Oracle® Health Sciences Clinical One Platform

Rules Developer Guide

22.2
F54972-03
June 2022

ORACLE

Oracle Health Sciences Clinical One Platform Rules Developer Guide, 22.2

F54972-03

Contents

3 Rules examples

4 Revision history

Preface

This preface contains the following sections:

Documentation accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related resources

All documentation and other supporting materials are available on the Oracle Help Center.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through Support Cloud.

Contact our Oracle Customer Support Services team by logging requests in one of the following locations:

- English interface of Oracle Health Sciences Customer Support Portal (https://hsgbu.custhelp.com/)
- Japanese interface of Oracle Health Sciences Customer Support Portal (https://hsgbu-jp.custhelp.com/)

You can also call our 24x7 help desk. For information, visit http://www.oracle.com/us/support/contact/health-sciences-cloud-support/index.html or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

1

Before you begin your rules development

- JavaScript basics
  Before working with rules in Oracle Clinical One Platform, you should have a basic understanding of JavaScript. While you do not need advanced programming skills, an understanding of JavaScript functions and variables is critical to your success.
- Javascript usage tips
  While Oracle Clinical One Platform uses Javascript as a programming language for rules, there are some usage caveats and limitations that you should know before beginning your rules development, especially if you are an experienced Javascript developer.
- Predefined rules versus custom rules
  Predefined rules are included as part of Oracle Clinical One Platform, Study designers can apply these rules to questions during the study design process. Custom rules are more advanced rules, typically created by a rules designer, that apply more complex validation, or determine actions in response to certain criteria.

JavaScript basics

Before working with rules in Oracle Clinical One Platform, you should have a basic understanding of JavaScript. While you do not need advanced programming skills, an understanding of JavaScript functions and variables is critical to your success.

JavaScript is a widely-used programming language most often used in web development. This language is straightforward to learn and can be used to develop both basic expressions and those with more complex logic. We use this language to write custom rules in Oracle Clinical One Platform. Within the rule expressions, we can invoke JavaScript functions (blocks of code) and the provided helper functions, together with variables, constants, operators, and various methods to accomplish specific tasks.

Teaching you the fundamentals of JavaScript is beyond the scope of this document. There are many excellent resources available on the web (such as W3schools) that can help you understand JavaScript concepts and basic programming methodology.

JavaScript functions

Functions are simply blocks of code designed to accomplish a specific task. You can use functions in your rules to perform a variety of tasks. When writing your rules, you invoke these functions from the rules interface. You can use native JavaScript functions or special helper functions that are provided as part of Oracle Clinical One Platform.

- Native JavaScript functions are functions that are part of standard JavaScript programming. These functions are familiar to JavaScript programmers and not specific to Oracle Clinical One Platform. You can invoke your code with parameters and the blocks of code you write are visible within the rule expression.

  Be sure to understand the limitations and usage caveats for using JavasScript in Oracle Clinical One Platform before programming your rules. For guidance, see Javascript usage tips.

- Oracle helper functions are also invoked using specific parameters and a value is returned by the helper function. The return value for a helper function is available to you for use in your rule expression and the logic of your rule can use this value to perform an action. However, the code within these functions is not visible to you in the rules interface. Refer to the Rules helper function reference for details on each function.

JavaScript variables

When working with JavaScript functions, you use variables to pass your data. Variables are simply containers used to hold values that you want to use in your rule. You must declare and define variables as part of your rule.

> **Note:**
>
> When writing rules, you must consider two types of variables, ones that hold values taken directly from the data entered into forms and ones that can be created within your rule code to store values generated within the code. Form variables are defined in the top portion of the rule editor and are populated by form values. Variables used in your code are defined in the code itself.

Rule validation

The rules interface provides a simple way for you to add your rule code and includes syntax validation. However, you should keep in mind that this validation is used to ensure that the syntax used in your rule is valid JavaScript (that is, you have not made a coding error such as forgetting a closing parentheses or required semi-colon).

Understanding and defining the logic you want for your rule is a difficult process regardless of your level of JavaScript knowledge! The system does not provide verification of your rule logic so it is critical that you verify your rule and ensure the rule is functioning as expected. To help you with the creation of your rule logic, we provide a number of examples as part of our helper function reference and a library of Rules examples. Reviewing these examples can help you gain a better understanding of the logic and functions used when writing JavaScript rule expressions for complex tasks. These examples can also be used as a base for your own custom code. This can significantly decrease your development time.

> **Note:**
>
> Rule verification should always be done in Testing mode for your study before it is deployed to Production. You must always verify the behavior of your rule at run time.

Javascript usage tips

While Oracle Clinical One Platform uses Javascript as a programming language for rules, there are some usage caveats and limitations that you should know before

beginning your rules development, especially if you are an experienced Javascript developer.

Rule processing caveats

Follow these guidelines to help your rules process efficiently:

- Use the documented helper functions to reduce your need to loop through repeating instances when performing certain matching and compare operations on data. This improves your rule performance.
- Use generic Javascript functions under ECMAScript 5. For example, you can process an array of elements with `filter( )`, `reduce( )`, and so forth to loop through an array for a specific purpose. This can simplify your coding.

Javascript limitations

The following common Javascript and HTML coding operations are not allowed in any rule expressions:

- Console operations
- Print operations
- File operations (such as `load( )` and `open( )`)
- DOM manipulations (such as `document` and `window`)
- Display messages (such as `alert`)
- Interrupting script processing (such as `exit( )` and `quit( )`)
- Debugger commands
- Looping operations (such as `for` and `while`)

Predefined rules versus custom rules

Predefined rules are included as part of Oracle Clinical One Platform, Study designers can apply these rules to questions during the study design process. Custom rules are more advanced rules, typically created by a rules designer, that apply more complex validation, or determine actions in response to certain criteria.

Predefined rules can be selected during study design and are often (but not always) used for validating data. Study designers apply these rules to a specific question in a specific form to help facilitate appropriate data entry in a study. These rules allow study designers to better control the quality of the data that is collected by generating a validation error message every time the condition set by the rule isn't met. However, these rules are only available for a limited number of commonly encountered conditions.

Custom rules allow queries to be displayed for questions that have been answered and when specific conditions are met for the data that is entered. Custom rules can also be used to calculate the values of read-only questions or determine other actions such as sending an email notification. These rules are written using Javascript and require some knowledge of basic coding principles. Oracle Clinical One Platform provides an efficient interface where you can create, validate, and publish rules using JavaScript.

Note:

When using predefined rules in your study design, you should be aware that queries raised by predefined validation rules must be resolved before subjects can be screened or randomized. They can also prevent dispensation actions. However, when using custom rules, the system allows subjects to be screened or randomized even when the query is not resolved. Apply predefined validation rules carefully. Always validate your custom rules in Testing mode to ensure you are getting the expected results before approving the rule for use in a Production study.

2

Rules helper function reference

General expressions

Use basic JavaScript expressions in your rules to perform a variety of tasks and validations.

- Range check
  Ensure that values don't exceed a specific range. For example, when you want to check that the temperature is between 97.8°F to 99.1°F.

Comparison

Use this expression when you want to compare two variables and generate a query when the result returns **false**.

> **Note:**
>
> You can use any valid JavaScript comparison operators. The example shown is just for illustration purposes.

```
if (a>b){
  return false;
}
```

Conversion

Automatically convert values such as Fahrenheit into Celsius or Celsius into Fahreinheit degrees.

Example 2-1 Celsius to Fahrenheit conversion

```
//given temperature in C, convert to F and return converted value

var fahrenheit;
fahrenheit = (celsius * (9/5)) + 32;
return Number(fahrenheit);
```

Example 2-2 Celsius to Fahrenheit conversion (Simplified) using inline calculation

```
//given temperature in C, returning converted value to F using inline
calculation

return (tempc * (9/5)) + 32;
```

Example 2-3 Fahrenheit to Celsius conversion using inline calculation

```
//given temperature in F, returning converted value to C using inline
calculation

return (tempf - 32) * 5/9);
```

Switch statement

Use this expression when you want to test multiple conditions. For instance, when you want to calculate the number of kits to be dispensed based on the weight of the subject.

```
var kit;
  switch (true) {
    case weight > 25: kit = 5; break;
    case weight > 20: kit = 4; break;
    case weight > 15: kit = 3; break;
    case weight > 10: kit = 2; break;
    case weight >= 1: kit = 1; break;
  }
return Number(kit);
```

Choice

Search for one of the answers in a choice question.

> **Note:**
>
> You can also consider the getStringFromChoice( ) helper function as an alternative, depending on your use case.

This example illustrates how to view female subjects that haven't done a pregnancy test.

```
if ((Gender.search('Female') > 0) && (Test.search('Not Done') > 0)){
  return false;
}
```

> **Note:**
>
> The `.search` function can only be used with a string. If you want to use this function with an object, you must first convert the object to a string.

Compare dates with different formats

Compare two dates with different formats. This can help confirm or resolve date inconsistencies.

> **Note:**
>
> There are also several helper functions available for date comparisons. These functions may reduce your coding effort. For details, see Date and time functions.

For example, the Date Performed (Date format) on the Physical exam form should be the same as the Check In date (date and time format) on the Housing form but they are different. You can create a rule to confirm that the Physical Exam was not done at the same time as the Check In or resolve the inconsistent dates.

```
var ci_dt = ci_datetime.getDate() + "-" + (ci_datetime.getMonth() + 1)
+ "-" + ci_datetime.getFullYear();
var pe_dt = pe_date.getDate() + "-" + (pe_date.getMonth() + 1) + "-" +
pe_date.getFullYear();
 if (ci_dt !== pe_dt){
  return false;
}
```

Range check

Ensure that values don't exceed a specific range. For example, when you want to check that the temperature is between 97.8°F to 99.1°F.

In this example, we are looking for temperatures outside of a certain range when respiration is abnormal.

```
var upper = 99.1;
var lower = 97.8;
if ((temp_res.search('Abnormal') > 0) && (temp_f > lower && temp_f <
upper)){
  return false;
}
```

Date and time functions

Compare and manipulate date and time values.

> **Note:**
>
> For date-related functions, partial dates are not supported except where they are explicitly mentioned.

- dateDiffInYears( )
  Calculate the difference between two dates in years. For instance, when you have two questions of type Date and Time in a form.
- dateDiffInDays( )
  Calculate the date difference between two dates measured in days.
- timeDiffInHours( )
  Calculate the time difference between two DateTime values measured in hours.
- timeDiffInMinutes( )
  Compare variables of type DateTime with time elements that do not contain partial dates.
- timeDiffInSeconds( )
  Calculate the time difference between two Date/Time values measured in seconds.
- areDatesEqual( )
  Compare two dates to determine if they are equivalent.
- isDateInRange( )
  Verify if a date falls within a defined range.
- areDateTimesEqual( )
  Compare two Date and Time values to determine if they are equivalent.
- isTimeInRange( )
  Verify if a Date and Time falls within a defined range.
- addDays( )
  Add a specific number of days to a date value. For example, when you need to ensure the Date and Time entered by the user doesn't exceed a specific value.
- addTimeInHours( )
  Add a specific number of hours to a Date and Time value. For example, when you need to ensure the Date and Time entered by the user doesn't exceed a specific value.
- addTimeInMinutes( )
  Add a specific number of minutes to a Date and Time value. For example, when you need to ensure the Date and Time entered by the user doesn't exceed a specific value.
- getDateDMYFormat( )
  Return a date in DD-Mon-YYYY format, including partial dates.
- getDatesCompareResult( )
  Compare two dates using a provided operation. This function handles partial dates.
- partialDateDiff( )
  Find the difference between two dates.

dateDiffInYears( )

Calculate the difference between two dates in years. For instance, when you have two questions of type Date and Time in a form.

The `dateDiffInYears( )` helper function is invoked with `toDate` and `fromDate` passed in as parameters. The function returns a negative or positive number value indicating the difference between the two dates in years. If the number value returned is a negative or zero value, `toDate` is before or the same as `fromDate`. If the function returns a positive value, `toDate` is after `fromDate`.

> **Note:**
>
> This function requires two JavaScript **Date** objects as input. These objects can include both date and time.

Syntax

```
dateDiffInYears(toDate, fromDate)
```

> **Note:**
>
> You must compare dates that have the same format.

Parameters

toDate
Date value.

fromDate
Date value.

Return value

Difference between the dates in years (number). This number can be positive or negative.

> **Note:**
>
> The order in which parameters are supplied for date helper functions is important; the resulting return value depends on which date you pass in as the first or second parameter.

Example 2-4 Difference between two Date items

```
// Given 2 form questions of type DateTime are defined in the rule as
variables:
return dateDiffInYears(dateItem1, dateItem2);
```

Example 2-5 Difference between two hard-coded dates

```
var toDate = new Date("March 1, 2020");
var fromDate = new Date("March 1, 2000");
return dateDiffInYears(toDate, fromDate);

// Returns value: 20
```

dateDiffInDays( )

Calculate the date difference between two dates measured in days.

> **Note:**
>
> This function is only used to compare variables of type DateTime that do not contain time elements and do not include partial dates. You can use the timeDiffInMinutes( ) helper function to compare two date and time items. If the date question contains partial date elements then use the getDatesCompareResult( ) helper function.

The `dateDiffInDays( )` helper function is invoked with `toDate` and `fromDate` passed in as parameters. The function returns a negative or positive number value indicating the difference between the two dates in days.

If the number value returned is a negative or zero value, `toDate` is before or the same as `fromDate`. If the function returns a positive value, `toDate` is after `fromDate`.

Syntax

```
dateDiffInDays(toDate, fromDate)
```

Parameters

toDate
Date value.

fromDate
Date value.

Return value

Difference between the dates in days (number). This number can be positive or negative.

> **Note:**
>
> The order in which parameters are supplied for date helper functions is important; the resulting return value depends on which date you pass in as the first or second parameter.

Example 2-6 Difference between two Date items

```
// Given 2 form questions of type DateTime are defined in the rule as
variables:
return dateDiffInDays(datetem1, dateItem2);
```

Example 2-7 Difference between two hard-coded dates

```
var toDate = new Date("March 1, 2020");
var fromDate = new Date("March 1, 2019");
return dateDiffInDays(toDate, fromDate);

// Returns value: 366 (leap year!)
```

timeDiffInHours( )

Calculate the time difference between two DateTime values measured in hours.

The `timeDiffInHours( )` helper function is invoked with `toDate` and `fromDate` passed in as parameters. The function returns a negative or positive number value indicating the difference between the two dates in hours. If the number value returned is a negative or zero value, `toDate` is before or the same as `fromDate`. If the function returns a positive value, `toDate` is after `fromDate`.

Syntax

```
timeDiffInHours(toDate, fromDate)
```

Parameters

toDate
DateTime value.

fromDate
DateTime value.

Return value

Difference between the dates in hours (number). This number can be positive or negative.

> **Note:**
>
> The order in which parameters are supplied for date helper functions is important; the resulting return value depends on which date you pass in as the first or second parameter.

Example 2-8 Difference between two DateTime items

```
// Given 2 form questions of type DateTime are defined in the rule as
variables:
return timeDiffInHours(dateTime1, dateTime2);
```

Example 2-9 Difference between two hard-coded DateTime items

```
var toDate = new Date("March 1, 2020 13:00:00");
var fromDate = new Date("March 1, 2020 12:00:00");
return timeDiffInHours(toDate, fromDate);

// Returns value: 1
```

timeDiffInMinutes( )

Compare variables of type DateTime with time elements that do not contain partial dates.

> **Note:**
>
> If your date questions do not contain time elements then use dateDiffInDays( ) helper function. If the date questions contain partial date elements, use getDatesCompareResult( ).

The `timeDiffInMinutes( )` helper function is invoked with `toDate` and `fromDate` passed in as parameters. The function returns a negative or positive number value indicating the difference between the two dates in minutes. If the number value returned is a negative or zero value, `toDate` is before or the same as `fromDate`. If the function returns a positive value, `toDate` is after `fromDate`.

Syntax

```
timeDiffInMinutes(toDate, fromDate)
```

Parameters

toDate
DateTime value.

fromDate
DateTime value.

Return value

Difference between the dates in minutes (number). This number can be positive or negative.

> **Note:**
>
> The order in which parameters are supplied for date helper functions is important; the resulting return value depends on which date you pass in as the first or second parameter.

Example 2-10 Difference between two DateTime items

```
// Given 2 form questions of type DateTime are defined in the rule as
variables:
return timeDiffInMinutes(dateTime1, dateTime2);
```

Example 2-11 Difference between two hard-coded DateTime items

```
var toDate = new Date("March 1, 2020 12:02:00");
var fromDate = new Date("March 1, 2020 12:00:00");
return timeDiffInMinutes(toDate, fromDate);

// Returns value: 2
```

timeDiffInSeconds( )

Calculate the time difference between two Date/Time values measured in seconds.

The `timeDiffInSeconds( )` helper function is invoked with `toDate` and `fromDate` passed in as parameters. The function returns a negative or positive number value indicating the difference between the two dates in days. If the number value returned is a negative or zero value, `toDate` is before or the same as `fromDate`. If the function returns a positive value, `toDate` is after `fromDate`.

Syntax

```
timeDiffInSeconds(toDate, fromDate)
```

Parameters

toDate
DateTime value.

fromDate
DateTime value.

Return value

Difference between the dates in seconds (number). This number can be positive or negative.

> **Note:**
>
> The order in which parameters are supplied for date helper functions is important; the resulting return value depends on which date you pass in as the first or second parameter.

Example 2-12 Difference between two DateTime items

```
// Given 2 form questions of type DateTime are defined in the rule as
variables:
return timeDiffInSeconds(dateTime1, dateTime2);
```

Example 2-13 Difference between two hard-coded DateTime items

```
var toDate = new Date("March 1, 2020 12:02:00");
var fromDate = new Date("March 1, 2020 12:00:00");
return timeDiffInSeconds(toDate, fromDate);

// Returns value: 120
```

areDatesEqual( )

Compare two dates to determine if they are equivalent.

Syntax

```
areDatesEqual(date1, date2)
```

Parameters

date1
Date value.

date2
Date value.

Return value

True, if dates are equal; **false**, if they are not.

Example 2-14 Compare two Date items

```
// Given 2 form questions of type DateTime are defined in the rule as
variables
if (areDatesEqual(date1, date2)) {
  return false;
```

```
} else {
  return true;
}

// Triggers a query if this is a query rule and dates are equal.
```

Example 2-15 Compare two hard-coded Date items

```
var date1 = new Date("March 20, 2020");
var date2 = new Date("March 1, 2020");
if (!areDatesEqual(date1, date2)) {
  return false;
} else {
  return true;
}

// Triggers a query if this is a query rule and dates are NOT equal.
```

isDateInRange( )

Verify if a date falls within a defined range.

Syntax

```
isDateInRange(dateToCheck, dateFrom, dateTo, inclusive)
```

Parameters

dateToCheck
Date value to check.

dateFrom
Date value range start.

dateTo
Date value range end.

inclusive
String: **both**, **from**, **to**, or **no**.

- **both**: include `dateTo` and `dateFrom` dates in the range check (`dateFrom <= dateToCheck && dateToCheck <= dateTo`)
- **from**: include only `dateFrom` in the range check (`dateFrom <= dateToCheck && dateToCheck < dateTo`)
- **to**: include only `dateTo` in the range check (`dateFrom < dateToCheck && dateToCheck <= dateTo`)
- **no**: don't include `dateTo` or `dateFrom` in the range check (`dateFrom < dateToCheck && dateToCheck < dateTo`)

Return value

true, if date is within range; **false**, if it is not.

Example 2-16 Check a Date value

```
// Given 3 form questions of type DateTime are defined in the rule as
variables
if (isDateInRange(dateToCheck, dateFrom, dateTo, "both")) {
  return true;
} else {
  return false;
}

// Triggers query if dateToCheck is not in range (dateFrom <= dateToCheck &&
dateToCheck <= dateTo)
```

Example 2-17 Compare three hard-coded dates

```
var dateToCheck = new Date("April 1, 2020");
var dateFrom = new Date("March 1, 2020");
var dateTo = new Date("March 30, 2020");

if (!isDateInRange(dateToCheck, dateFrom, dateTo, "both") {
  return false;
} else {
  return true;
}

//Triggers query since dateToCheck is not in range (dateFrom <= dateToCheck
&& dateToCheck <= dateTo)
```

areDateTimesEqual( )

Compare two Date and Time values to determine if they are equivalent.

Syntax

```
areDateTimesEqual(date1, date2)
```

Parameters

date1
DateTime value.

date2
DateTime value.

Return value

true, if dates are equal; **false**, if they are not.

Example 2-18 Compare two DateTime items

```
// Given 2 form questions of type DateTime are defined in the rule as
variables
if (areDateTimesEqual(date1, date2)) {
```

```
  return false;
} else {
  return true;
}

// Triggers query if dates are equal.
```

Example 2-19 Compare two hard-coded DateTime items

```
var date1 = new Date("March 1, 2020 13:00:00");
var date2 = new Date("March 1, 2020 12:00:00");
if (!areDateTimesEqual(date1, date2)) {
  return false;
} else {
  return true;
}

// Triggers query since dates are not equal.
```

isTimeInRange( )

Verify if a Date and Time falls within a defined range.

Syntax

```
isTimeInRange(dateToCheck, dateFrom, dateTo, inclusive)
```

Parameters

dateToCheck
DateTime value to check.

dateFrom
DateTime value range start.

dateTo
DateTime value range end.

inclusive
String: **both**, **from**, **to**, or **no**.

- **both**: include `dateTo` and `dateFrom` dates in the range check (`dateFrom <= dateToCheck && dateToCheck <= dateTo`)
- **from**: include only `dateFrom` in the range check (`dateFrom <= dateToCheck && dateToCheck < dateTo`)
- **to**: include only `dateTo` in the range check (`dateFrom < dateToCheck && dateToCheck <= dateTo`)
- **no**: don't include `dateTo` or `dateFrom` in the range check (`dateFrom < dateToCheck && dateToCheck < dateTo`)

Return value

true, if DateTime is within range; **false**, if it is not.

