been based in part on the OCL (Object Constraint Language) version 2.0 language developed by OMG (Object Management Group). In effect GELLO is the SQL for objects and supplies much the same functionality as SQL but operates over an object model rather than a database.

GELLO is an international standard maintained by the HL7 organization.

Getting Started

Let's learn about GELLO by working through some small examples.

Example 1 - very simple program

Here’s an example of a very simple GELLO program.

```
Let a: Integer = 1
Let b: Integer = a + 1
b
```

This program is composed of 3 lines. The first two lines comprise of Statements and the last line is a final GELLO expression.

GELLO statements are of several forms. Our example only contains Let statements. A Let statement defines a variable (in this example’s first statement, the variable ‘a’) and assigns a value to that variable.

The final GELLO expression signifies that the GELLO program should return whatever value the expression evaluates to.

In our example, the first let statement “Let a:integer = 1” defines the variable ‘a’ to be of type ‘integer’ and gives it the value ‘1’. (an integer is a whole number like 1, 2, 3, 4 and so forth and also includes negative numbers like -1, -5 and so forth). A type in GELLO refers to the kind of data that a variable can have. In this example, the type of the variable is “Integer”.

The second let statement defines the variable ‘b’ and assigns it the value of ‘a + 1’ (which turns out to be 1 + 1 = 2).

The final line of the program, the GELLO expression simply returns the value of ‘b’ which in this case happens to be 2.

Here is the output from running such a program…

```
A: Integer = 1
B: Integer = 2
Result is 2
```

Example 2 - some expressions

Here is a slightly more complicated GELLO example.

```
Let a: Integer = 50
Let b: Integer = (a*100 + 20) div 2
a + b
```

And for which the output is…

```
A: Integer = 50
B: Integer = 2510
Result is 2560
```

Example 3 - using strings

We can also work with expressions which are not numbers but rather are strings, which are simply a group of letters like words.

Here is an example using strings.

```
Let surname: String = 'Smith'
Let givenname: String = 'Fred'
givenname.concat(' ').concat(surname)
```

And the results are

```
SURNAME: String = Smith
GIVENNAME: String = Fred
Result is Fred Smith
```

This example produces a result by concatenating (or joining) the three strings givenname, ' ' and surname with the concat() operator producing the string 'Fred Smith'. Note that strings are not simply words, but can contain any character including spaces, however string values must always be surrounded by the ' character.

Example 4 - some real GELLO

Let's look at a more realistic example.

```
Context observations: Observations
Let sodiums:Observations = observations->select(code.name = "Sodium")
sodiums.value
```

This example introduces a new kind of statement, the context statement. This statement defines a variable much like the let statement, however, it is different in that instead of assigning a simple value, it assigns data to the variable from an external source. In this case, it defines a context variable of type Observation which will be populated by the caller with all the patient's observations. This data model will be typically supplied by the GELLO enbedded infrastructure.

The type Observation is predefined to be a collection (or list) of individual observations. Each of these observations will have predefined properties depending on the data model, which in our case will have at least the properties “code” and “value”. The codes of an observation will in turn have properties of “code” of type “CodedValue” (not to be confused with the observation’s “code”) and “name” which are both strings (groups of letters like words).

The next statement is somewhat familiar in that it is like the Let statement we saw earlier, however this time the type of the variable has been omitted. When written this way, the type of the variable is inferred from the expression used to create it. The expression assigned to the variable also looks a little unusual. You will notice after the variable name “Observation” is the symbol “->”. This is the query operator and it is used to process the variable in a particular way, and in this example, we are specifying that we want to select from the observation list all observations that have a code with name “Sodium”. We then finally assign the resulting new collection to the variable “sodiums” The final line in the program returns only the value properties of the variable “sodiums”.

You may have noticed that we have introduced two new concepts, properties and collections.

Variables can be defined with simple types like Integer, String, Real or Boolean (we’ll discuss Real and Boolean later), or they can be defined with more complex types which are aggregates of other variables called “properties”. We refer to the properties of a variable by using the “.” operator. In the example, there two places where we use “.”. The first is at the place “code.name” which is referring to the property name of the variable “code” (of type CodedValue). The second place is the reference “sodiums.value” which is a slightly more complicated use of the “.” operator.

The other new concept is that of collections. A collection is a group or list of values each representing the same type of data. In GELLO, these can be a “Set”, “Bag” or “Sequence”, but for this example, it does not matter greatly which kind of collection it is, just that the variable “observation” is a collection of individual observations and that “sodiums” is the resulting collection formed by filtering out just the items which have as their code the coded name of “Sodium”.

Let’s run the program to see what the results might look like. For this example we’ll assume that the test data has been loaded. If you are running this yourself, make sure your test data is loaded first, or you won’t see any results.

```
OBSERVATIONS: Bag(Observation) = 496 Elements
SODIUMS: Bag = Bag{137, 138, 142, 131, 140, 137, 139, 135, 136, 140}
Result is Sequence{137, 138, 142, 131, 140, 137, 139, 135, 136, 140}
```

Wrap up

We've seen a few brief examples of how GELLO might look and feel. Now it's time to go read the **The GELLO R2 User Manual**.