RuleML Query Generation GUI Documentation (v1.2)

Note: PatientSupporter also uses a calendar tool in order to pick dates. This was taken from http://www.rainforestnet.com and is not fully supported in this document.

Release notes:
August 6, 2010 - (v1.2) Due to substitutions no longer use in Prova for RR, according updates have been made.
July 9, 2010 – (v1.1) Simple error checking has been added to JavaScript.
June 18, 2010 – (v1.0) Original document created.

This documentation explains the 6 parts of the GUI used in PatientSupporter to easily create a wide variety of different RuleML queries. It is built upon HTML using JavaScript. Sections 1-5 details the several different sections of the JavaScript file gui.js. Section 6 details the HTML side of the GUI. The document concludes with recommendations and some troubleshooting guidelines.

Files Involved

/files/gui.js (JavaScript file) – This contains all the JavaScript functions required for the GUI. Comments in the file are used to split the file into 5 separate sections which are detailed below.

RuleResponder.html (HTML file) – This is the file that contains the html form elements referenced in the gui.js file. The GUI itself is accessible by visiting this page. Information regarding this file can be found in section 6.

/files/datetimepicker_css.js (JavaScript file) – A CSS based calendar tool found at http://www.rainforestnet.com. How to use the calendar tool is not mentioned in this document; however the referenced website contains many examples and pieces of documentation in some of the code itself.

/files/layout.css – A Cascade Style Sheet used for PatientSupporter. The only thing of interest here is that during error checking, textbox's change their style template to show there is an error or change back when it is corrected. The styles used are listed in this file.

Section 1: JavaScript Preliminary

In this section, the global variables are declared. For each variable in the desired RuleML query, there are two variables. One variable is for the RuleML query creation, and one is the English representation for the English description creation (which is optional, but great for end-user comprehension).

For the ProfileID variable used in the RuleML query, we create the JavaScript variables, *profileID* and *englishProfileID*. *profileID* will eventually contain the section of the RuleML query that contains ProfileID information. *englishProfileID* will contain a string that represents the contents of ProfileID that could be used in a sentence. If the user is querying for any ProfileID, the corresponding in this situation could be the following:

profileID = “<Var>ProfileID</Var>”
englishProfileID = “any profile”

The variable *messageHeader* contains a large string of all the static information which comes before the variables in the RuleML query. *messageFooter* contains all the static information that comes after the variables.

Finally, any lower level functions required for the operation of the functions in the next section are placed in the preliminary section. In PatientSupporter, there are several functions used, each one described in commenting of *gui.js*.

Section 2: JavaScript Variable Functions

In this section, each RuleML query variable has a function associated with it. The naming scheme in this section is: *elementSelected***NameOfVariable***()*. These functions are run when the RuleResponder.html page is opened or whenever the function’s corresponding html form element is edited. Based on the user’s input in the HTML form, the two variables (English description and RuleML variable) are given the proper values.

For example for ProfileID, the function name is *elementSelectedProfileID()*. It first uses a local variable (usually called *choice*) to store the value of the index number of the element selected in the html dropdown box. It then uses *padZeros()* from Section 1, to pad index value to have the correct number of 0’s to be a profileID value. If the index value is equal to zero that implies “any” was selected in the html form. In RuleML, this would be a free variable so the global variable profileID stores the value “<Var>ProfileID</Var>”. Otherwise, profileID becomes a constraint which would take the form “<Ind>p(**padded index number here)**</Ind>”.

Form elements that require user input have an additional error function that is called in the corresponding variable function. This is described in Section 3. At the end of all variable functions, the *createRuleML()* function is called, which regenerates the RuleML query with the new values, and also calls the function *createEnglish()* to regenerate the English description as well. See section 4 for more details.

Section 3: JavaScript Error Checking

When dealing with HTML form elements that require text input from the user, the chance of user error goes up significantly. PatientSupporter currently implements simple error checking that informs the user of their mistakes by simply changing the color of the text field that contains error, to red. When the error is removed, the text field returns to its original color.

Each form element that requires error checking has an error checking function which is called right before *createRuleML()* in its corresponding variable function (see Section 2). Some variables require the same error checking function as other variables because they are related. Because of this, the naming convention of these error functions is *elementError<TopicName>Check().* An example variables requiring the same error function is the startDate, startTime, endDate, endTime variables. While these variables could have no

standalone errors, together they may have an error. For example, if the user put endDate and endTime occurring before startDate and startTime.

Section 4: JavaScript Query and English Generation

In this section of the code, RuleML query and English generation is done. In *createRuleML()*, all the RuleML global variables (messageHeader, profileID…..messageFooter) are put together as one large string and placed in the RuleML query box. *createEnglish()* is then called which combines all the English variables into a paragraph describing the query. The sentence structure is mostly up to the person creating the GUI. In line if statements shown in Section 4 of the code

Section 5: JavaScript Starter

The startCheck function runs when the web page is loaded (see the body tag in html) and when example queries are used. It runs every form element function to generate a RuleML query.

Section 6: HTML Forms

As stated in Section 2, each form element function is tied to an HTML form element. That is how the values are generated and how updates to the RuleML query start. Each form element's type relies on the type of information that is required. PatientSupporter generally uses the following:

Selection(Drop down boxes): This is used when there is a set amount of values for the variable. The corresponding form element function is triggered when a new selection is made.

Text box: This is used for inputting numerical values. The corresponding form element function linked to this type of form element is triggered whenever a key is pressed.

More information regarding various html form object can be found on the W3C site: http://www.w3.org/TR/html401/interact/forms.html

Recommendations

If you decide to modify and use this GUI, it is suggested while you are making your modifications, you keep a very simple barebones html file containing only the required elements for the GUI. This is to insure that while you're modifying the code, no bugs appear that are not related to the GUI. After you are satisfied with the GUI, proceed to integrating it into the actual webpage that it will be accessible from.

PatientSupporter currently uses regular expressions for pattern matching in several places to avoid faulty values and does alert the user when an incorrect value is inputted (the box with the incorrect value turns red). However, further error checking could be used to prevent more kinds of errors in the queries. The large issue with error checking multiple values such as StartTime, StartDate, EndTime and EndDate, is that some of these may be free variables while others are not.

The GUI has had some rapid changes in order for changes in PatientSupporter to be represented accordingly. Further simplification of the code could probably be done which would help with the modularization of the code as.

In the future, the ability for the GUI to work “both ways” would prove very useful. This would mean the user could edit the RuleML manually and the user’s changes would appear in the GUI menu. This could be accomplished by having the variables in a separate textbox from messageHeader and messageFooter, and then doing some parsing. Of course then the submission process would have to work differently as well.

Troubleshooting

Blank Result from Rule Responder

Examine the query in the address bar and ensure nothing extra is being added on or removed from the query. Some web browsers have built in address bar character limits (which does not meet the standard for HTML submission forms!).

Blank RuleML Query

This can be caused by any kind of coding mistake; however it was mostly found this occurred when a single string had a syntax error.

Undefined Values in RuleML Query

Most often it is found that the HTML form elements have been given the incorrect or duplicate name.

Debugging Tips

-Using an IDE that supports JavaScript is very helpful for debugging.

-Having a web browser equipped with a Java console is very useful for understanding script failures. Google Chrome is especially useful for this.

-Using the JavaScript alert(“*a message here*”) is useful for debugging conditional statements.