Example 2-20 Check a DateTime value

```
// Given 3 form questions of type DateTime are defined in the rule as
variables
if (isTimeInRange(dateToCheck, dateFrom, dateTo, "both")) {
  return true;
} else {
  return false;
}

// Triggers query if dateToCheck is not in range (dateFrom <= dateToCheck &&
dateToCheck <= dateTo)
```

Example 2-21 Compare three hard-coded DateTime items

```
var dateToCheck = new Date("March 1, 2020 14:00:00");
var dateFrom = new Date("March 1, 2020 12:00:00");
var dateTo = new Date("March 1, 2020 13:00:00");

if (!isTimeInRange(dateToCheck, dateFrom, dateTo, "both") {
  return false;
} else {
  return true;
}

//Triggers query since dateToCheck is not in range (dateFrom <= dateToCheck
&& dateToCheck <= dateTo)
```

addDays( )

Add a specific number of days to a date value. For example, when you need to ensure the Date and Time entered by the user doesn't exceed a specific value.

Syntax

```
addDays(startDate, numberOfDays)
```

Parameters

startDate
Date value to check.

numberOfDays
Number of days to add to `startDate`.

Return value

A new date value increased by the number of days specified.

Example 2-22 date1 cannot be greater than 12 hours from date2

```
// Given 2 form questions of type Date are defined in the rule as
variables
if (addDays(dateTime1, 7) > dateTime2) {
  return false;
} else {
  return true;
}

// triggers query that dateTime1 cannot be > 7 days from dateTime2
```

addTimeInHours( )

Add a specific number of hours to a Date and Time value. For example, when you need to ensure the Date and Time entered by the user doesn't exceed a specific value.

Syntax

```
addTimeInHours(startTime, numberOfHours)
```

Parameters

startTime
Date value to check.

numberOfHours
Number of hours to add to `startTime`.

Return value

A new date value increased by the number of hours specified.

Example 2-23 dateTime1 cannot be greater than 12 hours from dateTime2

```
// Given 2 form questions of type DateTime are defined in the rule as
variables
if (addTimeInHours(dateTime1, 12) > dateTime2) {
  return false;
} else {
  return true;
}

// Triggers query that dateTime1 cannot be > 12 hours from dateTime2
```

addTimeInMinutes( )

Add a specific number of minutes to a Date and Time value. For example, when you need to ensure the Date and Time entered by the user doesn't exceed a specific value.

Syntax

```
addTimeInMinutes(startTime, numberOfMinutes)
```

Parameters

startTime
Date value to check.

numberOfMinutes
Number of minutes to add to `startTime`.

Return value

A new date value increased by the number of minutes specified.

Example 2-24 dateTime1 cannot be greater than 30 minutes from dateTime2

```
// Given 2 form questions of type DateTime are defined in the rule as
variables
if (addTimeInMinutes(dateTime1, 30) > dateTime2) {
  return false;
} else {
  return true;
}

// Trigges query that dateTime1 cannot be > 30 minutes from dateTime2
```

getDateDMYFormat( )

Return a date in DD-Mon-YYYY format, including partial dates.

Syntax

```
getDateDMYFormat(vDate, isPartial)
```

Parameters

vDate
Variable with date type.

isPartial
If the variable `vDate` is partial or not (**true/false**).

Return value

Date in DD-Mon-YYYY, for example **UNK-Jan-2025** or **01-Feb-2021**.

Example 2-25 Return a partial date in DD-Mon-YYYY format

```
var mdyDate = getDateDMYFormat(vDate, true);
return mdyDate.toString();


// returns date as UNK-Jan-2025
```

getDatesCompareResult( )

Compare two dates using a provided operation. This function handles partial dates.

> **Note:**
>
> This function is only valid when one of the dates used in the comparison is a partial date.

Syntax

```
getDatesCompareResult(date1,isPartial1,date2,isPartial2,operation)
```

Parameters

date1
Variable with date type.

isPartial1
Indicates if the `date1` variable is partial or not (**true**/**false**).

date2
Variable with date type.

isPartial2
Indicates if the `date2` variable is partial or not (**true**/**false**).

operation
The operation you want to use to compare date1 and date2. For example, `">"`, `">="`, `"<"`, `"<="`, `"==="`, or `"!=="`.

Return value

Returns **true** or **false**.

Example 2-26 Check if date1 is greater than date2

```
// check if date 1 is greater than date 2
return getDatesCompareResult(date1,true,date2,false,">");


// returns true or false
```

partialDateDiff( )

Find the difference between two dates.

Note: This is a JavaScript function. Quotes are not needed in the rule variable name.

Syntax

```
partialDateDiff(date1,isPartial1,date2,isPartial2,Datepart)
```

Parameters

date1
Variable with a date type.

isPartial1
If the variable `date1` is partial or not (**true** or **false**).

date2
Variable with a date type.

isPartial2
If the variable `date2` is partial or not (**true** or **false**).

Datepart
Day or Year.

Return value

Number that represents the difference between the two dates.

Note: In order for the rule to work, at least one of the DateTime parameters needs to be a partial date and one of the `isPartial` parameters needs to be **true**.

Example 2-27 Compare one full DateTime Item with one partial DateTime Item

```
// Given 2 form questions of type DateTime are defined in the rule as
variables
// date1 is a full date containing the value of 05-NOV-2021
// date2 is a partial date containing the value of UNK-OCT-2021
if(partialDateDiff(date1, false, date2, true, 'Day') > 28){
    returntrue;
}
else{
    returnfalse; // Query is triggered if the difference between dates is
greater than 28 days.
}

// The difference between dates is '31', the query will not be triggered
```

Repeating form functions

Find or evaluate a value in a repeating form.

- FindDuplicateRepeatingForm( )
  Detect duplicate data in a repeating form. The data is identified by a form ID which has duplicate item values for the search keys that are provided in each function argument evaluated as one key.
- FindDuplicateRepeatingFormWithinRange( )
  Detect duplicate data in a repeating form. The data is identified by form ID which has duplicate item values for the date ranges or search keys that are provided.
- FindMinInRepeatingForms( )
  Find the minimum value of a given data item in all instances of a repeating form. The data is retrieved by form ID and works only for numeric fields.
- FindMaxInRepeatingForms( )
  Find the maximum value of a given data item in all instances of a repeating form. The data is retrieved by form ID and works only for numeric fields.
- FindMinDateInRFs( )
  Find the minimum value of given date, date-time, or partial date items in all repeating instances of the form identified by the form ID. This function is only applicable to date fields.
- FindMaxDateInRFs( )
  Find the maximum value of the given date, date-time, or partial-date items in all of the repeating instances of the form identified by the form ID. This function is only applicable to date fields.
- FindMatchingRepeatingForm( )
  Find a repeating form instance that contains a value that matches the search value.
- FindMatchingRepeatingFormWithinRange( )
  Find an instance of a repeating form, identified by the form ID, that matches the item value provided as a search key. The search can be based on search keys or date ranges.
- FindRFInstance( )
  Find a repeating form instance that contains a value which matches the search value using a supplied operator.
- ListRFInstances( )
  List all instance numbers for a repeating form. You can use this helper function in your rule expression to check for instances of a specific question value in a repeating form.
- GetCurrentRFInstance( )
  Get the form instance number where the rule is currently being run.
- GetMatchingRepeatingFormsCount( )
  Get the number of repeating form instances of a form that match the item values provided as search keys.
- getRFValues( )
  Retrieves the current values for specified items on repeating form instances.

FindDuplicateRepeatingForm( )

Detect duplicate data in a repeating form. The data is identified by a form ID which has duplicate item values for the search keys that are provided in each function argument evaluated as one key.

For this function:

- You cannot use drop-downs, radio buttons, or checkbox values as function parameters or as a target.
- If a variable is designed to hold a partial date, you must provide the value for that parameter in the same partial date format.
- Deleted instances are not matched unless the helper functions provides a parameter to include deleted records.

Use as many arguments as needed to fully define the duplicate key.

> **Note:**
>
> This is an aggregation function. The rule will be run for each form instance in the case where the target is on a repeating form.

Syntax

```
FindDuplicateRepeatingForm(variable1, variable2,...)
```

Parameters

variable(s)
Item variables to check.

Return value

Returns **true** if duplicate values are found; **false** if duplicate values are not found.

Example 2-28 Check to see if any duplicate repeating form instances exist with the same values for Lab and Test Name

```
// Given 5 repeating form instances with items "Lab" and "Test Name"
if (FindDuplicateRepeatingForm(itmLab, itmTestName)) {
  return false;
} else {
  return true;
}


// Fires a query if more than 1 repeating form instance is found containing
Lab = "Mass General" and Test Name = "CBC"
```

FindDuplicateRepeatingFormWithinRange( )

Detect duplicate data in a repeating form. The data is identified by form ID which has duplicate item values for the date ranges or search keys that are provided.

For this function:

- You cannot use drop-downs, radio buttons, or checkbox values as function parameters or as a target.
- If a variable is designed to hold a partial date, you must provide the value for that parameter in the same partial date format.
- If a record has no data enter for the required dates, the record is skipped for comparison.

The first two parameters should always be the date range. Additional search keys can be provided.

> **Note:**
>
> This is an aggregation function. The rule will be run for each form instance in the case where the target is on a repeating form.

Syntax

```
FindDuplicateRepeatingFormWithinRange(startDateVariable,
endDateVariable, variable1,...)
```

Parameters

startDateVariable
Required. Date item on a repeating form.

endDateVariable
Required. Date item on a repeating form.

variable(s)
Optional. Additional Item variable to search.

Return value

Returns **true** if duplicate values are found; **false** if duplicate values are not found.

Example 2-29 Check to see if any repeating form instances exist within the same onset date range and symptom value

```
// Given 5 repeating form instances with items "onsetStartDate",
"onsetEndDate and "Symptom"
if (FindDuplicateRepeatingFormWithinRange(itmOnsetDateStart,
itmOnsetDateEnd, itmSymptom) === true) {
  return false;
} else {
```

```
  return true;
}

// Fires a query if more than 1 repeating form instance is found within the
same symptom value within the same date range.
//
// Note: If any repeating form instance has a null start or end date then
the system assumes a date in order to successfully
// perform the date range overlap check. If the start date is null then the
system assumes it to be 01 Jan 0001 and if the
// end date is null then it is assumed to be 01 Dec 3099.
```

FindMinInRepeatingForms( )

Find the minimum value of a given data item in all instances of a repeating form. The data is retrieved by form ID and works only for numeric fields.

For this function:

- You cannot use drop-downs, radio buttons, or checkbox values as function parameters or as a target.

> **Note:**
>
> This is an aggregation function. The rule is run for each form instance in the case where the target is on a repeating form.

Syntax

```
FindMinInRepeatingForms(variable)
```

Parameters

variable
Item variable to search.

Return value

Returns the minimum value across all instances or **null** if no minimum can be found.

Example 2-30 Find the minimum value of the "weight" number item across all repeating form instances in a visit

```
// Given 5 repeating form instances with "weight" item containing values of
"150, 200, 250, 300, 350"
return FindMinInRepeatingForms(varWeight);

// returns 150
```

FindMaxInRepeatingForms( )

Find the maximum value of a given data item in all instances of a repeating form. The data is retrieved by form ID and works only for numeric fields.

For this function:

- You cannot use drop-downs, radio buttons, or checkbox values as function parameters or as a target.

> **Note:**
>
> This is an aggregation function. The rule is run for each form instance in the case where the target is on a repeating form.

Syntax

```
FindMaxInRepeatingForms(variable)
```

Parameters

variable
Rule variables, which reference path: `eventId.formId.itemId`.

Return value

Returns the maximum value across all instances or **null** if no maximum can be found.

Example 2-31 Find the maximum value of "weight" number item across all repeating form instances in a visit

```
// Given 5 repeating form instances with "weight" item containing
values of "150, 200, 250, 300, 350"
return FindMaxInRepeatingForms(varWeight);

// returns 350
```

FindMinDateInRFs( )

Find the minimum value of given date, date-time, or partial date items in all repeating instances of the form identified by the form ID. This function is only applicable to date fields.

> **Note:**
>
> This is an aggregation function. The rule is run for each form instance in the case where the target is on a repeating form.

Syntax

```
FindMinDateInRFs(variable, DateMask)
```

Parameters

variable
Rule variables, with reference path: `eventId.formId.itemId`.

DateMask
Optional. Date mask to use for substitution for partial dates If not provided, partial dates are excluded from the calculation.

> **Tip:**
>
> Use `'DD-MON'` format as **DateMask** to substitute unknown (`UNK`) values in the partial date values.
> For example, if the mask is `'01-MAR'`:
>
> | Partial date value | Masked date | Notes |
> |---|---|---|
> | `'UNK-FEB-2020'` | `'01-FEB-2020'` | Made effective for calculation using the *day* part of the mask. |
> | `'UNK-UNK-2020'` | `'01-MAR-2020'` | Made effective for calculation using both, the *day* and *month* parts of the mask. |

Return value

Minimum date value in String format (**null** if the minimum date is not found). For example, 27-Jan-2021 00:00.

Example 2-32 Get the minimum date value for an item across a set of repeating form instances

```
// Get the minimum date value for an item across a set of repeating form
instances

return FindMinDateInRFs(aeDate);

// Same as above, using a partial date field aeDate (UNK-MMM-YYYY)

return FindMinDateInRFs(aeDate, '01-JAN');

//to compare with another date
var maxd= FindMinDateInRFs(a);
var today = new Date();
var maxdate = new Date(maxd);
```

```
if(dateDiffInDays(today,maxdate)>0){
   return "today>max";
}
return "today<max";
```

FindMaxDateInRFs( )

Find the maximum value of the given date, date-time, or partial-date items in all of the repeating instances of the form identified by the form ID. This function is only applicable to date fields.

> **Note:**
>
> This is an aggregation function. The rule is run for each form instance in the case where the target is on a repeating form.

Syntax

```
FindMaxDateInRFs(variable, DateMask)
```

Parameters

variable
Rule variables, with reference the path: `eventId.formId.itemId`.

DateMask
Optional. Date mask to use for substitution for partial dates. If not provided, partial dates are excluded from the calculation.

> **Tip:**
>
> Use `'DD-MON'` format as **DateMask** to substitute unknown (`UNK`) values in the partial date values.
> For example, if the mask is `'01-MAR'`:

Partial date value	Masked date	Notes
`'UNK-FEB-2020'`	`'01-FEB-2020'`	Made effective for calculation using the *day* part of the mask.
`'UNK-UNK-2020'`	`'01-MAR-2020'`	Made effective for calculation using both, the *day* and *month* parts of the mask.

Return value

Maximum date value in String format (**null** if maximum is not found). For example, 27-Jan-2021 00:00.

Example 2-33

```
// Get the maximum date value for an item across a set of repeating form
instances

return FindMaxDateInRFs(aeDate);

// Same as above, using a partial date field aeDate (UNK-MMM-YYYY)

return FindMaxDateInRFs(aeDate, '01-JAN');



//to compare with another date
var maxd= FindMaxDateInRFs(a);
var today = new Date();
var maxdate = new Date(maxd);
if(dateDiffInDays(today,maxdate)>0){
   return "today>max";
}
return "today<max";
```

FindMatchingRepeatingForm( )

Find a repeating form instance that contains a value that matches the search value.

For this function:

- You cannot use drop-downs, radio buttons, or checkbox values as function parameters or as a target.
- If a variable is designed to hold a partial date, you must provide the value for that parameter in the same partial date format.

> **Note:**
>
> This is an aggregation function. The rule is run for each form instance in the case where the target is on a repeating form.

Syntax

```
FindMatchingRepeatingForm(variable1, value1, variable2, value2, ...)
```

Parameters

variable(s)
Item variable to search.

value(s)
Search value(s). These must be hard-coded and cannot be rule variables. Hard-coded dates must be provided as a string **'Date(dd-mmm-yyyy)'**. This function can accept partial dates in the formats **<mmm-yyyy>** and **<yyyy>**.

Return value

Returns **-1** if no matches are found or the index number (**>0**) of the form instance if at least one matching instance is found. If multiple instances are found, only the first index is returned.

Example 2-34 Raise a query if any instances exist where symptom = "headache" and pulse rate = "100"

```
// Given 5 repeating form instances with items "itmSymptom" and
"itmPulse"
if (FindMatchingRepeatingForm(itmSymptom, "headache", itmPulse, 100) >
0) {
  return false;
} else {
  return true;
}

// Fires query if any of the 5 instances contain both itmSymptom =
"headache" AND itmPulse = 100.
```

FindMatchingRepeatingFormWithinRange( )

Find an instance of a repeating form, identified by the form ID, that matches the item value provided as a search key. The search can be based on search keys or date ranges.

For this function:

- You cannot use drop-downs, radio buttons, or checkbox values as function parameters or as a target.

The first two parameters should always be the date range. Additional search keys may be provided after that. The Rule returns the index of the repeating form instance with overlapping dates. You can use partial dates in the formats **<mmm-yyyy>** and **<yyyy>**. Dates must be provided inside the string **'Date(dd-mmm-yyyy)'**.

> **Note:**
>
> This function also considers entries where the dates are **null**. This is an aggregation function. The rule is run for each form instance in the case where the target is on a repeating form.

Syntax

```
FindMatchingRepeatingFormWithinRange(startDateVariable,startDateValue,
endDateVariable, endDateValue, variable1, value1, variable2, value2, ...)
```

Parameters

startDateVariable
Required. Date item.

startDateValue
Required. Date value in the form of a string. These must be hard-coded and cannot be rule variables. This function can accept partial dates in the formats **<mmm-yyyy>** and **<yyyy>**. Dates must be provided inside the string **'Date(dd-mmm-yyyy)'**.

endDateVariable
Required. Date item.

DateValue
Required. Date value in the form of a string. These must be hard-coded and cannot be rule variables. This function can accept partial dates in the formats **<mmm-yyyy>** and **<yyyy>**. Dates must be provided inside the string **'Date(dd-mmm-yyyy)'**.

variable(s)
Optional. Item variable to search.

value(s)
Optional. Search value(s). These must be hard-coded and cannot be rule variables.

Return value

Returns **-1** if no matches are found or the index number (>0) of the form instance if at least one matching instance is found. If multiple instances are found, only the first index is returned.

Example 2-35 Raise a query if any instances exist where a) onset date is between Jan 1 2020 and March 1 2020 and b) symptom = "headache"

```
// Given 5 repeating form instances with items "onsetStart", "onsetEnd" and
"itmSymptom":
if (FindMatchingRepeatingFormWithinRange(onsetStart, 'Date(01-JAN-2020)',
onsetEnd, 'Date(01-MAR-2020)', itmSymptom, "headache") > 0) {
  return false;
} else {
  return true;
}

// Fires query if any of the 5 instances contain onset dates between Jan 1
2020 - March 1 2020 AND itmSymptom = "headache"
//
// Note: If any repeating form instance has a null start or end date then
the system assumes a date in order to successfully
// perform the date range overlap check. If the start date is null then the
```

```
system assumes it to be 01 Jan 0001 and if the
// end date is null then it is assumed to be 01 Dec 3099.
```

FindRFInstance( )

Find a repeating form instance that contains a value which matches the search value using a supplied operator.

This function is similar to FindMatchingRepeatingForm( ). However, it allows the rule designer to specify matching operands (=, >, <, >=, <=).

> **Note:**
>
> This is an aggregation function. The rule is run for each form instance in the case where the target is on a repeating form.

Syntax

```
FindRFInstance (DateMask, variable1, compareOperator1, value1,
variable2, ...)
```

Parameters

DateMask
Flag to specify how partial dates should be handled.

- **null**. Ignore partial dates when doing comparisons.
- **value**. Replace the UNK component in the partial date with the specified value.

> **Tip:**
>
> Use `'DD-MON'` format as **DateMask** to substitute unknown (`UNK`) values in the partial date values.
>
> For example, if the mask is `'01-MAR'`:

Partial date value	Masked date	Notes
`'UNK-FEB-2020'`	`'01-FEB-2020'`	Made effective for calculation using the *day* part of the mask.
`'UNK-UNK-2020'`	`'01-MAR-2020'`	Made effective for calculation using both, the *day* and *month* parts of the mask.

variable
Item variables to search.

compareOperator
Operator to use for the compare: `=`, `>`, `<`, `>=`, `<=`.

value
Value to compare variable with as a string.

Return value

Returns **-1** if no matches are found or the index number (`>0`) of the form instance if at least one matching instance is found. If multiple instances are found, only the first index is returned.

> **Note:**
>
> - If other operands/variables are to be used for a value, it has to be assigned first to the JavaScript variable and then used in the function expression.
> - Values in the custom function should be JavaScript variables or direct values. Do not use operand variables directly in the custom function expression.
> - Date values can be provided as `'Date(10-Jun-2010)'` or `'10-Jun-2010'` or a JavaScript Date var.
> - For the DateMask, use `'DD-MON'` format to substitute values for `UNK` in the actual partial date values. For example, if the mask is `'01-MAR'`:
>   - `'UNK-FEB-2020' => '01-FEB-2020'`, effectively using the date part of the mask.
>   - `'UNK-UNK-2020' => '01-MAR-2020'`, effectively using both the date and the month part of the mask.

Example 2-36 Raise a query if there is an instance of pulse > 100

```
// Raise a query if there is an instance of pulse > 100
return (FindRFInstance(null, pulseVal,'>', 100) > 0)?false:true;

// Raise a query if there's an instance where onDate is >= 10-Jun-2010
(onDate is partial UNK-UNK-YYYY)
return (FindRFInstance('10-Jun', onDate, '>=', '10-Jun-2010') > 0)?
false:true;
```

ListRFInstances( )

List all instance numbers for a repeating form. You can use this helper function in your rule expression to check for instances of a specific question value in a repeating form.

> **Note:**
>
> This is an aggregation function. The rule is run for each form instance in the case where the target is on a repeating form.

Syntax

```
ListRFInstances(variable,includeDeleted)
```

Parameters

variable
An item that exists on the visit or form to search.

includeDeleted

- **null** or **0** - Do not include deleted Repeating instances in the return array.
- **1** - Include deleted Repeating instances in the array count.

Return value

An array of repeating form instance numbers.

Example 2-37 Raise a query if AE form instance #3 does not exist

```
// Raise a query if AE form instance #3 does not exist
var arrAE = ListRFInstances(onDate, 0);
return (arrAE.indexOf(2) == -1)?false:true;
```

> **Note:**
>
> In this example, `.indexOf(2)` equates to the third form instance as arrays start at position zero.

GetCurrentRFInstance( )

Get the form instance number where the rule is currently being run.

Syntax

```
GetCurrentRFInstance()
```

Parameters

None.

Return Value

Repeating form instance number where the rule is being run.

Example 2-38 If this is the first instance, raise a query if aeDate is not entered

```
// If this is the first instance, raise a query if aeDate is not entered


return (GetCurrentRFInstance() == 1 && aeDate === null)?false:true;
```

GetMatchingRepeatingFormsCount( )

Get the number of repeating form instances of a form that match the item values provided as search keys.

Syntax

```
GetMatchingRepeatingFormsCount(variable1, value1, variable2, value2, ...)
```

Parameters

variable
Rule variable.

value
Value to search for.

Return value

Count of matching repeating form instances.

- Can accept partial dates in the formats <mmm-yyyy> and <yyyy> and works with choice controls (radio controls, check box controls, and dropdowns).
- Choice controls can only be searched by label, not value.
- Dates have to be provided as a string in the format 'Date(dd-mmm-yyyy)'.
- Only one option can be provided as search text for choice controls.

Example 2-39 Raise a query if there is more than one instance where AE Outcome = 'Fatal"

```
// Raise a query if there is more than one instance where AE Outcome =
'Fatal"

// Get current repeating instance
var ins = GetCurrentRFInstance();
var curVal = "";

// Get value of aeOut from current instance
```

```
var rfData = getRFValues(ins, [aeOut] );
if(rfData.exists && rfData.aeOut){
        if((rfData.aeOut) !== "[]"){    // If the choice control has
been cleared out then do not read the label
            curVal = JSON.parse(rfData.aeOut)[0].label;
    }
}

// check to see if there are more than 1 instance with "Yes"
return ((curVal == "Fatal") && (GetMatchingRepeatingFormsCount(aeOut,
"Fatal") > 1))?false:true;
```

getRFValues( )

Retrieves the current values for specified items on repeating form instances.

If you'd like to fetch only a single value from a repeating form instance, you may also consider getQuestionValue( ).

> **Note:**
>
> This is an aggregation function, the rule will be run for each form instance in case the target is on a repeating form.

Syntax

```
getRFValues(formInstance, [var1, var2, varN] )
```

Parameters

formInstance
The instance number of the form you're retrieving values from. This parameter can be a JavaScript variable or it can be a number.

var1, var2, varN
Item variable values to retrieve.

Return value

Returns a JSON object containing the variables (of the same name as was passed in `param2`) with values:

- Returned variable value will be a C1Date object in case of variable being a partial date, or variable value will be a Date object in the case of the variable being a full date. This can be checked using the `isPartialDate( )` function as illustrated below.
- If the variable is a choice control (checkbox, radio button, or drop-down) then the returned variable will be in JSON format: (`"[{\"value\":\"3\",\"label\":\"TestLabel\"}]"`). This can be parsed using `JSON.parse` or `parseChoice( )`.
  - `parseChoice(rfData.v4_chk4))`

  - `JSON.parse(rfDate.v4_chk4))`

- The return object has a property name `'exists'` which returns **true** if any one of the variable passed in has a value for the passed in repeat form instance number.

Example 2-40 Get values for 3 item variables in AE form instance #1, and put them to a text item

```
varrfData = getRFValues(1, ["aeTerm","aeDate","aeSerious"] );
if(rfData.exists){
    returnrfData.aeTerm + "  | "  + rfData.aeDate.getFullYear() + " | "+
JSON.parse(rfData.aeSerious)[0].label;
} else{
    return;
}


// It is best practice to check if the variable has value before using it
varrfData = getRFValues(1, ["aeTerm","aeDate"] );
if(rfData.exists && rfData.aeTerm && rfData.aeDate){
    returnrfData.aeTerm + "  | "  + rfData.aeDate.getFullYear() ;
} else{
    return;
}
```

Two-section form functions

Find or evaluate a value in a two-section form.

- findDuplicate2SForm( )
  Use this helper function to detect duplicate data in a two-section form. The data is identified by a form ID which has duplicate item values for the search keys that are provided in each function argument evaluated as one key.
- findDuplicate2SFormWithinRange( )
  Detect duplicate data in a two-section form. The data is identified by row ID which has duplicate item values for the date ranges or search keys that are provided.
- findMinIn2SForms( )
  Find the minimum value of a given data item in all instances of the repeating section in a two-section form. This function works only for numeric fields.
- findMaxIn2SForms( )
  Find the maximum value of a given data item in all instances of a repeating section in a two-section form. This function works only for numeric fields.
- findMinDateIn2SForm( )
  Find the minimum value of given date, date-time, or partial date items in all repeating instances of a two-section form. This function is only applicable to date fields.
- findMaxDateIn2SForm( )
  Find the maximum value of the given date, date-time, or partial-date items in all of the repeating instances of a two-section form. This function is only applicable to date fields.
- findMatching2SForm( )
  Find a repeating section instance of a two-section form, identified by the row ID, that matches the item value provided as a search key. This function supports partial dates

- findMatching2SFormWithinRange( )
  Find an instance of a repeating section of a two-section form, identified by the row ID, that matches the item value provided as a search key. The search can be based on search keys or date ranges.
- find2SFormInstance( )
  Find an instance of the repeating section in a two-section form that contains a value which matches the search value using a supplied operator.
- list2SInstances( )
  List all instance numbers for a two-section form.
- getCurrent2SFormInstance( )
  Get the form instance number where the rule is currently being run.
- getCurrent2STableInstance( )
  For two-section forms, find the current table row instances where the rule is currently being run.
- getMatching2SFormsCount( )
  Get the number of repeating instances in a two-section form that match the item values provided as search keys.
- get2SValues( )
  Retrieve values for the provided variables of a two-section form or variables of a table in a two-section form based on the `tableInstance` parameter.

findDuplicate2SForm( )

Use this helper function to detect duplicate data in a two-section form. The data is identified by a form ID which has duplicate item values for the search keys that are provided in each function argument evaluated as one key.

Use as many arguments as needed to fully defined the duplicate key.

For this function:

- You cannot use drop-downs, radio buttons, or checkbox values as function parameters or as a target.
- Partial dates are accepted for date comparison.

> **Note:**
>
> The presence of any partial date among instances will make other full dates to be taken in the same format for comparison. For example, if there is a partial date instance '`UNK-JAN-2022`', only the month and year values in other dates will be taken for the comparison, even if they are full dates. Similarly, if there is a partial date instance '`UNK-UNK-2022`', only the year value will be used for comparison in all dates.

This is an aggregation function. The rule will be run for each form instance in the case where the target is on a two-section form.

Syntax

```
findDuplicate2SForm(formInstance, variable1, variable2,...)
```

Parameters

variable(s)
Item variables to evaluate. These are declared rule variables with reference path `eventId.formId.itemId`.

formInstance
If this is **null**, the check is done for all instances of a two-section form. If a value is provided, the check is done for table rows on that two-section form instance.

Return value

Returns **`true`** if duplicate values are found; **`false`** if duplicate values are not found.

Example 2-41 Check to see if any duplicate two-section form instances exist with the same values for Lab and Test Name

```
// Given 5 two-section form instances with items "Lab" and "Test Name"
if (findDuplicateRepeatingForm(itmLab, itmTestName)) {
  return false;
} else {
  return true;
}
```

Example 2-42 Check to see is a form instance is a duplicate of another form instance

```
// Raise a query if 2 section form instance is duplicate of any other form
instance

returnfindDuplicate2SForm(null, "txt");

// Raise a query if 2 section table instance #2 is duplicate of any other
table instance

vararrAE = list2SInstances(2, "txt");
return(arrAE.length <= 1)?false:true;
```

findDuplicate2SFormWithinRange( )

Detect duplicate data in a two-section form. The data is identified by row ID which has duplicate item values for the date ranges or search keys that are provided.

For this function:

- You cannot use drop-downs, radio buttons, or checkbox values as function parameters or as a target.
- If a variable is designed to hold a partial date, you must provide the value for that parameter in the same partial date format.
- If a record has no data entered for the required dates, the record is skipped for comparison.

The first two parameters should always be the date range. Additional search keys can be provided.

> **Note:**
>
> This is an aggregation function. The rule will be run for each instance in the case where the target is on the repeating section of a two-section form.

Syntax

```
findDuplicate2SFormWithinRange(formInstance,startDateVariable,endDateVa
riable,variable1,...)
```

Parameters

formInstance
Determines within which two-section form instance to look for a duplicate:

- If `null` and variable in the flat section, will search for a duplicate in the flat section across all form instances.
- If `null` and variable in the table section, will search for a duplicate in all table rows across all form instances.
- If a `formInstance` value is provided, the search will be performed across all the table rows of the specified instance.

startDateVariable
(Required) Date item on a repeating form. The date selected in this field, as per the variable definition, will be taken in as the lower limit of the date range.

endDateVariable
(Required) Date item on a repeating form. The date selected in this field, as per the variable definition, will be taken in as the higher limit of the date range.

variable(s)
(Optional) Additional Item variable to search.

Return value

Returns **true** if duplicate values are found; **false** if duplicate values are not found.

Example 2-43 Check to see if any instance exist within the same onset date range and symptom value across all two-section form instances.

```
// Given 5 repeating form instances with items "onsetStartDate",
"onsetEndDate and "Symptom"
if
(findDuplicate2SFormWithinRange(null,itmOnsetDateStart,itmOnsetDateEnd,
itmSymptom) === true) {
  return false;
} else {
  return true;
}

// Fires a query if more than 1 repeating instance in a two-section
form is found within the same date range and with the same symptom.
```

```
//
// Note: If any repeating instance of the two-section form has a null start
or end date then the system assumes a date in order to successfully
// perform the date range overlap check. If the start date is null then the
system assumes it to be 01 Jan 0001 and if the
// end date is null then it is assumed to be 01 Dec 3099.
```

findMinIn2SForms( )

Find the minimum value of a given data item in all instances of the repeating section in a two-section form. This function works only for numeric fields.

For this function you can only use number-type questions. You cannot use drop-downs, radio buttons, or checkbox values as function parameters or as a target.

> **Note:**
>
> This is an aggregation function. The rule is run for each instance in the case where the target is on a repeating section of a two-section form.

Syntax

```
findMinIn2SForms(variable)
```

Parameters

variable
Item variable to search.

Return value

Returns the minimum value across all instances or **null** if no minimum can be found.

Example 2-44 Find the minimum value of the "weight" number item across all repeating section instances in a two-section form in a visit

```
// Given 5 repeating section instances in a two-section form with "weight"
item containing values of "150, 200, 250, 300, 350"
return findMinIn2SForms(varWeight);

// returns 150
```

findMaxIn2SForms( )

Find the maximum value of a given data item in all instances of a repeating section in a two-section form. This function works only for numeric fields.

For this function you can only use number-type questions. You cannot use drop-downs, radio buttons, or checkbox values as function parameters or as a target.

> **Note:**
>
> This is an aggregation function. The rule is run for each instance in the case where the target is on a repeating section of a two-section form.

Syntax

```
findMaxIn2SForms(variable)
```

Parameters

variable
Item variable to search, which reference path is: `eventId.formId.itemId`.

Return value

Returns the maximum value across all instances or **null** if no maximum can be found.

Example 2-45 Find the maximum value of "weight" number item across all repeating instances of a two-section form in a visit

```
// Given 5 repeating section instances in a two-section form with
"weight" item containing values of "150, 200, 250, 300, 350"
return findMaxIn2SForms(varWeight);

// returns 350
```

findMinDateIn2SForm( )

Find the minimum value of given date, date-time, or partial date items in all repeating instances of a two-section form. This function is only applicable to date fields.

> **Note:**
>
> This is an aggregation function. The rule is run for each instance in the case where the target is on the repeating section of a two-section form.

Syntax

```
findMinDateIn2SForm(variable, DateMask)
```

Parameters

variable
Rule variables, with reference path: `eventId.formId.itemId`.

DateMask
Optional. Date mask to use for substitution for partial dates. If not provided, partial dates are excluded from the calculation.

> **Tip:**
>
> Use `'DD-MON'` format as **DateMask** to substitute unknown (`UNK`) values in the partial date values.
> For example, if the mask is `'01-MAR'`:
>
> | Partial date value | Masked date | Notes |
> |---|---|---|
> | `'UNK-FEB-2020'` | `'01-FEB-2020'` | Made effective for calculation using the *day* part of the mask. |
> | `'UNK-UNK-2020'` | `'01-MAR-2020'` | Made effective for calculation using both, the *day* and *month* parts of the mask. |

Return value

Minimum date value in String format (**null** if the minimum date is not found). For example, 27-Jan-2021 00:00.

Example 2-46 Get the minimum date value for an item across a set of repeating section instances on a two-section form

```
// Get the minimum date value for an item across a set of repeating section
instances on a two-section form

return findMinDateIn2SForm(aeDate);

// Same as above, using a partial date field aeDate (UNK-MMM-YYYY)

return findMinDateIn2SForm(aeDate, '01-JAN');

//to compare with another date
var mind= findMinDateIn2SForm(a);
var today = new Date();
var mindate = new Date(mind);
if(dateDiffInDays(today,mindate)>0){
   return "today>min";
}
return "today<min";
```

findMaxDateIn2SForm( )

Find the maximum value of the given date, date-time, or partial-date items in all of the repeating instances of a two-section form. This function is only applicable to date fields.

> **Note:**
>
> This is an aggregation function. The rule is run for each repeating section instance in the case where the target is on a two-section form.

Syntax

```
findMaxDateIn2SForm(variable, DateMask)
```

Parameters

variable
Rule variables, with reference the path: `eventId.formId.itemId`.

DateMask
Optional. Date mask to use for substitution for partial dates. If not provided, partial dates are excluded from the calculation.

> **Tip:**
>
> Use `'DD-MON'` format as **DateMask** to substitute unknown (`UNK`) values in the partial date values.
> For example, if the mask is `'01-MAR'`:
>
> | Partial date value | Masked date | Notes |
> |---|---|---|
> | `'UNK-FEB-2020'` | `'01-FEB-2020'` | Made effective for calculation using the *day* part of the mask. |
> | `'UNK-UNK-2020'` | `'01-MAR-2020'` | Made effective for calculation using both, the *day* and *month* parts of the mask. |

Return value

Maximum date value in String format (**null** if maximum is not found). For example, 27-Jan-2021 00:00.

Example 2-47 Get the maximum date value for an item across a set of repeating section instances on a two-section form

```
// Get the maximum date value for an item across a set of repeating section
instances on a two-section form

return findMaxDateIn2SForm(aeDate);

// Same as above, using a partial date field aeDate (UNK-MMM-YYYY)

return findMaxDateIn2SForm(aeDate, '01-JAN');



//to compare with another date
var maxd= findMaxDateIn2SForm(a);
var today = new Date();
var maxdate = new Date(maxd);
if(dateDiffInDays(today,maxdate)>0){
   return "today>max";
}
return "today<max";
```

findMatching2SForm( )

Find a repeating section instance of a two-section form, identified by the row ID, that matches the item value provided as a search key. This function supports partial dates

For this function:

- Drop downs, radio buttons, and checkbox values are not supported as a function parameter or as a target.
- If a variable is designed to hold a partial date then provide the value for that parameter in the same partial date format.

> **Note:**
>
> The presence of any partial date among instances will make other full dates to be taken in the same format for comparison. For example, if there is a partial date instance '`UNK-JAN-2022`', only the month and year values in other dates will be taken for the comparison, even if they are full dates. Similarly, if there is a partial date instance '`UNK-UNK-2022`', only the year value will be used for comparison in all dates.

This is an aggregation function. The rule is run for each form instance in the case where the target is on a two-section form.

Syntax

```
findMatching2SForm(formInstance, variable1, value1, variable2, value2, ...)
```

Parameters

formInstance

- If **null**, the search returns an array of existing instances of two-section forms. Ideally this should be used with variables inside the flat section of the form.
- If a value is provided, the search returns an array of existing table rows on the specified instance of the two-section form.

variable(s)
Item variable to search.

value(s)
Search value(s). These values must be hard-coded and cannot be rule variables. Hard-coded dates must be provided as a string `'Date(dd-mmm-yyyy)'`. The function accepts partial dates in the formats **<mmm-yyyy>** and **<yyyy>**.

Return value

Returns **-1** if no matches are found or the index number (`>0`) of the form instance if at least one matching instance is found. If multiple instances are found, only the first index is returned.

When searching across all instances for the two-section form (such as `formInstance = null`), the function returns the form instance number of the match. When searching a specific instance (for example, `formInstance = 1`), The function returns the table row instance number of the match.

Example 2-48 Raise a query if any instances exist where symptom = "headache" and pulse rate = "100"

```
// Given 5 two-sections form instances with items "itmSymptom" and
"itmPulse" on flat part
// Fires query if any of the 5 instances contain both itmSymptom =
"headache" AND itmPulse = 100.
if (findMatching2SForm(null, itmSymptom, "headache", itmPulse, 100) >
0) {
  return false;
} else {
  return true;
}

// Search table rows inside the 4th instance of the 2-section form
// Fires query if any rows inside the 4th form instance contain both
itmSymptom = "headache" AND itmPulse = 100.
if (findMatching2SForm(4, itmSymptom, "headache", itmPulse, 100) > 0) {
  return false;
} else {
  return true;
}
```

findMatching2SFormWithinRange( )

Find an instance of a repeating section of a two-section form, identified by the row ID, that matches the item value provided as a search key. The search can be based on search keys or date ranges.

For this function:

- You cannot use drop-downs, radio buttons, or checkbox values as function parameters or as a target.

The first two parameters should always be the date range. Additional search keys may be provided after that. The Rule returns the index of the repeating section instance in a two-section form with overlapping dates. You can use partial dates in the formats **<mmm-yyyy>** and **<yyyy>**. Dates must be provided inside the string **'Date(dd-mmm-yyyy)'**.

> **Note:**
>
> This function also considers entries where the dates are **null**. This is an aggregation function. The rule is run for each instance in the case where the target is on the repeating section of a two-section form.

Syntax

```
findMatching2SFormWithinRange(startDateVariable,startDateValue,
endDateVariable, endDateValue, variable1, value1, variable2, value2, ...)
```

Parameters

startDateVariable
Required. Date item.

startDateValue
Required. Date value in the form of a string. These must be hard-coded and cannot be rule variables. This function can accept partial dates in the formats **<mmm-yyyy>** and **<yyyy>**. Dates must be provided inside the string **'Date(dd-mmm-yyyy)'**.

endDateVariable
Required. Date item.

DateValue
Required. Date value in the form of a string. These must be hard-coded and cannot be rule variables. This function can accept partial dates in the formats **<mmm-yyyy>** and **<yyyy>**. Dates must be provided inside the string **'Date(dd-mmm-yyyy)'**.

variable(s)
Optional. Item variable to search.

value(s)
Optional. Search value(s). These must be hard-coded and cannot be rule variables.

Return value

Returns **-1** if no matches are found, or the index number (**>0**) of the instance of the repeating section in a two-section form if at least one matching instance is found. If multiple instances are found, only the first index is returned.

Example 2-49 Raise a query if any instances exist where a) onset date is between Jan 1 2020 and March 1 2020 and b) symptom = "headache"

```
// Given 5 repeating form instances with items "onsetStart",
"onsetEnd" and "itmSymptom":
if (findMatching2SFormWithinRange(onsetStart, 'Date(01-JAN-2020)',
onsetEnd, 'Date(01-MAR-2020)', itmSymptom, "headache") > 0) {
  return false;
} else {
  return true;
}

// Fires query if any of the 5 instances contain onset dates between
Jan 1 2020 - March 1 2020 AND itmSymptom = "headache"
//
// Note: If any repeating form instance has a null start or end date
then the system assumes a date in order to successfully
// perform the date range overlap check. If the start date is null
then the system assumes it to be 01 Jan 0001 and if the
// end date is null then it is assumed to be 01 Dec 3099.
```

find2SFormInstance( )

Find an instance of the repeating section in a two-section form that contains a value which matches the search value using a supplied operator.

This function is similar to findMatching2SForm( ). However, it allows the rule designer to specify matching operands (=, >, <, >=, <=).

> **Note:**
>
> This is an aggregation function. The rule is run for each instance in the case where the target is on the repeating section in a two-section form.

Syntax

```
find2SFormInstance(DateMask, variable1, compareOperator1, value1,
variable2, ...)
```

Parameters

DateMask
Flag to specify how partial dates should be handled.

- **'null'**. Ignore partial dates when doing comparisons.
- **value**. Replace the UNK component in the partial date with the specified value.

> **Tip:**
>
> Use `'DD-MON'` format as **DateMask** to substitute unknown (`UNK`) values in the partial date values.
>
> For example, if the mask is `'01-MAR'`:
>
> | Partial date value | Masked date | Notes |
> |---|---|---|
> | `'UNK-FEB-2020'` | `'01-FEB-2020'` | Made effective for calculation using the *day* part of the mask. |
> | `'UNK-UNK-2020'` | `'01-MAR-2020'` | Made effective for calculation using both, the *day* and *month* parts of the mask. |

variable
Item variables to search.

compareOperator
Operator to use for the comparison: `=`, `>`, `<`, `>=`, `<=`.

value
Value to compare variable with as a string.

Return value

Returns **-1** if no matches are found or the index number (**>0**) of the repeating section instance if at least one matching instance is found. If multiple instances are found, only the first index is returned.

> **Note:**
>
> - Do not use operand variables directly in the custom function expression. Values in the custom function should be JavaScript variables or direct values. If other operands/variables are to be used for a value, it has to be assigned first to the JavaScript variable and then used in the function expression.
> - Date values can be provided as `'Date(10-Jun-2010)'` or `'10-Jun-2010'` or a JavaScript Date var.

Example 2-50 Raise a query if there is an instance of pulse > 100

```
// Raise a query if there is an instance of pulse > 100
return (find2SFormInstance(null, pulseVal,'>', 100) > 0)?false:true;
```

Example 2-51 Raise a query if there is an instance of a date on or after a given date

```
// Raise a query if there's an instance where onDate is >= 10-Jun-2010
//(onDate is partial UNK-UNK-YYYY)
return (Find2SFormInstance('10-Jun', onDate, '>=', '10-Jun-2010') > 0)?
false:true;


//Example using operand variable values
//Raise a query if there's an instance where onDate is on or after the
enddt
dtval=enddt;
return (Find2SFormInstance('10-Jun', onDate, '>=', dtval) > 0)?
false:true;
```

list2SInstances( )

List all instance numbers for a two-section form.

> **Note:**
>
> This is an aggregation function. The rule is run for each form instance in the case where the target is on a two-section form.

Syntax

```
list2SInstances(variable,includeDeleted)
```

Parameters

variable
An item that exists on the visit or form to search.

includeDeleted

- **null** or **0** - Do not include deleted two-section instances in the return array.
- **1** - Include deleted two-section instances in the array count.

Return value

An array of two-section form instance numbers.

Example 2-52 Raise a query if AE form instance #2 does not exist

```
// Raise a query if AE form instance #2 does not exist
var arrAE = list2SInstances(onDate, 0);
return (arrAE.indexOf(2) == -1)?false:true;
```

getCurrent2SFormInstance( )

Get the form instance number where the rule is currently being run.

> **Note:**
>
> This is an aggregation function. The rule is run for each form instance in the case where the target is on a two-section form.

Syntax

```
getCurrent2SFormInstance()
```

Parameters

None.

Return value

Two-section form instance number where the rule is being run.

Example 2-53 If this is the first instance, raise a query if aeDate is not entered

```
// If this is the first instance, raise a query if aeDate is not entered


return (getCurrent2SFormInstance() == 1 && aeDate === null)?false:true;
```

getCurrent2STableInstance( )

For two-section forms, find the current table row instances where the rule is currently being run.

> **Note:**
>
> This is an aggregation function. The rule is run for each form instance in the case where the target is on a two-section form.

Syntax

```
getCurrent2STableInstance()
```

Parameters

None.

Return value

- Instance number (starts with 1) where the rule is currently being run.
- **-1** if it is not a two-section form.

Example 2-54 If this is the first instance, raise a query if aeDate is not entered

```
// If this is the first instance, raise a query if aeDate is not
entered


return (getCurrent2STableInstance() == 1 && aeDate === null)?
false:true;
```

getMatching2SFormsCount( )

Get the number of repeating instances in a two-section form that match the item values provided as search keys.

Syntax

```
getMatching2SFormsCount(formInstance, variable1, value1, variable2,
value2, ...)
```

Parameters

formInstance
Determines within which two-section form instance or section to look for a duplicate:

- If `null` and variable in the flat section, will search for a duplicate in the flat section across all form instances.
- If `null` and variable in the table section, will search for a duplicate in all table rows across all form instances.
- If a `formInstance` value is provided and variable is in the flat section, the search will be performed in the flat section of the specified instance.

> **Note:**
>
> This would constitute a search in a single instance.

- If a `formInstance` value is provided and variable is in the table section, the search will be performed across all the table rows of the specified instance.

variable
Rule variable.

value
Value to search for.

Return value

Count of matching instances depending on the passed in parameters:

- If `formInstance` is `null` and variable is in flat section, the count of matching repeating form instances will be returned.
- If `formInstance` is `null` and variable is in a table row, the count of matching repeating table row instances will be returned.
- If `formInstance` value is provided and variable is in the flat section, the count of matching instances within the flat section of specified form instance will be returned.
- If `formInstance` value is provided and variable is in the table section, the count of matching repeating table row instances within the specified form instance will be returned.

Usage Tips

- Can accept partial dates in the formats <mmm-yyyy> and <yyyy> and works with choice controls (radio controls, check box controls, and dropdowns).
- Choice controls can only be searched by label, not value.
- Dates have to be provided as a string in the format 'Date(dd-mmm-yyyy)'.
- Only one option can be provided as search text for choice controls.

Example 2-55 Raise a query if there is more than one instance where AE Outcome = 'Fatal"

```
// Raise a query if there is more than one instance where AE Outcome =
'Fatal"

// Get current repeating instance
var ins = GetCurrent2SFormInstance();
var curVal = "";

// Get value of aeOut from current instance
var 2sData = get2SValues(ins,getCurrent2STableInstance(),[aeOut] );
if(2sData.exists && 2sData.aeOut){
        if((2sData.aeOut) !== "[]"){    // If the choice control has been
cleared out then do not read the label
            curVal = JSON.parse(2sData.aeOut)[0].label;
    }
}

// check to see if there are more than 1 instance with "Yes"
return ((curVal == "Fatal") && (getMatching2SFormsCount(1, aeOut, "Fatal") >
1))?false:true;
```

get2SValues( )

Retrieve values for the provided variables of a two-section form or variables of a table in a two-section form based on the `tableInstance` parameter.

If you want to fetch a single value from a two-section form instance, consider getQuestionValue( ).

Syntax

```
get2SValues(formInstance, tableRowInstance, [var1, var2, varN])
```

Parameters

formInstance
Instance number of the two-section form to retrieve values from. This can be a JavaScript variable or a number.

tableRowInstance
Instance number of the table of the two-section form to retrieve values from. This parameter is **null** if you want to retrieve the value from the flat part of the two-section form. This parameter can be a JavaScript variable or a number.

var1, var2, varN...
Item variable values to retrieve.

Return value

Returns a JSON object containing the variables (of same name as passed in `param2`) with values:

- The returned variable value is the C1Date object if the variable is a partial date, or a Date object if the variable is a full date. You can check this using the `isPartialDate( )` function.
- If the variable is a choice control (checkbox, radio, or drop-down), the returned variable value is in JSON format: (`"[{\"value\":\"3\",\"label\":\"TestLabel\"}]"`). This can be parsed using `JSON.parse` or the helper function `parseChoice( )`.
  - `parseChoice(rfData.v4_chk4))`
  - `JSON.parse(rfData.v4_chk4))`
- The return object has a property named `exists` which returns **true** if any one of the variables passed in has value for the passed in two-section form instance number.

Example 2-56 Get values for three item variables in AE form instance #1, and put them to a text item

```
// Get values for 3 item variables in AE form instance #1, and put
them to a text item
var rfData = get2SValues(1, null, ["aeTerm","aeDate","aeSerious"] );
if(rfData.exists){
   return rfData.aeTerm + "  | "  + rfData.aeDate.getFullYear() + " |
```

```
" + JSON.parse(rfData.aeSerious)[0].label;
} else {
    return;
}

// It is best practice to check if the variable has value before using it
var rfData = get2SValues(1, 2, ["aeTerm","aeDate"] );
if(rfData.exists && rfData.aeTerm && rfData.aeDate){
   return rfData.aeTerm + "  | "  + rfData.aeDate.getFullYear() ;
} else {
    return;
}
```

Control the behavior of a rule

Control how a rule behaves, whether it relates to a study's version, visits in the study or the JavaScript expression logic.

- isStudyVersion( )
  Compare the provided version with the current study version using the operator provided.
- getCurrentVisitPropertyValue( )
  Control rule behavior on different visits. For instance, if a rule is created on a question in a form, and that form is associated with multiple visits, you can raise a query only on certain visits in a study, not all visits.
- logMsg()
  Get specific information regarding a rule's logic while debugging. Place log statements where needed in the JavaScript expression to display values of defined variables and messages to reveal a rule's behavior.

isStudyVersion( )

Compare the provided version with the current study version using the operator provided.

Syntax

```
isStudyVersion(operator, version)
isStudyVersion(operator, version, variable1, variable2, ...)
```

Parameters

operator
Can be any of the following: <, >, =, <=, >=.

variable
Variables that are used in the rule expression for the **true** condition.

Return value

Returns **true** or **false**, depending on the comparison result.

Example 2-57 Example 1

```
//If Study Version is >= 1.0.0.5, multiply num1 by 10. Otherwise just
return num1.
if (isStudyVersion(">=", "1.0.0.5")) {
    return num1*10;
} else {
    // Do something else
    return num1;
}
```

Example 2-58 Example 2

```
If( isStudyVersion(">","10.1.2", variable1, variable2, variable3) ) {
//do something
return variable1 + variable2 + variable3;
} else {
    If( isStudyVersion("<=","10.1.2", variable1, variable2) )
        //do something else
        return variable1 + variable2;
    }
}
```

Note:

- `isStudyVersion( )` with two parameters doesn't cover the situation where item data is cleared or never entered. You can take care of this case by comparing variables with an empty value and writing your own code for such a situation. (For example, `if (var) { //do something}`).
- `isStudyVersion( )` with more than two parameters lets a rule behave similar to standard rule behavior for the situation where data is cleared or never entered. In this case, no extra action is required.
- If `isStudyVersion( )` is used with more than two parameters, use an `else` condition as shown in *Example 2*.

getCurrentVisitPropertyValue( )

Control rule behavior on different visits. For instance, if a rule is created on a question in a form, and that form is associated with multiple visits, you can raise a query only on certain visits in a study, not all visits.

By default, a rule is executed against every visit in the study that contains the form. When declaring variables, for the visit field:

- If you leave *-All Visits-* in the visits field, the variable data will be retrieved from the form in current visit where rule is being run.
- If you select a specific visit, for example the Screening visit, the variable data will be retrieved from the form in the specified visit, in this case the Screening visit, for every visit where the rule is executed.

You use this helper function for a rule to apply and be executed only against a specific visit.

Syntax

```
getCurrentVisitPropertyValue('propertyName')
```

Parameters

propertyName
propertyName in single quotes. *propertyName* could be any of the following:

- title
- visit ID
- event type

eventtype values can be one of following based on visit Design time: **ScreeningVisit**, **ScheduleAbleVisit**, **SubjectWithdrawalVisit**, **SubjectCompletionVisit**, **UnScheduleAbleVisit**, **Event**, **AdverseEvent**.

Return value

Returns the current visit property.

Example 2-59 Fetch the property value of the provided property of the current visit

```
// Returns (short-name) the current visit property 'visitid':
return getCurrentVisitPropertyValue('visitid')

// Returns (name) the current visit property 'title':
return getCurrentVisitPropertyValue('title')

// Returns event-type of the visit - will be one of the following
"ScreeningVisit","ScheduleAbleVisit","SubjectWithdrawalVisit","SubjectComplet
ionVisit","UnScheduleAbleVisit","Event","AdverseEvent"
return getCurrentVisitPropertyValue('eventtype')


// Example to demonstrate functionality based on current visit
if(getCurrentVisitPropertyValue ("visitid")===='visit1'){
    //add visit1 functionality here
}
else if(getCurrentVisitPropertyValue ("visitid")===='visit2'){
    //add visit2 functionality here
}
else{
    //else functionality
}
```

logMsg()

Get specific information regarding a rule's logic while debugging. Place log statements where needed in the JavaScript expression to display values of defined variables and messages to reveal a rule's behavior.

Syntax

```
logMsg(argument)
```

Parameters

argument
Expression or variable value to get logged and displayed for debug.
Only **strings** and **numbers** are supported as arguments, which means variables of these types can be used. If you want to use an object as an argument you must use the `stringify()` method so it is passed as a string.

Return value

The argument passed to the `logMsg()` helper function will be returned and displayed in the log window when debugging.

Usage tips

> **Note:**
>
> Since the `logMsg()` helper function only runs in debug mode, there is no need to remove the calls before publishing the rule.

Example 2-60 Using labels for variables

Use labels when logging variables to make the output easier to follow.

```
logMsg("weight: "+weight); // logs the label and the variable value
```

Output in the log window

```
weight: 160
```

Example 2-61 Using log statements to debug rule logic

Log messages as flags to reveal the logic driving the rule's behavior.

```
var weight = "All visits"."Form Demo"."item weight";
logMsg("weight: "+weight); // logs the label and the variable value
if(weight >160){
    logMsg("weight > 160"); // log the execution path for "if" return
false; }
else{
```

```
    logMsg("NOT weight > 160"); // log the execution path for "else" return
true;}
```

Output in the log window

```
weight: 160
NOT weight > 160
```

Example 2-62 Using `stringify()` method to pass objects

Use `stringify()` method to parse objects to strings, so they can be used in log statements.

```
var val1 = getValues("dt1","tpt");
logMsg("dt1 = "+JSON.stringify(dt1));
```

Output in the Log Window

```
dt1 =

[{"visitName":"SCR","deleted":false,"tableRowInstance":null,"branchName":null
,"eventType":"ScreeningVisit","formRepeatNumber":1,
"value":"2022-03-09T00:00:00.000Z","cycleNumber":null,"empty":false,"treatmen
tArm":null},
{"visitName":"SCR","deleted":false,"tableRowInstance":null,"branchName":null,
"eventType":"ScreeningVisit","formRepeatNumber":2,
"value":"2022-03-09T00:02:00.000Z","cycleNumber":null,"empty":false,"treatmen
tArm":null},
{"visitName":"SCR","deleted":false,"tableRowInstance":null,"branchName":null,
"eventType":"ScreeningVisit","formRepeatNumber":3,
"value":"2022-03-09T06:00:00.000Z","cycleNumber":null,"empty":false,"treatmen
tArm":null}]
```

Detect missing data

Use this example of a custom Javascript rule when you want to find missing data.

- Search and detect missing values
  Verify collected data and raise a query any time a data field is found incomplete.

Search and detect missing values

Verify collected data and raise a query any time a data field is found incomplete.

```
if ( (ae.search("Yes") >0) && (seriousnessCriteria === null)) {
  return false;
} else {
  return true;
}
```

If the adverse event is serious, then the seriousness criteria must be filled.

Multiple choice question functions

Modify a value in a multiple-choice type of question.

- Deprecated - getArrayFromDropdown( )
  Convert the selected drop-down (choice) labels into an array.
- Deprecated - getStringFromDropdown( )
  Convert selected drop-down (choice) labels into a comma-separated string.
- setChoiceLabel( )
  Use this helper function in a calculated rule to add a selection to an existing choice (drop-down, radio button, or checkbox).
- setChoiceValue( )
  Use this helper function in a calculated rule to add a value to an existing choice (drop-down, radio button, or checkbox).
- clearChoice( )
  Use this helper function to clear values in multiple choice type questions.
- getArrayFromChoice( )
  Convert the selected choice labels (dropdown, radiobuttons, checkboxes) into an array.
- getStringFromChoice( )
  Convert selected choice labels (dropdown, radiobuttons, checkboxes) into a comma-separated or other string.

Deprecated - getArrayFromDropdown( )

Convert the selected drop-down (choice) labels into an array.

> **Note:**
>
> This function continues to work for drop-downs, However, consider using getArrayFromChoice( ) which supports all choice-type controls.

Syntax

```
getArrayFromDropdowm(variable)
```

Parameters

variable
Drop-down variable from the rules editor.

Return value

Returns an empty array, if no values are selected or an array of selected drop-down labels.

Example 2-63 Given a drop-down dd with labels "Yes" and "No" selected

```
// Return the first selected label from dropdown item dd:
return getArrayFromDropdown(dd2)[0];
// returns "Yes"

// Return the second selected label from dropdown item dd:
return getArrayFromDropdown(dd2)[1];
// Returns "No"
```

Deprecated - getStringFromDropdown( )

Convert selected drop-down (choice) labels into a comma-separated string.

> **Note:**
>
> This function continues to work for drop-downs. However, consider using getStringFromChoice( ) which supports all choice-type controls.

Syntax

```
getStringFromDropdown(variable)
```

Parameters

variable
Drop-down variable from the rules editor.

Return value

Returns an empty string, if no labels are selected or a comma-separated string of selected drop-down labels.

Example 2-64 Given a drop-down dd with labels "Yes" and "No" selected

```
// return all selected labels from dropdown
return getStringFromDropdown(dd2);


// if single label is selected, returns "label1"
// If multiple labels are selected, returns "label1,label2"
```

setChoiceLabel( )

Use this helper function in a calculated rule to add a selection to an existing choice (drop-down, radio button, or checkbox).

The expression creates a string JSON value that must be returned to the target control and must be used in combination with clearChoice( ).

Syntax

```
setChoiceLabel(labelStr, variable)
```

Parameters

labelStr
Label string.

variable
Choice variable from the rule editor.

Return value

Returns an empty JSON object string or a JSON object array string of selected choice labels.

Example 2-65 Given a drop-down (choice) control with multiple labels including "Allergies" and "Obesity" as the target of the calculation rule

```
// Select "Allergies"
if (someCondition) {
    return setChoiceLabel("Allergies");
} else {
    return clearChoice();
}
// selects "Allergies" in the calculated control

// Select "Allergies" and "Obesity"
var b;
if (someCondition) {
    b = setChoiceLabel("Allergies");
    return setChoiceLabel("Obesity", b);
} else {
    return clearChoice();
}
// selects "Allergies" and "Obesity" in the calculated control
```

setChoiceValue( )

Use this helper function in a calculated rule to add a value to an existing choice (drop-down, radio button, or checkbox).

The expression creates a string JSON value that must be returned to the target control and must be used in combination with clearChoice( ).

Syntax

```
setChoiceValue(valueStr, variable)
```

Parameters

valueStr
Value string.

variable
Choice variable from the rule editor.

Return value

Returns an empty JSON object string or a JSON object array string of selected choice values.

Example 2-66 Given a drop-down (choice) control with multiple labels including "Allergies" and "Obesity" with values "4" and "45" respectively, as the target of the calculation rule

```
// Select label "Allergies" having value "4"
if (someCondition) {
    return setChoiceValue("4");
} else {
    return clearChoice();
}
// selects "Allergies" in the calculated control

// Select "Allergies" having value "4" and "Obesity" having value "32"
var b;
if (someCondition) {
    b = setChoiceValue("4");
    return setChoiceValue("32", b);
} else {
    return clearChoice();
}
// selects "Allergies" and "Obesity" in the calculated control
```

clearChoice( )

Use this helper function to clear values in multiple choice type questions.

Syntax

```
clearChoice()
```

Parameters

None.

Return value

Returns an empty JSON object string.

Example 2-67 Given a drop-down (choice) control, clear the control

```
// Clear a dropdown control. Must be returned to clear the control.
return clearChoice();
// clears the target calculated control
```

getArrayFromChoice( )

Convert the selected choice labels (dropdown, radiobuttons, checkboxes) into an array.

Syntax

```
getArrayFromChoice(variable)
```

Parameters

variable
Choice variable from rule editor.

Return value

- Empty array if nothing is selected.
- An array of the selected choice labels.

Example 2-68 Given a dropdown (choice) control d2 with the labels "Yes" and "No" selected

```
// Return the first selected label from choice item dd2:
returngetArrayFromChoice(dd2)[0];
// returns "Yes"

// Return the second selected label from choice item dd2:
return getArrayFromChoice(dd2)[1];
// Returns "No"
```

getStringFromChoice( )

Convert selected choice labels (dropdown, radiobuttons, checkboxes) into a comma-separated or other string.

Syntax

```
getStringFromChoice(variable)
```

Parameters

variable
Choice variable from the rule editor.

Return value

- Empties string if no labels are selected.

- A comma-separated string of the selected choice labels.

Example 2-69 Given a dropdown (choice) control dd2 with labels "Yes" and "No" selected

```
// return all selected labels from choice
return getStringFromChoice(dd2);


// if single label is selected, returns "label1"
// If multiple labels are selected, returns "label1,label2"
```

Example 2-70 Convert a codelist term used as a coding target item into a string value

You can use an expression to convert a coding target using a choice question with a related specify text question.

For this example, you have designed a choice question with an option for **Other**. When the site user chooses **Other**, they are prompted to enter descriptive text into another question. This expression code allows you to combine predefined choice text and the **Other** specified text into a single question that is then tagged as the context item for coding. This is helpful because you can only tag a single question as the given context item.

```
if (ROUTE !== null)
  {
    return (ROUTESP === null ? getStringFromChoice(ROUTE) :
(getStringFromChoice(ROUTE) + ': ' + ROUTESP));
  }
else
  {
    return '';
  }
```

Multiple visit schedules and cycle visit functions

Control data collection on multiple visit schedules and cycles visits.

- getCurrentBranch( )
  Get the ID of the current branch.
- isSubjectOnBranch( )
  Check if a subject has started any visit in a specific branch.
- getCurrentTreatmentArm( )
  Retrieve the treatment arm short name that the current subject is on.
- getQuestionValue( )
  Return a single question value for provided item path. The item path should be a whole path and should contain values for **visitId**, **formId**, and **itemId**. This function fetches values from questions on both repeating and flat forms.
- getDataElementsArray( )
  Return an array of data element arrays that contain data collection information about all existing instances for each variable.
- getCurrentCycle( )
  Retrieve the current cycle instance number.

- getCycleCount( )
  Get the current cycle instance number per subject within the input branch of the current subject. For example, you can get the count of existing cycles.
- getCompletedCycle( )
  Retrieve the number of cycles that were completed by a subject.

getCurrentBranch( )

Get the ID of the current branch.

Syntax

```
getCurrentBranch()
```

Parameters

None.

Return value

Returns branch short ID of the current branch (string) or an empty string if the visit is not a branch visit.

> **Note:**
>
> If the branch name is changed between study versions, use the isStudyVersion( ) function to get the appropriate name.

Example 2-71 If the current branch is 'Branch01', set value of a dropdown to true

```
if (getCurrentBranch() == "Branch01") {
  return setChoiceLabel("TRUE");
} else {
  return setChoiceLabel("FALSE");
}
```

isSubjectOnBranch( )

Check if a subject has started any visit in a specific branch.

Syntax

```
isSubjectOnBranch(branchShortName)
```

Parameters

branchShortName
The branch short ID for the branch you are querying for.

Return value

Returns **true** if the branch with the short name *branchShortName* contains any visits where data has been entered or **false** if no visits are initialized in that branch.

> **Note:**
>
> If the branch name is changed between study versions, use the isStudyVersion( ) function to get the appropriate name.

Example 2-72 If subject is on Branch01, set value of text box to "Branch1"

```
var onBranch = isSubjectOnBranch("Branch1");
if (onBranch) {
  return "Branch1 has been started";
} else {
  return "Branch1 NOT started";
}
```

getCurrentTreatmentArm( )

Retrieve the treatment arm short name that the current subject is on.

Syntax

```
getCurrentTreatmentArm()
```

Return value

Returns the current treatment arm short name for the subject (string) or an empty string, if the treatment arm does not exist.

Example 2-73 Fetch the current treatment arm of the subject and return a value

```
if (getCurrentTreatmentArm()==="Placebo") {
    return "On Placebo" ;
} else {
    return "Not on Placebo" ;
}
```

getQuestionValue( )

Return a single question value for provided item path. The item path should be a whole path and should contain values for **visitId**, **formId**, and **itemId**. This function fetches values from questions on both repeating and flat forms.

All Visits cannot be used in the variable definition. If you'd like to fetch multiple values from a repeating form instance in a single function call, use thegetRFValues( ) function.

> **Note:**
>
> This rule does not support Visit Start Date items.
>
> This is aggregation function, the rule is run for each form instance in the case where the target is on a repeating form.

Syntax

```
getQuestionValue('ruleVariable', eventInstanceNumber,
formInstanceNumber, repeatFormNumber)
```

Parameters

ruleVariable
Variable (from the rule editor).

eventInstanceNumber
Event instance number/cycle instance number (optional). To reference non-repeating visits, pass an empty string.

formInstanceNumber
Form instance number (optional).To reference a non-repeating form, pass an empty string.

repeatFormNumber
Repeat form number (optional) to reference items on a two section form.

> **Note:**
>
> If optional parameters are missing, the current values for the instance numbers are used.

Return value

The question value, or null if there is no value.

- Dates: Returned variable value will be a C1Date object in case of a variable being a partial date, or it will be a Date object in the case of the variable being a full date.

  > **Note:**
  >
  > When using date items, a null check must be included. See the example for more information.

- If optional parameters are not provided, the event instance number and form instance number are taken from rule target context.
- If the variable is a choice control (checkbox, radio, or dropdown) then the returned variable value is a string in JSON format:

("`[{\"value\":\"3\",\"label\":\"TestLabel\"}]`"). This can be parsed using `JSON.parse` or the helper function `parseChoice`.

- `parseChoice(result))`
- `JSON.parse(result))`

Example 2-74 Using getQuestionValue in 2-section forms

To use `getQuestionValue` for items inside table rows of 2-section forms, the syntax is:

```
getQuestionValue('ruleVariable', eventInstanceNumber, repeatNumber,
formInstanceNumber)

// Get a value from an item in a flat form
return (getQuestionValue('text1');


// Get a value from an item in repeating form instance #1
// Pass an empty string to eventInstance
return (getQuestionValue('text1', '', 1);


// Get a value from a flat form in unscheduled visit instance #1
// Pass an empty string to eventInstance
return (getQuestionValue('text1', 1, '');


// Get a date value from a flat form
var res = getQuestionValue('dt1');
if(res !== null) {
  return res.getFullYear();
} else {
  // do nothing
  return;
}


//In a 2-section form, get a value from Form 2, Table Row 3:
return getQuestionValue('v2', '', 3, 2);
```

getDataElementsArray( )

Return an array of data element arrays that contain data collection information about all existing instances for each variable.

Return an array of data element arrays that contain data collection information about all existing instances for each variable.

Syntax

```
getDataElementsArray(var1, var2, ...)
```

Parameters

var1, var2, ...
Variables that are defined based on visits, forms, and items.

Return value

The rule returns an array of data element arrays with visit or branch short name.

Example 2-75 Rule with two variables: txt and num

```
var obj = getDataElementsArray(txt, num);
var result = "";

if(obj && obj.result)
{
    //list of dataelements for txt variable
    var txtPathObject = obj.result[0];
    //list of dataelements for num variable
    var numPathObject = obj.result[1];

    //dataelement value can be referenced through index
    //return txtPathObject[0].value + " --- " + numPathObject[0].value;

    //dataelement value can be referenced through forEach loop
    txtPathObject.forEach(function(txtVar) {
        result = result + ">>>" + txtVar.value;
    });

    /*var result = "";
    numPathObject.forEach(function(numVar) {
        result = result + ">>>" + numVar.value;
    });*/
}

return result;


var obj = getDataElementsArray(txt, num);
var result = "";

if(obj && obj.result)
{
    //list of dataelements for txt variable
    var txtPathObject = obj.result[0];
    //list of dataelements for num variable
    var numPathObject = obj.result[1];

    //access to dataelements properties for txt variable
    if(txtPathObject[0].visitShortName=='Visit1')
        //do something
    if(txtPathObject[0].visitType=='SCHEDULED') //visit type
        //do something
    if(txtPathObject[0].eventInstanceNum=='1') //cycle instance number
or unscheduled visit instance number
```

```
        //do something
    if(txtPathObject[0].repeatSequenceNumber=='1') //repeating form instance
number
        //do something
    if(txtPathObject[0].value=='Yes') //###user friendly value to be
implemented
        //do something
}

return result;
```

These types of JavaScript expressions can be used in Oncology Solid Tumor studies to sum up all the lesions prior to that visit and determine the lowest prior sum. Additionally, the rule can be used to check if a certain value exists in at least one visit or to compare values with the current visit, form, and so forth.

getCurrentCycle( )

Retrieve the current cycle instance number.

Syntax

```
getCurrentCycle()
```

Parameters

None.

Return value

Returns the current cycle instance number or **-1** if the target form is not on a cycle visit.

Example 2-76 If the current cycle instance is even (2, 4, 6, and so on), set the value of the dynamic form launch item to true

```
// get current cycle instance number
var currCycle = getCurrentCycle();

// if cycle instance is even, set value to true
if (currCycle > 1 && currCycle % 2 == 0) {
  return setChoiceLabel("TRUE");
} else {
  return setChoiceLabel("FALSE");
}
```

getCycleCount( )

Get the current cycle instance number per subject within the input branch of the current subject. For example, you can get the count of existing cycles.

Syntax

```
getCycleCount(branchShortName)
```

Parameters

branchShortName
The short name for the branch you want to count.

Return value

Returns the subject branch cycle instance number or **-1** if the subject is not provided on the branch.

> **Note:**
>
> If the branch name is changed between study versions, use the isStudyVersion( ) function to get the appropriate name.

Example 2-77 If the Branch01 has at least 1 started cycle, set the value of a drop-down to true

```
// get the current cycle count of branch 'Branch01'
var cycleCount = getCycleCount('Branch01');

// if at least 1 cycle has been started in Branch01, set value to true
if (cycleCount > 1) {
  return setChoiceLabel("TRUE");
} else {
  return setChoiceLabel("FALSE");
}
```

getCompletedCycle( )

Retrieve the number of cycles that were completed by a subject.

Syntax

```
getCompletedCycle(visitShortName)
```

Parameters

visitShortName
The branch short ID for the branch you are querying for.

Return value

Returns the number of cycles where the visit is in the Completed status (number) or **-1** if `visitShortName` is not a cycle visit.

> **Note:**
>
> If the branch name is changed between study versions, use the isStudyVersion( ) function to get the appropriate name.

Example 2-78 If 3 cycle visits for a subject have been completed, set value of a dynamic form launch item to true

```
if (getCompletedCycle("Vitals") == 3) {
  return setChoiceLabel("TRUE");
} else {
  return setChoiceLabel("FALSE");
}
```

Formatting and other functions

Format messages and queries and perform other useful operations.

- setQueryMessage( )
  Set a query message dynamically within a rule. This query message is used for query creation when the rule returns **false**.
- enableNotificationDetails( )
  Dynamically include or exclude the notification details in the notification email message. This function defaults to `yes`. If not specified in the rule expression, the details are included in the email message by default.
- getValues( )
  Use this helper function to fetch values for one or more variables across multiple visits, in an array format ordered by visits.

setQueryMessage( )

Set a query message dynamically within a rule. This query message is used for query creation when the rule returns **false**.

> **Note:**
>
> This function cannot be used for rules that spawn a notification.

You can use the getDateDMYFormat( ) helper function to format a date before passing it to this function.

Syntax

```
setQueryMessage(strMessage)
```

Parameters

strMessage
A string containing the query message. This string can be dynamically generated within the rule expression.

Return value

A string containing the query message that has been set for query creation. Or, an empty string if any errors occurred during the running of the function.

> **Note:**
>
> The dynamic query message is not set if its value is null, undefined, an empty string, or contains a space only. The default query message provided on rule creation is used when no dynamic query message is set upon running the rule.

Example 2-79 Set the query message when weight is less than 120

```
// Given "weight" item containing value of 110.
if (weight < 120){
var strMessage = "Subject weight of " + weight + " lb is less than the
required weight of 120 lb."
setQueryMessage(strMessage);
  return false; // create query
} else {

   return true;  // close query
}

// A query is created with message "Subject weight of 110 lb is less
than the required weight of 120 lb."
```

enableNotificationDetails( )

Dynamically include or exclude the notification details in the notification email message. This function defaults to `yes`. If not specified in the rule expression, the details are included in the email message by default.

The notification details include the following information, specific to the subject and target item containing the rule:

- Study Name
- Study Mode
- Site Name
- Subject Number
- Visit
- Form

- Question
- Sequence Number (repeating form instance number, if present)
- Date & Time of Notification

Syntax

```
enableNotificationDetails(option)
```

Parameters

option
Set to `'yes'` to include the notification details (default).
Set to `'no'` to exclude the notification details.

Return value

Returns null.

Example 2-80 Exclude the notification details from the notification email

```
// Exclude the notification details
enableNotificationDetails('no');

// The notification mail message is created without the notification header
```

getValues( )

Use this helper function to fetch values for one or more variables across multiple visits, in an array format ordered by visits.

> **Note:**
>
> This is an aggregation function. The rule is run for each form instance in the case where the target is on a repeating form.
>
> When used with a two-section form, GetCurrentRFInstance( ) can be used to restrict the rule to run only on the current two-section form instance.

Syntax

```
getValues('var1', 'var2', ...)
```

Parameters

'var1', 'var2', ...
Rule variable names (which define the visit, form, or item).

Return value

true on success; otherwise, **false**.

Also, during the call to this function, rule variables are redefined so that after this call, rule variables contain the following information:

- **null** - if there is no data elements related to this variable

  var1, var2, ... – hold an array with results, each item in the array contains the following var1[0].

  - **var1[0].visitName** – the name of the visit
  - **var1[0].cycleNumber** – the cycle number of the visit in case it is in a cycle
  - **var1[0].formRepeatNumber** – the repeating form instance number in case form is a repeating form
  - **var1[0].branchName** – name of branch
  - **var1[0].eventType** – type of visit - ScheduleAbleVisit, UnScheduleAbleVisit, Event, and so forth
  - **var1[0].treatmentArm** – the treatment arm
  - **var1[0].empty** – true if value was cleared or never entered in repeating form
  - **var1[0].deleted** – true if repeating form instance is deleted
  - **var1[0].tableRowInstance** – the repeating table row instance in case it is within a table

  value – data element value, or **null** if values were cleared

- **Dates**: The returned variable value is a C1Date object in the case where a variable is a partial date, or a Date object in the case where the variable is the full date. For partial dates, values are returned as an object and must be fetched using date parts such as `item.value.day`, `item.value.month`, and so forth instead of just `item.value`.

```
// Sample JSON response for a partial date item:
[
    {
        "visitName": "visit1",
        "deleted": false,
        "tableRowInstance": null,
        "branchName": null,
        "eventType": "ScheduleAbleVisit",
        "formRepeatNumber": null,
        "value": {
            "partialDate": true,
            "date": null,
            "day": "UNK",
            "month": 2,
            "year": 2021
        },
        "cycleNumber": null,
        "empty": false,
        "treatmentArm": null
    }
]
```

- If the variable is a choice control (checkbox, radio, or dropdown) then the returned variable value is a string in JSON format:

("[{\"value\":\"3\",\"label\":\"TestLabel\"}]"). This can be parsed using JSON.parse or parseChoice( ).

- parseChoice(result))
- JSON.parse(result))

Limitations

The amount of data which is returned by getValues( ) can be significant if the question exists in more than one visit. This can impact performance of the data submit.

> **Note:**
>
> Once you have passed a variable into getValues( ), it cannot be reused as a discrete value elsewhere in the rule. If you need to reference the discrete value for the row and visit elsewhere, you need to declare a second variable.

Example 2-81 View results of the getValues function call for a single item

```
// this can be helpful during rule development to view the results returned
by the getValues function
// using a read-only text field as the target

var val = getValues("item1");
if (val == true) {
    return JSON.stringify(item1);
}

/* example results:
[
    {
        "visitName": "visit1",
        "deleted": false,
        "tableRowInstance": null,
        "branchName": null,
        "eventType": "ScheduleAbleVisit",
        "formRepeatNumber": null,
        "value": "Test",
        "cycleNumber": null,
        "empty": false,
        "treatmentArm": null
    },
    {
        "visitName": "visit2",
        "deleted": false,
        "tableRowInstance": null,
        "branchName": null,
        "eventType": "ScheduleAbleVisit",
        "formRepeatNumber": null,
        "value": "Test",
        "cycleNumber": null,
        "empty": false,
```

```
        "treatmentArm": null
    }
]
*/
```

Example 2-82 Sum all tumor diameter values across all visits

```
var sumTotalLongestDiameter = -1;

function calculateTumor(item, index)
{
    if ( longestDiameter[index] !== null )
    {
        sumTotalLongestDiameter += longestDiameter[index].value;
    }
}

var rc = getValues("tumorID", "longestDiameter");

if ( rc === true )
{
    tumorID.forEach(calculateTumor);

    // Set the value for the current row where this rule runs
    return sumTotalLongestDiameter;
}
```

Example 2-83 Sum tumor diameter values across all visits, excluding the Screening visit by using a filter function

```
var rc = getValues("tumorID", "longestDiameter");

//filter by visit name
function filterFunction(item) {
  return item.visitName  !== 'Screening';
}

var sumTotalLongestDiameter = -1;

function calculateTumor(item, index)
{
    if ( longestDiameter[index] !== null )
    {
        sumTotalLongestDiameter += longestDiameter[index].value;
    }
}

if ( rc === true )
{
    //exclude Screening visit
    var filterResult = tumorID.filter(filterFunction);
    filterResult.forEach(calculateTumor);

    // Set the value for the current row where this rule runs
```

```
        return sumTotalLongestDiameter;
    }
```

3

Rules examples

Rule examples provide real world examples for rules using multiple helper functions. You can use these examples as the basis for your own custom rules.

While helper functions can be used alone for simple rules, most studies require complex rules that combine multiple functions to achieve the desired rule logic. Use the examples provided as-is or as a basis for a similar complex rule.

Electronic Data Collection (EDC) examples

Range check

Check if a given value is in the range or not.

Rule description: the Oral Temperature must be between 35-40.6 C or 95-105 F (inclusive).

Rule expression

```
if(tempval!==null)
{
```

```
if(getStringFromChoice(tempunit)==='C')
{
   if(tempval>=35.0 && tempval<=40.6)
   {
       return true;
   }
   else
   {
       setQueryMessage("The value entered for Oral Temperature is out
of range: 35-40.6 °C. Please confirm or correct.")
       return false;              //System sends query if return
false condition is met
   }
}
else
{
   if(getStringFromChoice(tempunit)==='F')
{
   if(tempval>=95.0 && tempval<=105.0)
   {
       return true;
   }
   else
   {
       setQueryMessage("The value entered for Oral Temperature is out
of range: 95-105 F. Please confirm or correct.")
       return false;              //System sends query if return
false condition is met
   }
}
else
{
   return true;
}
}
}
else
{
   return true;
}
```

Query message (Dynamic): The value entered for Oral Temperature is out of range: {tempRange}. Please confirm or correct.

Definitions

tempval
Corresponds to the *Temperature* from rule description.

tempunit
Corresponds to the *Temperature unit* from the rule description.

getStringFromChoice( )
Convert the label of the selected choice from a drop-down, radio button or check box into a string or comma-separated value. Takes in the question item variable as parameter.

setQueryMessage( )
Specify dynamic query text passed in as a parameter.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Verification steps

Test step	Test description	Result
1	Using a subject go to the visit <Visit refname> form <Form refname> and enter the item <unit item refname> as '**C**'.	No query
2	Go to the visit <Visit refname> form <Form refname> and enter the item <Item Refname> as 'x' (lower range for C).	No query
3	Update the item <Item Refname> as 'x-1'.	Query
4	Update the item <Item Refname> as 'x+1.'	No query
5	Update the item <Item Refname> as 'y' (higher range for C).	No query
6	Update the item <Item Refname> as 'y-1.'	No query
7	Update the item <Item Refname> as 'y+1.'	Query
8	Update the item <unit item refname> as '**F**'.	Query
9	Update the item <Item Refname> as 'a-1' (a is lower range for F).	Query
10	Update the item <Item Refname> as 'a.'	No query
11	Update the item <Item Refname> as 'a+1.'	No query
12	Update the item <Item Refname> as 'b'(higher range for F).	No query
13	Update the item <Item Refname> as 'b-1.'	No query
14	Update the item <Item Refname> as 'b+1.'	Query

Test step	Test description	Result
15	Update the item <Item Refname> as 'b-2.'	No query
16	Update the item <unit item refname> as '**C**'.	Query

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-1 The weight must be between 36.2-136.1 kg or 80-300 lbs (inclusive)

```
if (wtval!==null)
{
if (getStringFromDropdown(wtunit)==='kg')
{
    if (wtval>=36.2 && wtval<=136.1)
    {
        return true;
    }
    else
    {
        return false;
    }
}
else
{
    if (getStringFromDropdown(wtunit)==='lb')
{
    if (wtval>=80.0 && wtval<=300.0)
    {
        return true;
    }
    else
    {
        return false;
    }
}
else
{
    return true;
}
}
}
else
{
```

```
    return true;
}
```

Query message: The value entered for Weight is out of range. Please confirm or correct.

Item completion check

Check that an item has been completed.

Rule description: evaluate that Reason for Withdrawal is not null when the Date of Discontinuation is provided.

Rule expression

```
if(dt !== null && reason === null)
{
   return false;               //System sends query when return false
condition is met
}
else
{
   return true;
}
```

Query Message: Date of Discontinuation is provided but Reason for Withdrawal is missing. Please verify.

Definitions

dt
Corresponds to the *Date of Discontinuation* from the rule description.

reason
Corresponds to the *Reason for Withdrawal* from the rule description.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Verification steps

In the following verification steps for the given rule expression, we use *<item1>* that refers to Date of discontinuation and *<item2>* that is Reason for withdrawal.

Test step	Test description	Result
1	Using a subject go to the visit <Visit refname> form <Form refname> and complete the item <item1 refname>.	Query
2	Go to the visit <Visit refname> form <Form refname> and complete the item <item2 refname>.	No query

Test step	Test description	Result
3	Clear the item <item2 refname>.	Query
4	Complete the item <item2 refname> (any value other than selected in Step2).	No query
5	Clear the item <item1 refname>.	No query
6	Clear the item <item2 refname>.	No query

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-2 Collected Date and Time is provided and Clinical Significance = null so a query is issued

```
if(VSDTTIM !== null)
{
    if(VSCLSIG !== null)
    {
        return true;
    }
    else
    {
        return false;
    }
}
else
{
    return true;
}
```

Query message: Collected Date and Time is provided but Clinical Significance is missing. Please verify.

BMI calculation check

Calculate a subject BMI.

Rule description: Calculate a BMI using the below formula:

BMI = Weight/Height * Height

The result has one decimal place (for example, **25.1**). The unit is kg/m2. If weight and height units are provided in pounds (lb) and centimeters or inches (cm or in), convert them into kilograms (kg) and meters (m).

Rule expression

```
if(hght===0||wght===0){
return 0;}
else{
     if(getStringFromChoice(hghtunt)=='cm'){
        hght=(hght*0.01);}
     else if(getStringFromChoice(hghtunt)=='in'){
        hght=(hght*0.0245);}
     else if(getStringFromChoice(wghtunt)=='lb'){
        wght=(wght*0.453);}
     return (wght/((hght)*(hght)));}
```

Definitions

wght
Corresponds to *Weight* from rule description.

hght
Corresponds to *Height* from the rule description.

hghtunt
Corresponds to *Height unit* from the rule description.

wghtunt
Corresponds to *Weight unit* from the rule description.

Return value

Number
Returns a calculated numerical value rounded according to the target item format. In this case one decimal place, for example **21.5**.

Verification steps

Test step	Test description	Result
1	Using a subject; go to test visit <Visit refname> test form <Form refname>] and enter the items <Height value item refname> and <Height unit item refname> as '**175**' & '**cm**' respectively.	No Calculated Value
2	Enter the items <Weight value item refname> and <Weight unit item refname> as '**50.0**' & '**kg**'.	16.3
3	Update item<Weight unit item refname> as '**lb**'.	7.4
4	Update item <Weight value item refname> as '**78.0**'.	11.5
5	Update item <Weight unit item refname> as '**kg**'.	25.5

Test step	Test description	Result
6	Update item <Height value item refname> as '**72**'	150.5
7	Update item <Height unit item refname> as '**in**'.	23.3
8	Clear item <Height value item refname> as '**0.0**'.	0

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Oracle Central Coding mapping

Perform mapping on for Central Coding questions.

Rule description: When a codelist value is mapped to Oracle Central Coding and includes 'Other' as an option for a field (ROUTE), user will be requested to specify in another text field (ROUTEOTHR), then we will map the specified text to the codelist value in the following format: "Other: {ROUTEOTHR}".

Rule expression

```
if (ROUTE !== null)
 {
    return (ROUTEOTHR === null ? getStringFromChoice(ROUTE) :
(getStringFromChoice(ROUTE) + ': ' + ROUTEOTHR));
 }
else
 {
    return '';
 }
```

Definitions

ROUTE
Corresponds to the mapped codelist value item *(ROUTE)* from the rule description.

ROUTEOTHR
Corresponds to the specified text item *(ROUTEOTHR)* that needs to be mapped along the 'Other' codelist value from the rule description.

Return value

String
The route item is a choice control and the rule is mapping data entered in choice control to a text item.

Verification steps

Test step	Test description	Result
1	Using a subject; go to test visit <Visit refname> test form <Form refname>] and Select the item <item refname> as '**Oral**'.	Verify the target item is populated as '**Oral**'.
2	Update the item <item refname> as '**topical**'.	Verify the target item is populated as '**topical**'.
3	Update the item <item refname> as '**IM**'.	Verify the target item is populated as '**IM**'.
4	Update the item <item refname> as '**Other**' .	Verify the target item is populated as '**Other**'.
5	Update the item <item refname> as '**Other**' and specify <item2 refname> as '**unknown**'.	Verify the target item is populated as '**Other: unknown**'.
6	Clear the item <item refname>.	Verify target item is cleared.

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-3 Details from Route and Route Other specify should be mapped to Route of Administration

> **Note:**
>
> If CMROUTE = Other and CMROUTEOTH = null then NO VALUE will be populated in Route of Administration

```
if(CMROUTE===null || cmrouteoth===null){}
var txt=getStringFromChoice(CMROUTE);
if(txt==='Other')
{
   return cmrouteoth!==null? cmrouteoth : "NO VALUE";
}
else if(txt!==''){
    return txt;
}else
{
    return '';
}
```

Choice question check

Check the value of a choice question.

Rule description: if injection site is "Other" for Study Vaccine Administration, issue a query.

Rule expression

```
if(getStringFromChoice(INJSITELOC)==='Other')
{
   return false;               //System sends query when return false
condition is met
}
else
{
   return true;
}
```

Query Message: Potential Protocol Deviation: The Injection is not administered in a recomended muscle. Please reconcile or complete Protocol Deviation CRF.

Definitions

INJSITELOC
Corresponds to *Injection Site* choice question from the rule description.

getStringFromChoice( )
Convert the label of the seleceted choice from a drop-down, radio button or check box into a string or comma-separated value. Takes in the question item variable as parameter.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Usage tips

Use this only for choice question types.

Verification steps

Test step	Test description	Result
1	Using a subject go to the visit <Visit refname> form <Form refname> and select item <item refname> as '**Other**'.	Query
2	Update item <item refname> as a value other than 'Other'.	No query
3	Update item <item refname> as '**Other**'.	Query

Test step	Test description	Result
4	Clear item <item refname>.	No query
5	Update item <item refname> as a value other than 'Other' and different to the one selected in step 2..	No query

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-4 If Stop Date is present, then Outcome must be Recovered/Resolved, Recovered/Resolved with Sequelae, or Fatal

```
if(stpdt!==null)
{
   if(getStringFromChoice(outcm).contains('Recovered/Resolved') ||
getStringFromChoice(outcm).contains('Recovered/Resolved with Sequelae') ||
getStringFromChoice(outcm).contains('Fatal'))
   {return true;}
   else{return false;}
}
else
{    return true;}
```

Query message: You have entered a Stop Date but the Outcome is not RECOVERED/ RESOLVED, RECOVERED/RESOLVED WITH SEQUELAE, or FATAL. Please change the Outcome or remove the Stop Date.

Example 3-5 If Were Height and Weight collected? on VS form is No, issue a query

```
if (getStringFromChoice(VSYN)==='No')
{
    return false;
}
else
{
    return true;
}
```

Query message: Potential Protocol Deviation: Height and/or Weight was/were not assessed as schedule at Screening. Please reconcile or complete Protocol Deviation CRF.

Example 3-6 If Standard Toxicity Grade is Grade 4 or Grade 5, then Is AE Serious? must be Yes

```
if(getStringFromChoice(toxicity).contains('Grade 4') ||
getStringFromChoice(toxicity).contains(Grade 5'))
```

```
   {
      if(getStringFromChoice(aeser)==='Yes')
      {return true;}
      else{return false;}
  }
else
{    return true;}
```

Query message: Standard Toxicity Grade is selected as either Grade 4 or Grade 5. Please assess whether this AE meets seriousness criteria. If no, please confirm. If yes, please change Is AE serious? to Yes and report SAE.

Example 3-7 If Outcome is Fatal then the answer to Is AE Serious? must be Yes

```
if(getStringFromChoice(outcm).contains('Fatal'))
   {
      if(getStringFromChoice(aeser)==='Yes')
      {return true;}
      else{return false;}
  }
else
{    return true;}
```

Query message: Outcome is FATAL, but Is AE Serious? is No. Please correct outcome or seriousness.

Example 3-8 If Hypersensitivity Reaction Term is Other then (Other) Specify must be completed

```
 if(getStringFromChoice(reacterm).contains('Other'))
   {
      if(othspec!==null)
      {return true;}
      else{return false;}
  }
else
{    return true;}
```

Query message: Other is selected; however the (Other) Specify field is blank. Please correct or clarify.

Example 3-9 Fire Query if Pregnancy Test is Positive

```
if(getStringFromChoice(pregtest)==='Positive')
   {
     return false;
   }
    else
    {
     return true;
    }
```

Query message: Pregnancy Test Result is recorded as Positive. If this is correct, please report immediately to the Sponsor Safety Team.

Blood pressure comparison check

Compare systolic and diastolic blood pressure values.

Rule description: Systolic Blood Pressure must be greater than the corresponding Diastolic Blood Pressure.

Rule expression

```
if (SYS>DIA)
{
return true;
}
else
{
return false;                //System sends query when return false
condition is met
}
```

Query Message: Systolic Blood Pressure is less than or equal to Diastolic Blood Pressures. Please correct or confirm.

Definitions

SYS
Corresponds to *Systolic Blood Pressure* item from rule description.

DIA
Corresponds to *Diastolic Blood Pressure* from the rule description.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Verification steps

Test step	Test description	Result
1	Using a subject go to the visit <Visit refname> form <Form refname> and enter the item <Systolic item refname> as '**120**'.	No query
2	Go to the visit <Visit refname> form <Form refname> and enter the item <Diastolic item refname> as '**120**'.	Query
3	Update the item <Diastolic item refname> as '**119**'.	No query
4	Update the item <Diastolic item refname> as '**121**'.	Query

Test step	Test description	Result
5	Update item <Systolic item refname> as '**123**'.	No query
6	Update item <Systolic item refname> as '**115**'.	Query
7	Clear the item <Systolic item refname>.	No query
8	Update the item <Systolic item refname> as '125.'	No query

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Format check

Check the format of a question.

Rule description: the Subject Initials value must be 3 characters or 2 characters with a dash in place of the middle initial. No numbers, spaces, or special characters are allowed.

Rule expression

```
var str=txtitem1.toUpperCase();
if(str.length==3 && (str.match("^([A-Z]){3}$") || str.match("^(([A-Z])
[-]([A-Z]))$")))
{
   return true;
}
else
{
   return false;               //System sends query if the return
false condition is met
}
```

Query Message: Value is not recorded in the required format of 3 charactes or 2 with a dash in place of the middle initial

Definitions

txtitem1
Question or Item for which you want to check the format, *Subject Initials* from the rule description.

.toUpperCase()
JavaScript method for string objects to convert a string in all upper cases.

.match()
JavaScript method for string objects to check a string value against a regular expression which returns an array of matches.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Verification steps

In the following verification steps for the given rule expression, we use *<item>* that refers to subject initials.

Test step	Test description	Result
1	Using a subject go to the visit <Visit refname> form <Form refname> and enter the item <item refname> as '**ABC**'.	No query
2	Update the item <item refname> as '**abc**'.	No query
3	Update the item <item refname> as '**AbC**'.	No query
4	Update the item <item refname> as '**A-b**'.	No query
5	Update the item <item refname> as '**A-A**'.	No query
6	Update the item <item refname> as '**a-z**'.	No query
7	Update the item <item refname> as '**A**'.	Query
8	Update the item <item refname> as '**AB**'.	Query
9	Update the item <item refname> as '**AB**'.	Query
10	Update the item <item refname> as '**A_B**'.	Query
11	Update the item <item refname> as '**123**'.	Query
12	Update the item <item refname> as '**A13**'.	Query
13	Update the item <item refname> as '**AB@**'.	Query
14	Update the item <item refname> as '**AB$**'.	Query
15	Update the item <item refname> as '**AB!**'.	Query
16	Update the item <item refname> as '**AB&**'.	Query

Test step	Test description	Result
17	Update the item <item refname> as '**A B**'.	Query
18	Update the item <item refname> as '**Abc**'.	No query

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-10 The 'Kit Number:' must be 5 digits

```
var wk2num=KITNUM.toString();
if(wk2num.length==5)
{
    return true;
}
else
{
    return false;
}
```

Query message: Kit number does not meet the requirements (kit number must be 5 digits). Please correct or clarify.

Age calculation check

Calculate an age using Informed Consent and Date of Birth.

Rule description: calculate age using Date of Informed Consent Signed and Date of Birth.

Rule expression

```
//Returns the age value as the difference infconst-dob
return dateDiffInYears(infconst,dob);
```

Definitions

infconst
Corresponds to the *Informed Consent* item form rule description.

dob
Corresponds to the *Date of Birth* item from rule description.

Return value

Number
Returns a calculated numerical value with the difference in years between the two given dates.

Verification steps

Test step	Test description	Result
1	Using a subject go to test visit <visit refname> test form <form refname> enter Date of Birth <Item refname> as '**2-Jan-1942** ' and enter Inform Consent Date <Item refname> as '**2-Jan-2021**'.	Verify that the test item gets populated with the value 79
2	Change Date of Birth <Item refname> to '**3-Jan-1942**'.	Verify that the test item gets populated with the value 78..
3	Change Date of Birth <Item refname> to '**1-Jan-1942**'.	Verify that the test item gets populated with the value 79.
4	Change Date of Birth <Item refname> to '**1-Jan-1993**'.	Verify that the test item gets populated with the value 28.
5	Change Date of Birth <Item refname> to '**3-Jan-1993**'.	Verify that the test item gets populated with the value 27.
6	Change Inform Consent Date <Item refname> to '**3-Jan-2021**'.	Verify that the test item gets populated with the value 28.
7	Clear Inform Consent Date <Item refname>.	Verfiy that the test item is blank.

Date examples

- Date comparisons
- Partial date comparisons
- Dates with Dynamic Query Text

Date comparisons

- Date comparison
  Compare two date questions that do not have fields for an exact time (hour and minutes) and raise a query if the dates for those questions are not as expected.
- DateTime comparison
  Compare two date questions that also contain time fields (hour and seconds), and raise a query if the dates are not as expected.
- Date comparison within range: On or after
  Check if one date is the same, or a number of days after (inclusive) another date, and raise a query if the date is outside of this window.

- Date comparison within range: Days before
  Check if one date is within a number of days prior to another date (inclusive) and raise a query if the date is outside of this window.
- Map dates
  Map a date question that has time elements to a read-only question.

Date comparison

Compare two date questions that do not have fields for an exact time (hour and minutes) and raise a query if the dates for those questions are not as expected.

Rule description: the Onset Date value must be on or before the Date of Completion value, or else a query is raised.

Rule expression

```
//to meet the rule description criteria onstdt-compdt should be a
negative value or zero (<=0)
if(dateDiffInDays(onstdt,compdt)<=0)
{
return true;
}
else
{
return false;                          //System sends query when return
false condition is met
}
```

Query Message: The Onset Date is after the Date of Completion. Please correct or confirm the date(s).

Definitions

onstdt
Corresponds to the *Onset Date* from the rule description.

compdt
Corresponds to the *Date of Completion* from the rule description.

<=
Less Than or Equal To operator. Update the operator based on the rule description.

dateDiffInDays
Calculates the difference between *date1* (`onstdt`) and *date2* (`compdt`) (*date1-date2*) in days.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Usage tips

Always use the relevant date helper function to compare dates rather than directly comparing the variables using comparison operators.

Verification steps

In the following verification steps for the given rule expression, we use <*date1*> that refers to Onset Date and <*date2*> that is Date of Completion.

Test step	Test description	Result
1	Using a subject, go to the visit <*Visit refname*> form <*form refname*> and enter the item <*date1 item refname*> as '**10-May-2021**'.	No query
2	Go to the visit <*Visit refname*> form <*Form refname*> and enter the item <*date2 item refname*> as '**10-May-2021**'.	No query
3	Update the item <*date1 item refname*> as '**11-May-2021**'.	Query
4	Update the item <*date1 item refname*> as '**09-May-2021**'.	No query
5	Update the item <*date1 item refname*> as '**09-Jun-2021**'.	Query
6	Update the item <*date1 item refname*> as '**11-Apr-2021**'.	No query
7	Clear the item <*date1 item refname*>.	No query
8	Update the item <*date1 item refname*> as '**12-May-2021**'.	Query
9	Update the item <*date2 item refname*> as '**14-May-2021**'.	No query

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-11 Date of Death must be greater than or equal to the Date of Randomization

```
if(dateDiffInDays(deathdt,randt)>=0)
{
return true;
}
else
{
return false;
}
```

Query message: The Date of Death is prior to the Randomization Date. Please correct or confirm the date(s).

Example 3-12 The Date of Completion or the Withdrawal Date must be equal to the Date of Death

```
//Apply this rule when Reason for discontinuation is 'death'
if(dateDiffInDays(compdt,deathdt)===0)
{
return true;
}
else
{
return false;
}
```

Query message: Reason for discontinuation is death, the Date of Completion or the Withdrawal Date must equal the Date of Death. Please correct or confirm the date(s).

Example 3-13 Start date of hypoglycaemic episode must be <== Date of subject withdrawal

Note: The hypoglycaemic episode is a date and time question, time elements are ignored in this logic.

```
if(dateDiffInDays(hypodt,withdrawdt)<=0)
{
return true;
}
else
{
return false;
}
```

Query message: The date of the hypoglycaemic episode is after the subject ended the trial. Please correct or clarify.

Example 3-14 Medical History Start Date must be on or after Date of Birth

```
if(dateDiffInDays(mhstdt,dob)>=0)
{
return true;
}
else
{
return false;
}
```

Query message: Start Date is before the Date of Birth on the Demographics form. Please correct the date(s).

DateTime comparison

Compare two date questions that also contain time fields (hour and seconds), and raise a query if the dates are not as expected.

Rule description: the End Date and the Time must be on or after to the Start Date and Time in the Administration form.

Rule expression

```
//to meet the rule description criteria enddt-stdt should be a positive
value or zero (>=0)
if(dateDiffInMinutes(enddt, stdt)>=0)
{
     return true;
}
else
{
     return false;                    //System sends query when return false
condition is met
}
```

Query Message: The End Date and Time is before the Start Date and Time. Please correct or confirm the date(s).

Definitions

enddt
Corresponds to the *End Date* from the rule description.

stdt
Corresponds to the *Start Date* from the rule description.

>=
Greater Than or Equal To operator. Update operator based on the rule description.

timeDiffInMinutes( )
Calculates the difference between date1 (`enddt`), date2 (`stdt`) (date1-date2) in minutes.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Usage tips

Always use the relevant date helper function to compare dates rather than comparing the variables directly using comparison operators.

Verification steps

In the following verification steps for the given rule expression, we use *<date1>* that refers to End Date and *<date2>* that is the Start Date.

Test step	Test description	Result
1	Using a subject go to the visit <*Visit refname*> form <*form refname*> and enter the item <*date2 item refname*> as '**10-May-2021 11:00 AM**'.	No query
2	Go to the visit <*Visit refname*> form <*Form refname*> and enter the item <*date1 item refname*> as '**10-May-2021 11:00 AM**'.	No query
3	Update the item <*date2 item refname*> as '**10-May-2021 11:01 AM**'.	Query
4	Update the item <*date2 item refname*> as '**10-May-2021 10:59 AM**'.	No query
5	Update the item <*date2 item refname*> as '**11-May-2021 10:59 AM**'.	Query
6	Update the item <*date2 item refname*> as "**09-May-2021 10:59 AM**".	No query
7	Clear the item <*date2 item refname*>.	No query
8	Update the item <*date2 item refname*> as '**10-May-2021 11:00 PM**'.	Query
9	Update the item <*date1 item refname*> as '**10-May-2021 11:05 PM**'.	No query

Note:

Repeat the above steps if the form is present in multiple visits.

Date comparison within range: On or after

Check if one date is the same, or a number of days after (inclusive) another date, and raise a query if the date is outside of this window.

Rule description: the Date of Study Completion must be on or within 30 days after the V5C Visit Date.

Rule expression

```
//to meet the rule description criteria DSENDT1-VISDAT should be
between 0 and 30 (inclusive)
//so greater than or equal to 0 (>=0) AND less than or equal 30 (<=30)
```

```
if(dateDiffInDays(DSENDT1,VISDAT)>=0 && dateDiffInDays(DSENDT1,VISDAT)<=30)
{
    return true;
}
else
{
    return false;                    //System sends query when return false
condition is met
}
```

Query message: Date of Study Completion is prior to, or not within 30 days of, V5C DOV. Please verify.

Definitions

DSENDT1
Corresponds to the *Date of Study Completion* from the rule description.

VISDAT
Corresponds to the *Visit Date* from the rule description.

>=, <=
Greater Than or Equal To and *Less Than or Equal To* operators. Update operator based on the rule description.

dateDiffInDays
Calculates difference between *date1* (`DSENDT1`), *date2*(`VISDAT`) (*date1-date2*) in days.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Usage tips

Always use the relevant date helper function to compare dates rather than comparing the variables directly using comparison operators.

Verification steps

In the following verification steps for the given rule expression, we use *<date1>* that refers to Date of Study Completion and *<date2>* that is the 'V5C' Visit Date.

Test step	Test description	Result
1	Using a subject go to the visit *<Visit refname>*; form *<Form refname>* and enter the item *<date2 item refname>* as '**10-May-2021**'.	No query
2	Go to the visit *<Visit refname>*; form *<Form refname>* and enter the item *<date1 item refname>* as '**10-May-2021**'.	No query
3	Update the item *<date1 item refname>* as '**10-Jun-2021**'.	Query

Test step	Test description	Result
4	Update the item <*date1 item refname*> as '**09-Jun-2021**'.	No query
5	Update the item <*date1 item refname*> as '**10-May-2022**'.	Query
6	Update the item <*date1 item refname*> as '**11-May-2021**'.	No query
7	Update the item <*date2 item refname*> as '**05-May-2022**'.	Query
8	Update the item <*date2 item refname*> as '**11-May-2021**'.	No query
9	Clear the item <*date2 item refname*>.	No query
10	Update the item <*date2 item refname*> as '**06-May-2022**'.	Query

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-15 'Collection Date' must be within 30 days of 'Date Initial Informed Consent Obtained'

```
if(dateDiffInDays(COLLDT,INFCNST)>=0 &&
dateDiffInDays(COLLDT,INFCNST)<=30)
{
    return true;
}
else
{
    return false;
}
```

Query message: Collection Date is not within 30 days of Date of Initial Informed Consent Obtained. Please Verify.

Date comparison within range: Days before

Check if one date is within a number of days prior to another date (inclusive) and raise a query if the date is outside of this window.

Rule description: the Date of Measurement must be 1 to 28 days prior (inclusive) to the Day 1 visit start date.

Rule expression

```
//to meet the rule description criteria DOV-MEASDT should be between 1 and
28 (inclusive)
//so greater than or equal to 1 (>=1) AND less than or equal 28 (<=28)
if(dateDiffInDays(DOV,MEASDT)<=28 && dateDiffInDays(DOV, MEASDT)>=1)
{
return true;
}
else
{
return false;                    //System sends query when return false
condition is met
}
```

Query message: Date of Measurement at Screening visit was not taken within -28 to -1 days prior to Day 1. Please verify the dates.

Definitions

DOV
Corresponds to *Day 1 Visit Start date* from the rule description.

MEASDT
Corresponds to *Date of Measurement* from the rule description.

<=, >=
Less Than or Equal To and *Greater Than or Equal To* operators. Update operator based on the rule description.

dateDiffInDays
Calculates difference between *date1* (`DOV`) and *date2* (`MEASDT`) (*date1-date2*) in days.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Usage tips

Always use the relevant date helper function to compare dates rather than comparing the variables directly using comparison operators.

Verification steps

In the following verification steps for the given rule expression, we use *<date1>* that refers to Day 1 Visit Start Date and *<date2>* that is the Date of Measurement.

Test step	Test description	Result
1	Using a subject go to the visit *<Visit refname>*; form *<Form refname>* and enter the item *<date2 item refname>* as '**10-May-2021**'.	No query

Test step	Test description	Result
2	Go to the visit <*Visit refname*>; form <*Form refname*> and enter the item <*date1 item refname*> as '**10-May-2021**'.	Query
3	Update the item <*date1 item refname*> as '**11-May-2021**'.	No query
4	Update the item <*date1 item refname*> as '**09-May-2021**'.	Query
5	Update the item <*date1 item refname*> as '**12-Apr-2021**'.	Query
6	Update the item <*date2 item refname*> as '**11-Apr-2021**'.	No query
7	Update the item <*date2 item refname*> as '**12-Apr-2021**'.	Query
8	Update the item <*date2 item refname*> as '**14-Mar-2021**'.	Query
9	Enter the item <*date2 item refname*> as '**15-Mar-2021**'.	No query
10	Enter the item <*date1 item refname*> as '**13-Apr-2021**'.	Query
11	Clear the item <*date2 item refname*>.	No query

Note:

Repeat the above steps if the form is present in multiple visits.

Map dates

Map a date question that has time elements to a read-only question.

Rule description: Use this rule to map *DateTime2* in *form2* onto *DateTime1* in *form1*.

Rule expression

```
//variable declaration
var
fullmnth=["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","
Nov","Dec"];
var mnth=dt2.getMonth();                          //gets numeric value
of the month and holds it in a variable

//RETURN statement, returns date in DD-Mon-YYYY HH:MM format
return pad2(dt2.getDate())+"-"+fullmnth[mnth]+"-"+dt2.getFullYear()+"
"+pad2(dt2.getHours())+":"+pad2(dt2.getMinutes()
);
```

```
//defined function to return a two-digits numerical value, adds leading '0'
when required
function pad2(number) {
   return (number < 10 ? '0' : '') + number;
}
```

Definitions

dt2
Corresponds to *DateTime2* in rule description.

.getMonth()
JavaScript method for date type elements. Retrieves the numeric value of the month in a date, for example '11' as the numeric value of Novemeber in '01-Nov-2021 15:03'.

.getDate()
JavaScript method for date type elements. Retrieves the numeric value of the day in a date, for example '1' in '01-Nov-2021 15:03'.

.getFullYear()
JavaScript method for date type elements. Retrieves the numeric value of the year in a date, for example '2021' in '01-Nov-2021 15:03'.

.getHours()
JavaScript method for date-time type elements. Retrieves the numeric value of the hours in the time component a date, for example '15' in '01-Nov-2021 15:03'.

.getMinutes()
JavaScript method for date-time type elements. Retrieves the numeric value of the minutes in a date, for example '03' in '01-Nov-2021 15:03'.

pad2(*number*)
Function defined in code. Takes a number type element as parameter and returns a two-digit numerical value, adding a leading zero when the passed-in *number* is less than 10.

Return value

Date
Returns a date (including a partial date) in **DD-Mon-YYYY HH:MM** format by passing in:

- **DD (day value):** uses the JavaScript `getDate( )` method passed into the `pad2( )` function that ensures a leading zero is appended where required to ensure a two-digit numerical value is returned.
- **- (separator):** appends a hyphen "-" in string format.
- **Mon:** uses the JavaScript `getMonth( )` method to return a number that represents the month of the date (0 to 11) into a new variable `mnth`. This variable is used as an index for the `fullmnth` array to return the month as a three-letter abbreviation. For example, `Apr`.
- **- (separator):** appends a hyphen "-" in string format.
- **YYYY (Year value):** uses the JavaScript `getFullYear()` method.
- appends a blank space " ".

- **HH (hours value):** uses the JavaScript `getHours( )` method passed into the `pad2( )` function that ensures a leading zero is appended where required to ensure a two-digit numerical value is returned.
- **: (time elements separator):** appends a colon ":" in string format.
- **MM (minutes value):** uses the JavaScript `getMinutes( )` method passed into the `pad2( )` function that ensures a leading zero is appended where required to ensure a two-digit numerical value is returned.

> **Note:**
>
> `+" "+pad2(dt2.getHours())+":"+pad2(dt2.getMinutes())` is used to add time component. Exclude from the return statement if the date item does not include a time element.

Verification steps

Test step	Test description	Result
1	Using a subject go to the visit <*Visit refname*> form <*form refname*> and enter the <*item refname*> as '**30-Oct-2021 01:23**'.	Verify the target item is populated as '**30-Oct-2021 01:23**'.
2	Update the item <item reframe> as '**31-Oct-2021 23:59**'.	Verify the target item is populated as '**31-Oct-2021 23:59**'.
3	Clear item <*item refname*>	View the target item is cleared.

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-16 Map date of visit in status form as visit date

```
var
fullmnth=["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","
Nov","Dec"];
var mnth=dov.getMonth();
return pad2(dov.getDate())+"-"+fullmnth[mnth]+"-"+dov.getFullYear();

function pad2(number)
{
   return (number < 10 ? '0' : '') + number;
}
```

Partial date comparisons

- Partial date comparison
  Compare two date questions where at least one of the dates is a partial date and raise a query if the dates are not as expected.
- Partial date unknown month evaluation
  Check if the month of the date question is selected as unknown (UNK) and display a query if needed.

Partial date comparison

Compare two date questions where at least one of the dates is a partial date and raise a query if the dates are not as expected.

Rule description: the AE Start Date must be on or after the Date of Informed Consent.

> **Note:**
>
> If any parts of the AE Start date are unknown (UNK), compare the available parts of the date.

Rule expression

```
//to meet the rule description criteria 'aestdt >= infconsdt' should be met
if(getDatesCompareResult(aestdt,true,infconsdt,false,">="))
 {
  return true;
 }
 else
 {
  return false;          //System sends query when return false condition is
met
 }
```

Query message: Do not record events starting before the Date of Informed Consent. If dates are correct, move to Medical history. Otherwise, correct dates.

Definitions

aestdt
Corresponds to the *AE Start Date* from the rule description.

infconsdt
Corresponds to the *Date of Informed Consent* from the rule description.

getDatesCompareResult( )
Compares two dates (`aestdt`, `infconsdt`) using the passed in operator (**>=**), in this case: `aestdt` >= `infconsdt`.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Usage tips

- Always use the relevant date helper function to compare dates rather than comparing the variables directly using comparison operators.
- Use this when you want to perform a comparison for date questions where at least one of the dates is a partial date.

Verification steps

In the following verification steps for the given rule expression, we use <*date1*> that refers to the AE Start Date and <*date2*> that is Date of Informed Consent.

Test step	Test description	Result
1	Using a subject go to the visit <*Visit refname*>; form <*Form refname*> and enter the item <*date2 item refname*> as '**02-Dec-2021**'.	No query
2	Go to the visit <*Visit refname*>; form <*Form refname*> and enter the item <*date1 item refname*> as '**02-Dec-2021**'.	No query
3	Update item <*date1 item refname*> as '**01-Dec-2021**'.	Query
4	Update item <*date1 item refname*> as '**Unk-Dec-2021**'.	No query
5	Update item <*date1 item refname*> as '**Unk-Nov-2021**'.	Query
6	Update item <*date1 item refname*> as '**03-Dec-2021**'.	No query
7	Update item <*date2 item refname*> as '**05-Dec-2021**'.	Query
8	Update item <*date2 item refname*> as '**02-Dec-2021**'.	No query
9	Update item <*date2 item refname*> as '**01-Jan-2022**'.	Query
10	Update item <*date2 item refname*> as '**04-Dec-2021**'.	Query
11	Clear the item <*date2 item refname*>.	No query
12	Update item <*date2 item refname*> as '**02-Dec-2021**'.	No query
13	Update item <*date1 item refname*> as '**01-Dec-2021**'.	Query

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-17 AE Start Date must not be greater than Date of Death

```
if(getDatesCompareResult(aestdt,true,deathdt,false,'<=')
{
  return true;
}
else
{
  return false;
}
```

Query message: Start date of AE is greater than date of death. Please reconcile.

Example 3-18 AE Stop Date must be greater than AE Start Date

```
if(getDatesCompareResult(aestpdt,true,aestdt,true,'>=')
{
  return true;
}
else
{
  return false;
}
```

Query message: Stop Date is prior to Start Date. Please correct.

Example 3-19 AE Stop Date must not be greater than Date of Death

```
if(getDatesCompareResult(aestpdt,true,deathdt,false,'<=')
{
  return true;
}
else
{
  return false;
}
```

Query message: Stop Date of AE is greater than date of death. Please reconcile.

Partial date unknown month evaluation

Check if the month of the date question is selected as unknown (UNK) and display a query if needed.

Rule description: if UNK is selected as Month for Date of Initial Diagnosis then a query is issued.

Rule expression

```
if(DIADT.getMonth()==='UNK')     //checks for the presence of 'UNK'
value as the month of a date
{
   return false;                   //System sends query when return
false condition is met
}
else
{
   return true;
}
```

Query message: UNK has been selected for Month. Please verify and provide.

Definitions

DIADT
Corresponds to the *Date of Initial Diagnosis* from rule description.

===
Equal to comparison operator. Compares both the value and type to be equal.

.getMonth()
JavaScript method for date type elements. Retrieves the numeric value of the month in a date, for example '11' as the numeric value of Novemeber in '01-Nov-2021 15:03'. Returns 'UNK' if month value is not present.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Verification steps

In the following verification steps for the given rule expression, we use *<date1>* that refers to Date of Initial Diagnosis.

Test step	Test description	Result
1	Using a subject go to the visit *<Visit refname>* form *<Form refname>* and enter the item *<date1 item refname>* as '**10-May-2021**'	No query
2	Update the item *<date1 item refname>* as '**UNK-UNK-2021**'	Query
3	Update the item *<date1 item refname>* as '**UNK-May-2021**'	No query
4	Update the item *<date1 item refname>* as '**05-Jun-2021**'	No query
5	Update the item *<date1 item refname>* as '**UNK-UNK-2021**'	Query

Test step	Test description	Result
6	Clear the item <*date1 item refname*>	No query

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Dates with Dynamic Query Text

- Date comparison - dynamic query
  Compare two date questions that do not have time elements and display a dynamic query if the dates are not as expected.
- Date Time comparison - dynamic query
  Compare two date questions that also have time elements and display a dynamic query if the dates are not as expected.
- Partial date comparison with dynamic query text
  Compare two date questions where at least one of the dates is partial then issue a query that contains dynamic text if the dates are not as expected.

Date comparison - dynamic query

Compare two date questions that do not have time elements and display a dynamic query if the dates are not as expected.

Rule description: the date of the Informed Consent is Signed must be on or before the date entered in the Visit Date field for the Screening or Baseline visit.

Rule expression

```
//to meet the rule description criteria onstdt-compdt should be a negative
value or zero (<=0)
if(dateDiffInDays(icdat,vstdt)<=0)
 {
  return true;
 }
 else
 {
 setQueryMessage("Date Informed Consent signed
"+getDateDMYFormat(icdat,false)+" must be on or before the Visit date
"+getDateDMYFormat(vstdt,false) +" .Please correct or clarify.");
  return false;                    //Query message set dynamically. System
sends query when return false condition is met
 }
```

Query message: Date Informed Consent was signed {infconstdt} must be on or before the Visit date {visitdate}. Please correct or clarify.

Definitions

icdat
Corrsponds to the *Date of Informed Consent* from the rule description.

vstdt
Corresponds to the *Visit date* from the rule description.

<=
Less Than or Equalt To operator. Update operator based on the rule description.

dateDiffInDays( )
Calculates difference between date1 and date2 (*date1-date2*) in days, in this case `icdat - vstdt`.

setQueryMessage( )
Specify dynamic query text passed in as a parameter.

getDateDMYFormat( )
Use the getDateDMYFormat helper function to return a date (including partial dates) in DD-MON-YYYY format.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Usage tips

Always use the relevant date helper function to compare dates rather than comparing the variables directly using comparison operators.

Verification steps

In the following verification steps for the given rule expression, we use <*date1*> that refers to Date of Informed Consent and <*date2*> that is the Visit Date.

Test step	Test description	Result
1	Using a subject go to the visit <*Visit refname*> form <*form refname*> and enter the item <*date1 item refname*> as '**10-May-2021**'.	No query
2	Go to the visit <*Visit refname*> form <*Form refname*> and enter the item <*date2 item refname*> as '**10-May-2021**'.	No query
3	Update the item <*date1 item refname*> as '**11-May-2021**'.	Query. Verify correct date values are populated in the Query Text
4	Update the item <*date1 item refname*> as '**09-May-2021**'.	No query

Test step	Test description	Result
5	Update the item <*date1 item refname*> as '**09-Jun-2021**'.	Query. Verify correct date values are populated in the Query Text
6	Update the item <*date1 item refname*> as '**11-Apr-2021**'.	No query
7	Clear the item <*date1 item refname*>.	No query
8	Update the item <*date1 item refname*> as '**12-May-2021**'.	Query. Verify correct date values are populated in the Query Text
9	Update the item <*date2 item refname*> as '**14-May-2021**'.	No query

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-20 Date of Study Discontinuation must be >= Date Informed Consent signed

```
if(dateDiffInDays(studycompdt,infdt)>=0)
{
return true;
}
else
{
setQueryMessage("Date of Study Discontinuation
"+getDateDMYFormat(studycompdt,false)+" is less than Informed Consent date
"+getDateDMYFormat(infdt,false)+". Please correct or clarify.");
return false;
}
```

Query message: Date of Study Discontinuation {discontdate} is less than Informed Consent date {infdt}. Please correct or clarify.

Example 3-21 Date of Infusion must be equal to visit date of respective visits

```
if(dateDiffInDays(infudt, visdt)==0)
{
    return true;
}
else
{
setQueryMessage("Date of Infusion "+getDateDMYFormat(infudt,false)+" is
prior to or greater than visit date "+getDateDMYFormat(visdt,false)+".
Please correct or clarify.");
```

```
 return false;
}
```

Query message: Date of Infusion {infusiondt} is prior to or greater than visit date {visitdate}. Please correct or clarify.

Example 3-22 Start date of hypoglyceamic episode must be >= Date of randomization

> **Note:**
>
> Hypoglycaemic episode is date question with time elements.

```
if(dateDiffInDays(hypodt, randdt)>=0)
{
    return true;
}
else
{
setQueryMessage("Start date "+getDateDMYFormat(hypodt,false)+" is
prior to date of randomisation "+getDateDMYFormat(randdt,false)+".
Please correct.");
 return false;
}
```

Query message: Start date is prior to date of randomization ({RandDate}). Please correct.

Date Time comparison - dynamic query

Compare two date questions that also have time elements and display a dynamic query if the dates are not as expected.

Rule description: Collection Date and Time must be on or prior to the Study Vaccine Administration Date and Time of Injection.

Rule expression

```
//to meet the rule description criteria colldt-vaccdt should be a
negative value or 0 (<=0)
if(timeDiffInMinutes(colldt,vaccdt)<=0)
{
return true;
}
else
{
var dt1=DynamicDateInQT(colldt,false);
var dt2=DynamicDateInQT(vaccdt,false);
var qtstr="Potential Protocol Deviation: Blood sample "+dt1+" was
obtained post-injection "+dt2+".Please reconcile or complete Protocol
Deviation CRF."
setQueryMessage(qtstr);      //query message set dynamically
```

```
return false;                //System sends query when return false
condition is met
}

//defined function to returen datetime in DD-Mon-YYY HH:MM format
function DynamicDateInQT(dtval,ispartdate)
{
var qt1='';                      //placeholder for date value
var
fullmnth=["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","
Dec"];      //Month array to provide month in string format
qt1+=(pad2(dtval.getDate())+"-"+fullmnth[dtval.getMonth()]
+"-"+dtval.getFullYear()+" "+pad2(dtval.getHours())
+":"+pad2(dtval.getMinutes()));
return qt1;                      //returns date value

//defined function to return a two-digits numerical value, adds leading '0'
when required
}
function pad2(number) {
  return (number < 10 ? '0' : '') + number;
}
```

Query message: Potential Protocol Deviation: Blood sample {SampleDate} was obtained post-injection {injectiondate}. Please reconcile or complete Protocol Deviation CRF.

Definitions

colldt
Corresponds to the *Collection Date and Time* from the rule description.

vaccdt
Corresponds to the *Study Vaccine Administration Date and Time* from rule description.

<=
Less Than or Equal To operator. Update operator based on the rule description.

timeDiffInMinutes( )
Calculates difference between date1 (`colldt`), date2 (`vaccdt`) (*date1-date2*) in minutes.

setQueryMessage( )
Specify dynamic query text using the `setQueryMessage( )` function.

DynamicDateInQT(*dtval*,*ispartdate*)
Function defined in code. Returns a DateTime in DD-MON-YYYY HH:MM format. Takes in two parameters: a date (*dtval*) and a boolean (*ispartdate*) that states whether the date is partial or not.

> **Note:**
>
> `+" "+pad2(dt2.getHours())+":"+pad2(dt2.getMinutes())` is used to add time component. Exclude from the return statement if the date item does not include a time element.

.getMonth()
JavaScript method for date type elements. Retrieves the numeric value of the month in a date, for example '11' as the numeric value of Novemeber in '01-Nov-2021 15:03'.

.getDate()
JavaScript method for date type elements. Retrieves the numeric value of the day in a date, for example '1' in '01-Nov-2021 15:03'.

.getFullYear()
JavaScript method for date type elements. Retrieves the numeric value of the year in a date, for example '2021' in '01-Nov-2021 15:03'.

.getHours()
JavaScript method for date-time type elements. Retrieves the numeric value of the hours in the time component a date, for example '15' in '01-Nov-2021 15:03'.

.getMinutes()
JavaScript method for date-time type elements. Retrieves the numeric value of the minutes in a date, for example '03' in '01-Nov-2021 15:03'.

pad2(*number*)
Function defined in code. Takes a number type element as parameter and returns a two-digit numerical value, adding a leading zero when the passed-in *number* is less than 10.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Usage tips

- Always use the relevant date helper function to compare dates rather than comparing the variables directly using comparison operators.
- Use this when comparison should be performed for full dates with time.

Verification steps

In the following verification steps for the given rule expression, we use <*date1*> that refers to the Sample Collection Date and <*date2*> that is the Study Vaccine administration date.

Test step	Test description	Result
1	Using a subject go to the visit <*Visit refname*> form <*form refname*> and enter the item <*date1 item refname*> as '**10-May-2021 10:00**'	No query

Test step	Test description	Result
2	Go to the visit <*Visit refname*>form <*Form refname*> and enter the item <*date2 item refname*> as '**10-May-2021 10:00**'	No query
3	Update the item <*date1 item refname*> as '**10-May-2021 10:01**'	Q. Verify correct date values are populated in the Query Text
4	Update the item <*date1 item refname*> as '**10-May-2021 09:59**'	No query
5	Update the item <*date1 item refname*> as '**11-Jun-2021 10:00**'	Q. Verify correct date values are populated in the Query Text
6	Update the item <date1 item refname> as '**11-Apr-2021 07:01**'	No query
7	Clear the item <date1 item refname>	No query

Other examples

Example 3-23 ECG Date [and Time Performed] must be on or prior to either 'Date of Study Completion'

```
if(dateDiffInDays(ecgdt,compdt)<=0)
{
return true;
}
var dt1=DynamicDateInQT(ecgdt,false);
var dt2=getDateDMYFormat(compdt,false);
var qtstr="Date is "+dt1+" after Date of Study Completion or Discontinuation
"+dt2+". Please correct or confirm date(s).";
setQueryMessage(qtstr);
return false;
}

function DynamicDateInQT(dtval,ispartdate)
{
var qt1='';
var
fullmnth=["Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","
Dec"];
qt1+=(pad2(dtval.getDate())+"-"+fullmnth[dtval.getMonth()]
+"-"+dtval.getFullYear()+" "+pad2(dtval.getHours())
+":"+pad2(dtval.getMinutes()));
return qt1;
}
function pad2(number) {
  return (number < 10 ? '0' : '') + number;
}
```

Query message: Date is after Date of Study Completion or Discontinuation. Please correct or confirm date(s).

Partial date comparison with dynamic query text

Compare two date questions where at least one of the dates is partial then issue a query that contains dynamic text if the dates are not as expected.

Rule description: AE Stop Date must be on or after the Date Informed Consent.

> **Note:**
>
> If any parts of AE Stop date are unknown (UNK), compare the available date parts.

Rule expression

```
if(getDatesCompareResult(aeenddt,true,infconsdt,false,'>=')) {
  return true;
 }
 else
 {
 setQueryMessage("AE Stop date "+getDateDMYFormat(aeenddt,true)+" is
prior to Informed Consent date "+getDateDMYFormat(infconsdt,false)+".
Please correct or confirm.");
  return false;              //Query message set dynamically. System
sends query when return false condition is met.
 }
```

Query message: AE Stop date is prior to Informed Consent date. Please correct or confirm.

Definitions

aeenddt
Corresponds to *AE Stop Date* from the rule description (Partial Date), followed by `True`, as *AE Stop Date* is partial date.

infconsdt
Corrsponds to the *Informed Consent Date* from rule description (full date), followed by `False`, as *Informed Consent Date* is full date.

>=
Greater Than or Equal To operator. Update operator based on the rule description.

getDatesCompareResult( )
Compares two dates (`aeenddt`, `infconsdt`) using the passed in operator (**>=**). In this case: `aeenddt` **>=** `infconsdt`.

getDateDMYFormat( )

Use the getDateDMYFormat helper function to return a date (including partial dates) in DD-MON-YYYY format.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Usage tips

- Use this when comparison should be performed for date questions and at least one of the dates is partial.
- Query text should contain the dynamically entered date question values in it.

Verification steps

In the following verification steps for the given rule expression, we use <*date1*> that refers to the AE Stop Date and <*date2*> that is the Date of Informed Consent.

Test step	Test description	Result
1	Using a subject go to the visit <*Visit refname*> form <*form refname*> and enter the item <*date2 item refname*> as '**02-Dec-2021**'	No query
2	Go to the visit <*Visit refname*> form <*Form refname*> and enter the item <*date1 item refname*> as '**02-Dec-2021**'	No query
3	Update the item <*date1 item refname*> as '**01-Dec-2021**'.	Query
4	Update the item <*date1 item refname*> as '**UNK-Dec-2021**'.	No query
5	Update the item <*date1 item refname*> as '**UNK-Nov-2021**'.	Query
6	Update the item <*date1 item refname*> as '**03-Dec-2021**'.	No query
7	Update the item <*date2 item refname*> as '**05-Dec-2021**'.	Query
8	Update the item <*date2 item refname*> as '**02-Dec-2021**'.	No query
9	Update the item <*date2 item refname*> as '**01-Jan-2022**'.	Query
10	Update the item <*date2 item refname*> as '**04-Dec-2021**'.	Query
11	Clear the item <*date2 item refname*>.	No query
12	Update the item <*date2 item refname*> as '**02-Dec-2021**'.	No query

Test step	Test description	Result
13	Update the item <*date1 item refname*> as '**1-Dec-2021**'.	Query

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-24 Date of Study Completion must be on or after the Last Date of Study Drug

Note: If any parts of the parts of the Last Date of Study Drug are UNK, compare the available date parts.

```
if(getDatesCompareResult(compdt,false,drugdt,true,'>='))
{
  return true;
}
else
{
setQueryMessage("Date of Study Completion
"+getDateDMYFormat(compdt,false)+" is prior to Last Date of Study Drug
"+getDateDMYFormat(drugdt,true)+" .Please correct or confirm.");
  return false;
}
```

Query message: Date of Study Completion is prior to Last Date of Study Drug. Please correct or confirm.

Example 3-25 CM Stop Date must be on or after CM Start Date

Note: If any parts of CM Start/Stop date are unknown, compare available date parts.

```
if(getDatesCompareResult(cmenddt,true,cmstdt,true,'>='))
{
  return true;
}
else
{
setQueryMessage("Date of Study Completion
"+getDateDMYFormat(cmenddt,true)+" is prior to Last Date of Study Drug
"+getDateDMYFormat(cmstdt,true)+" .Please correct or confirm.");
  return false;
}
```

Query message: CM Stop Date is prior to CM Start Date. Please correct and clarify.

Repeating form examples

- Instance count
  Count the number of instances in a repeating form.
- Duplicate values check
  Check for duplicate data in a repeating form.
- Compare related instances
  Check a value for a matching instance in a repeating form.

Instance count

Count the number of instances in a repeating form.

Rule description: if the Indication on a Concomitant Medications (ConMeds) form is assigned to an Adverse Event, there should be at least one non-deleted adverse event record present on the Adverse Event (AE) form.

Rule expression

```
 if(AESER!==null || AESERY!==null || AESERYOTH!==null || AESTDT!==null ||
AEONGO!==null || AEENDT!==null || AEOUT!==null){}          //This is to make
sure the code runs when any of the other items of the AE form is updated. It
does not include the item already used in the code
 var indval=getStringFromChoice(INDICAT);
 var aecnt=ListRFInstances(AETERM,0);
 if(indval.contains("Adverse Event"))          //multi-select question.
Evaluates for a specific given choice.
 {
 if(aecnt.length<=0)
 {
 return false;                    //System sends query when return false
condition is met
 }
 else
 {
   return true;
 }
}
else
{
    return true;
}
```

Query message: Medication is indicated as being taken for an adverse event but no records are recorded on the Adverse Events (AE) eCRF. Please verify.

Definitions

INDICAT
Corresponds to the *Indication on a Concomitant Medications (ConMeds) form* from the rule description.

AETERM
Corresponds to the *Adverse Events records on the Adverse Event (AE) form* from the rule description.

AESER, ***AESERY***, ***AESERYOTH***, ***AESTDT***, ***AEONGO***, ***AEENDT***, ***AEOUT***
Items in the repeating form.

aecnt
Defined JavaScript variable that stores a list of instances. When using **`aecnt.length`** we retrive the number of instances contained in this list.

getStringFromChoice( )
Converts selected label for the choice element (drop-down, radio buttons or checkboxes) to a string or a comma-separated value. Takes in the choice element as parameter.

ListRFInstances( )
Lists all repeating form instances of the passed-in variable. Takes an item variable in the form as a parameter.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Usage tips

To make sure the rule runs whenever any of the items in the repeating form is completed or updated, you must create global variables for each of them and use the variables to evaluate if any of these are not null. This is done in the first line of the rule expression.

> **Note:**
>
> For this evaluation, you should not include the item passed as a parameter to the ListRFInstances( ) helper function in the rule expression logic.

Verification steps

In the following verification steps for the given rule expression, we use *<item refname>* refers to indication on Concomitant Medications form as Adverse Event.

Test step	Test description	Result
1	Using a subject go to the visit <Visit refname> form <Form refname> and enter the item <item refname> as '**Yes**'	Query

Test step	Test description	Result
2	Go to the visit <Visit refname>; form <Form refname>; First Repeating Form and complete all items (all the items of the repeating form for which it is required to check atleast one non deleted instance is present.)	No query
3	Clear all the item in First Repeating Form completed in previous step	No query
4	Delete the First Repeating Form	Query
5	Update the item <item refname> as 'No'	No query
6	Update the item <item refname> as 'Yes'	Query
7	Create a Second Repeating Form and enter only one item or less than all the items of the repeating form for which it is required to check atleast one non deleted instance is present.	Query
8	Create 3rd repeating form and Complete the remaining required items from previous step in 3rd repeating form	Query
9	Complete the remaining required items in 2nd repeating form	No query
10	Delete the Third Repeating Form	No query

Note:

Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-26 If 'Was a follow-up lesion assessment performed?' if 'No,' then there isn't a non-deleted add entry record present on TARGET

```
if(R2!==null || R3!==null || R4!==null || R5!==null || R6!==null || R7!
==null || R8!==null){}
var chk1val=getStringFromChoice(FOLLOWASS);
var tarcnt=ListRFInstances(R1, 0);
   if(chk1val.contains("No"))
 {
```

```
 if(tarcnt.length<=0)
 {
 return true;
 }
 else
 {
   return false;
 }
}
else
{
    return true;
}
```

Query message: You have selected No. However, information has been recorded. Please review and correct.

Duplicate values check

Check for duplicate data in a repeating form.

Rule description: we do not want any duplicates for Follow-up 'RECIST Evaluation Number' to be recorded on the TARGET form.

Rule expression

```
if(FindDuplicateRepeatingForm(TLFEVAL))
{
 return false;                    //System sends query when return
false condition is met
}
else
{
 return true;
}
```

Query message: A duplicate RECIST Evaluation Number has been recorded. Please verify and correct.

TLFEVAL
Corresponds to the *RECIST Evaluation Number* from rule description.

FindDuplicateRepeatingForm( )
Helper function to detect duplicate data in a repeating form for search keys passed in as parameters (**TLFEVAL**)

> **Note:**
>
> This is an aggregation function. The rule will be run for each form instance in the case where the target is on a repeating form.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Usage tips

Use this when the item is not a choice control.

Verification steps

Test step	Test description	Result
1	Using a subject go to the visit <Visit refname> form <Form refname> and first Repeating Form and enter the item <Item Refname> as 'Value 1' (input value is as per item type i.e. Text/ Date/Number).	No query
2	Go to the visit <Visit refname>; form <Form refname> and Second Repeating Form and enter the item <Item Refname> as 'Value 1.'	Query [1st and 2nd RF]
3	Update item <Item Refname> in 2nd RF as 'Value 2.'	No query [1st RF & 2nd RF]
4	Update item <Item Refname> in 1st RF as 'Value 2.'	Query [1st RF & 2nd RF]
5	Clear item <Item Refname> in 2nd RF.	No query [1st RF & 2nd RF]
6	Enter item <Item Refname> in 2nd RF as 'Value 3'	No query [1st RF & 2nd RF]
7	Go to 3rd Repeating test Form <Form refname> and enter the item <Item Refname> as 'Value 2.'	Query [1st & 3rd RF]; No query[2nd RF]
8	Update the item <Item Refname> in 3rd RF as 'Value 1.'	No query [1st & 2nd & 3rd RF]
9	Update the item <Item Refname> in 3rd RF as 'Value 3.'	Query [2nd & 3rd RF]; No query[1st F]
10	Delete the 2nd Repeating Form	No query

RF = Repeating Form.

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-27 If more than one AE Term exists with the same start date, issue a query

```
if(FindDuplicateRepeatingForm(aetrm,onstdt)) {
  return false;
} else {
  return true;
}
```

Query message: An adverse event term with the same start date is reported more than once. Please correct.

Compare related instances

Check a value for a matching instance in a repeating form.

Rule description: severity (a radio control) must be different from the previous one for the same related Adverse Event (AENUM).

Rule expression

```
//variable declaration
var rc;                          //Radio Control - severity
var ins;                         //placeholder for Repeating Form
instance
var outc='';                     //placeholder for severity label value
var cnt=0;                       //counter variable

//function definition to identify number of instances matching the
severity value
function functi(item,index)
{
    if(item.deleted===false && item.value!==null && index!==(ins-1) &&
index<ins-1)
    {


if(item.value===aenum1val)                                    //Checks
if the passed-in instance has a matching AE number value with the
current instance
        {
             if(newsev[index]!==null && sevval!
==null)
             {
                 outc=JSON.parse(newsev[index].value)[0].label;     //
Retrieves label from severity selection made by the user in the
related instance
                 if(outc===sevval)                                  //
Checks if the severity value in the related instance is matching in
the current instance
                 {
```

```
                    cnt=cnt+1;                                    //Updates
the number of identified matching instances
                }
                else
                {
                   cnt=cnt;
                }
               }
            else
            {
                 cnt=cnt;
            }
        }
        else
        {
             cnt=cnt;
        }
     }
     else
     {
          cnt=cnt;
     }
}

try
{
//variable declaration
    ins = GetCurrentRFInstance();                                 //Retrieves
instance of the current AE form
    var aenum1val=aenum1;                                         //Adverse
event number item
    var sevval=getStringFromChoice(newsev1);                     //Variable
for severity item to retrieve the label for the severity choice item
selected by the user
    rc=getValues('aenum','newsev');                               //Gets
values entered for AE number and severity items
    if(rc===true && aenum1val!==null && sevval!==null && ins>1)     //Checks
if getValues function retrieved results array, if AE number and severity
values are not null and if current instance is not the first instance of the
form
    {
        aenum.forEach(functi);        //Execute the function code for all AE
number values entered in the form
        if(cnt>0 )
        {
            return false;          //System sends query when return false
condition is met
        }
        else
        {
            return true;
        }
    }
    else
    {
```

```
        return true;
    }
}
catch(err)
{
    setQueryMessage(err);                //set query message to
display an encountered error
    return false;                        //System sends query when
return false condition is met
}
```

Query: The new severity is the same as the previous one. Please check.

Definitions

newsev
Created variable for the *Severity* item from rule description.

aenum
Corresponds to the *Adverse Event Number* item from the rule description.

GetCurrentRFInstance( )
Gets the form instance number of the current repeating form.

getStringFromChoice( )
Converts selected label for the choice element (drop-down, radio buttons or checkboxes) to a string or a comma-separated value. Takes in the choice element as parameter.

getValues( )
Fetches values for one or more variables across multiple visits, in an array format ordered by visits. In this case takes *aenum* and *newsev* variables described above.

setQueryMessage( )
Specify dynamic query text passed in as a parameter.

functi(*item*,*index*)
Function declared in code. Identifies number of instances matching the giving severity value.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Verification steps

Test step	Test description	Result
1	Using a new patient go to test visit <Visit refname> test form <Form Refname> Enter Advese Event Number item as '**1**' and select item Severity <Item refname> as '**Grade 1**'.	No query [1st RF]

Test step	Test description	Result
2	Create second repeating test form <Form Refname> Enter Advese Event Number item as '**1**' and Select item Severity <Item refname> as '**Grade 1**'.	Query [2nd RF]
3	Select second repeating form <Form Refname> and update item Severity <Item refname> as '**Grade 2**'.	No query [1st & 2nd RF]
4	Select second repeating test form <Form Refname> update Severity item <Item refname> as '**Grade 1**'.	Query [2nd RF]
5	Select second repeating test form <Form Refname> Update Adverse Event Number item <Item refname> as '**2**'.	No query [1st & 2nd RF]
6	Create third repeating test form <Form Refname> Enter Adverse Event Number item <Item refname> as '**1**' and select item Severity <Item refname> as '**Grade 1**'.	Query [3rd RF]
7	Select third repeating test form <Form Refname> Update Adverse Event Number item <Item refname> as '**3**'.	No query [1st & 2nd & 3rd RF]
8	Select third repeating test form <Form Refname> Update Adverse Event Number item <Item refname> as '**2**'.	Query [3rd RF]
9	Select second repeating test form <Form Refname> and update Severity item <Item refname> as '**Grade 3**'.	No query [1st & 2nd & 3rd RF]

RF = Repeating Form.

Other examples

Example 3-28 The start date of the treatment must be after or the same as the stop date of treatment for the previous prescription of the same drug

```
var rc;
var ins;
var ind=-1;
var res='';
function functi(item,index)
{
    if(item.deleted===false && item.value!==null && index!==(ins-1) &&
index<ins-1)
    {
```

```
        if(item.value===trtname1)
        {
            if(stpdt[index]!==null && stdt[ins-1]!==null)
            {

if(getDatesCompareResult(stdt[ins-1].value,true,stpdt[index].value,true
,'>='))
                {
                    ind=1;

                }
                else{  ind = 0;}
            }
        }
    }
}
    ins = GetCurrentRFInstance();
    rc=getValues("trtname","stdt","stpdt");
    if(rc===true && ins!==1)
    {
        trtname.forEach(functi);
        if(ind===0)
        {
            return false;
        }
        else { return true; }
    }
    else
    {
        return true;
    }
```

Query message: Start date is prior to stop date of previous prescription. Please correct or confirm.

Two-section form examples

- Table instance count
  Find table row instances where the rule is currently being executed for a two-section form.
- Form instance count
  Find the number of form instances where the rule is run on two-Section forms.
- Duplicate values check - flat section items
  Check if more than one form instance contains the same value for a given item in the flat section of a two-section form.
- Duplicate values check - table section items
  Check if more than one table instance contains the same value for a given item in a respective two-section form.

Table instance count

Find table row instances where the rule is currently being executed for a two-section form.

Rule description: if *Yes* is selected for *Does the subject have any relevant Medical History?*, then there must be at least one non-deleted table instance recorded or a query is issued.

Rule expression

```
If(MHSTDT!==null || MHONG!==null || MHENDT!==null){…..}
var instval=getCurrent2SFormInstance();
if(getStringFromChoice(MHYes)==='Yes' )
{
var instcnt=list2SInstances(MHTERM,instval,0);
if(instcnt.length > 0)
{
return true;
}
else
{
return false;               //System sends query when return false condition
is met
}
}
else
{
return true;
}
```

Query message: "Does the subject have any relevant Medical History?" has been answered "Yes", therefore data is expected in the table. Please review and complete.

Definitions

MHSTDT*, *MHONG*, *MHENDT
Table section items in two-Section form.

MHYes
Item on Flat section which is target item.

MHTERM
Table section item in two-Section form used as parameter to the list2SInstances() helper function.

getCurrent2SFormInstance( )
Gets the form instance number of the current two-section form.

getStringFromChoice( )
Converts selected label for the choice element (drop-down, radio buttons or checkboxes) to a string or a comma-separated value. Takes in the choice element as parameter.

list2SInstances( )

Lists all table instances of the passed-in variable in a two-section form. Takes an item variable of the table section in the form as a parameter.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Usage tips

To make sure the rule runs whenever any of the items in the table section of a two-section form is completed or updated, you must create global variables for each of them and use the variables to evaluate if any of these are not null. This is done in the first line of the rule expression.

> **Note:**
>
> For this evaluation, you should not include the item passed as a parameter to the ListRFInstances( ) helper function in the rule expression logic.

Verification steps

Test step	Test description	Result
1	Using a subject go to the visit <Visit refname> form <Form refname> create 1st form instance and enter the flat section item <item refname> as '**Yes**'.	Query [1st form instance]
2	In 1st form instance create 1st instance and complete all the items.	No query [1st form instance]
3	In 1st form instance clear all the items in first table instance.	No query [1st form instance]
4	In 1st form instance delete first table instance.	Query [1st form instance]
5	In 1st form instance Update flat section item <item refname> as '**No**'.	No query [1st form instance]
6	In 1st form instance Update flat section item <item refname> as '**Yes**'.	Query [1st form instance]
7	In 1st form instance create second table instance and complete any of the items.	No query [1st form instance]
8	Create 2nd form instance and complete item <item refname> as '**Yes**'.	Query [2nd form instance]

Test step	Test description	Result
9	In 2nd form instance create 1st table instance and complete any of the items.	No query [1st & 2nd form instances]
10	Delete the 2nd form instance Form.	No query [1st & 2nd form instances]

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-29 Trigger query if the read only "Was PE Date populated?' is populated with "Yes", and there is no date completed in the repeating section.

```
var instval=getCurrent2SFormInstance();
if(PEDT!==null)
{
var instcnt=list2SInstances(RES,instval,0);
if(instcnt.length > 0)
{
return true;
}
else
{
return false;
}
}
else
{
return true;
}
```

Query message: Date of Physical Exam is entered. However, there is no entry in the table.

Form instance count

Find the number of form instances where the rule is run on two-Section forms.

Rule description: there should not be more than five form instances recorded on the Target Lesion form, or a query is issued.

Rule expression

```
 If(organ!==null || vst!==null || assess!==null)
var cnt= list2SInstances(lesid,null,0);
if(cnt.length>5)
{
return false;              //System sends query when return false condition
```

```
is met
}
else
{
return true;
}
```

Query message: There are five or less Target Lesion measurements expected, please verify and correct.

Definitions

organ
An item in the form (Including items in flat section and table section).

vst
An item in the form (Including items in flat section and table section).

assess
An item in the form (Including items in flat section and table section).

lesid
Target item which is Flat section item.

list2SInstances( )
Lists all table instances of the passed-in variable in a two-section form. Takes an item variable of the table section in the form as a parameter.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Usage tips

To make sure the rule runs whenever any of the items in the table section of a two-section form is completed or updated, you must create global variables for each of them and use the variables to evaluate if any of these are not null. This is done in the first line of the rule expression.

> **Note:**
>
> For this evaluation, you should not include the item passed as a parameter to the ListRFInstances( ) helper function in the rule expression logic.

Verification steps

Test step	Test description	Result
1	Using a subject go to the visit <Visit refname> form <Form refname> and create 1st form instance by entering value in target item <item refname>.	No query

Test step	Test description	Result
2	Create 2nd form instance - Create 1st table instance and complete any of the items.	No query
3	Create 3rd form instance and enter any items in the flat section of the form.	No query
4	Create 4th form instance and enter any items in the flat section of the form.	No query
5	Create 5th form instance and enter any items in the flat section of the form.	No query
6	Create 6th form instance and enter any items in the flat section of the form.	Query [all 6 instances]
7	Delete 2nd form instance.	No query

Note:

Repeat the above steps if the form is present in multiple visits.

Duplicate values check - flat section items

Check if more than one form instance contains the same value for a given item in the flat section of a two-section form.

Rule description: all Form Instances contain a unique Lesion ID. Issue a query if a Lesion ID is duplicated.

Rule expression

```
if(findDuplicate2SForm(null,lesid))
{
    return false;               //System sends query when return false
condition is met
}
else
{
    return true;
}
```

Query Message: The number recorded for Lesion ID has already has been used. Please confirm and correct.

Definitions

lesid
Corresponds to *Lesion ID* that is present in the flat section of a two-section form from rule description.

findDuplicate2SForm( )
Identifies duplicated data as item values for the variables provided as parameters, in this case `lesid`.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Usage tips

Use this when the item is not a choice control.

Verification steps

Test step	Test description	Result
1	Using a subject go to the visit <Visit refname> form <Form refname> and first form instance and enter the item <Item Refname> as '**1**'.	No query [1st form instance]
2	Create Second form instance and enter the item <Item Refname>as '**1**'.	Query [1st & 2nd form instances]
3	Update item <Item Refname> in Second form instance as '**2**'.	No query [1st & 2nd form instances]
4	Update item <Item Refname> in first form instance as '**2**'.	Query [1st & 2nd form instances]
5	Clear item <Item Refname> in 2nd form instance.	No query [1st & 2nd form instances]
6	Update item <Item Refname> in 2nd form instance as '**3**'.	No query [1st & 2nd form instances]
7	Create 3rd form instance and enter the item <Item Refname>as '**2**'.	Query [1st & 3rd form instances]
8	Update the item <Item Refname> in 3rd form instance as '**1**'	No query [1st & 2nd & 3rd form instances]
9	Update the item <Item Refname> in 3rd form instance as '**3**'.	Query [2nd &3rd form instances]
10	Delete 2nd form instance.	No query [1st & 2nd & 3rd form instances]

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-30 Method of Assessment should stay the same across all records for each form instance/Lesion ID

For example, if the Method of Assessment for Lesion Number '1' is MRI, then all records for Lesion ID '1' must have a Method of Assessment entered as 'MRI.'

```
if(findDuplicate2SForm(null,assmethod))
{
    return true;
}
else
{
    return false;
}
```

Query message: Method of Assessment is different than the value previously recorded. Please verify.

Duplicate values check - table section items

Check if more than one table instance contains the same value for a given item in a respective two-section form.

Rule description: issue a query if a duplicate Abnormality/Condition is entered in the Medical History table section.

Rule expression

```
var instval=getCurrent2SFormInstance();
if(findDuplicate2SForm(instval,MHCondition))
{
   return false;               //System sends query when return false
condition is met
}
else
{
    return true;
}
```

Query Message: Abnormality/Condition has been recorded in duplicate, please verify and correct.

Definitions

MHCondition
Corresponds to the *Abnormality/Condition* that is present in the table section of a two-section form, from rule description.

getCurrent2SFormInstance( )
Gets the form instance number of the current two-section form.

findDuplicate2SForm( )
Identifies duplicated data as item values for the variables provided as parameters, in this case `lesid`.

Return value

Boolean
Returns either `true` or `false`. System raises query when return false condition is met.

Usage tips

Use this when an item is not a choice control.

Verification steps

Test step	Test description	Result
1	Using a subject go to the visit <Visit refname> form <Form refname> and first form instance. In first table instance enter the item <Item Refname> as '**Value 1**'.	No query [1st table instance]
2	In 1st form instance create Second table instance and enter the item <Item Refname>as '**Value 1**'.	Query [1st & 2nd table instances]
3	Update item <Item Refname> in Second table instance as '**Value 2**' .	No query [1st & 2nd table instance]
4	Update item <Item Refname> in first table instance as '**Value 2**'.	Query [1st & 2nd table instances]
5	Clear item <Item Refname> in 2nd table instance.	No query [1st & 2nd table instance]
6	Update item <Item Refname> in 2nd table instance as '**Value 3**'.	No query [1st & 2nd table instances]
7	Create 3rd table instance and enter the item <Item Refname>as '**Value 2**'.	Query [1st & 3rd table instance]
8	Update the item <Item Refname> in 3rd table instance as '**Value 1**'.	No query [1st & 2nd & 3rd table instance]

Test step	Test description	Result
9	Update the item <Item Refname> in 3rd table instance as '**Value 3**'.	Query [2nd & 3rd table instances]
10	Create 2nd form instance and in 1st table instance enter value as '**Value 3**'.	No query [2nd form instance - 1st table instance]
11	In 2nd form instance In 2nd table instance enter value as '**Value 3**'.	Query [2nd form instance - 1st table instance]
12	In 2nd form instance delete 2nd table instance.	No query [2nd form instance - 1st table instance]

> **Note:**
>
> Repeat the above steps if the form is present in multiple visits.

Other examples

Example 3-31 If Timepoint (or Visit) is selected and previous record already uses timepoint, then fire query

```
var frminst=getCurrent2SFormInstance();
if(findDuplicate2SForm(frminst,Visit))
{
return false;
}
else
{
return true;
}
```

Query message: The time point selected has already been reported on a previous record. Please review and reconcile.

Example 3-32 Date of Assessment cannot be duplicated. For example, if 01/01/2021 is already recorded for a previous Timepoint, it cannot be recorded again

```
var frminst=getCurrent2SFormInstance();
if(findDuplicate2SForm(frminst,assdt))
{
return false;
}
else
{
return true;
}
```

Query message: Date of Assessment is already recorded. Please review and correct.

4 Revision history

Table 4-1 Revision History

Date	Part Number	Description
June 2022	F54972-01	Original version of the document.
June 2022	F54972-02	Included updates for the current release.
June 2022	F54972-03	Changed the publication release date.