Oracle® Life Sciences Clinical One Platform

Rules Developer Guide

Release 24.2
F99798-02
September 2024

ORACLE

Oracle Life Sciences Clinical One Platform Rules Developer Guide, Release 24.2

F99798-02

Primary Author: Oracle Life Sciences Documentation Team

Contents

3 Rules helper function reference

6 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 Customer Support Portal (https://hsgbu.custhelp.com/)
- Japanese interface Customer Support Portal (https://hsgbu-jp.custhelp.com/)

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

Additional copyright information

This documentation may include references to materials, offerings, or products that were previously offered by Phase Forward Inc. Certain materials, offerings, services, or products may no longer be offered or provided. Oracle and its affiliates cannot be held responsible for any such references should they appear in the text provided.

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 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`)
- Words internally restricted by the Rules engine:
  - Expression
  - Window

There are also some reserved words that are typically not allowed in JavaScript. For a complete list, see JavaScript Reserved Words.

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.

The Subject Object

The Subject Object provides access to subject information and other data not collected directly into a form, so you can include it in a rule's processing or as a return value.

Sometimes, a rule's logic may need to reference data that wasn't collected in a form, such as the subject number and status, or site-related information. The Subject Object has different attributes for data that is not included in any form. You can use the Subject Object and its attributes for rule processing or as a result for a calculated value.

> **Tip:**
>
> This last scenario can be useful to run Show Visit, Show Form, and Show Question predefined rules or as a form question included for visit branching.

All of these attributes will always return the current value. This means that, if there is any change to the subject, old data is overwritten and the new data is returned. Also, changes in the values returned by the Subject Object do not trigger a rule to run or re-run. Because rules run on data submission, a form item variable must be referenced to trigger the rule to run whenever it changes to meet the specified criteria.

Usage tips

- When accessing the object properties using dot notation, it is always a good practice to check if the property is defined to avoid code failure when null. For example:

```
if(Subject.SubjectNumber){
    var subnumber = Number(Subject.SubjectNumber.substring(10));
}
```

- A predictive text feature is available as you type, with descriptions of these attributes and all available rule helper functions. See Use predictive text to write rules.

Subject Object attributes

> **Caution:**
>
> Some of these attributes allow access to potentially unblinding information. Use them with caution and along with assigned queries and data classifications, when applicable, to limit access to this information.

Attribute	Description
`SubjectNumber`	Indicates the number assigned to a subject.
`SubjectStatus`	Indicates a subject's current status. The following statuses can be displayed: • New • Screening_Initiated • Screen_Failed (when a site user manually marks a subject as having failed screening) • Auto_Screen_Failed (when a subject automatically fails screening in the system) • Enrolled • Active • Withdrawn • Complete
`ScreeningDate`	Indicates the date when a subject is screened. By default, all dates are returned with a standard 00:00 GMT time.
`ScreenFailureDate`	Indicates the date when a subject fails screening, whether the screen failure occurs manually or automatically. By default, all dates are returned with a standard 00:00 GMT time. **Note:** If the date of the screen failure is entered manually and differs from the system date, the manually entered value is returned. Subjects that were automatically screen failed prior to the 24.1 release will not have a date returned for the `ScreenFailureDate`.
`CompletionDate`	Indicates the date when a subject completed all visits in a study. By default, all dates are returned in GMT time.
`WithdrawalDate`	Indicates the date when a subject is withdrawn from a study. By default, all dates are returned with a standard 00:00 GMT time.

Attribute	Description
RandomizationDate	Indicates the date for any of the following events: • When a subject is randomized whether the randomization number subsequently changed. • When a subject is re-randomized. By default, all dates are returned with a standard 00:00 GMT time. **Note:** In the event of a manual randomization outside of Oracle Clinical One Platform, this returns the date the subject was randomized in Oracle Clinical One Platform, not the date of manual randomization that occurred outside of Oracle Clinical One Platform.
TreatmentArm	Indicates the ID (also known as the short name) of the treatment arm for the subject. **Caution:** Potentially unblinding. **Note:** Treatment arms containing a backslash ("\") on their IDs, cannot be referenced as text in a rule. You can use logMsg() statements to verify the retrieved string, see Debug a rule.
CohortName	Indicates the name of the cohort where a subject belongs to, if applicable.
SiteNumber	Indicates a site's ID where a subject is assigned to, as specified on the Create Organization: Institution dialog, in the **Institution ID** field.
SiteName	Indicates a site's name where a subject is assigned to, as specified on the Create Organization: Institution dialog, in the **Institution Name** field.
Country	Indicates the ISO code of a country specified for a site where the subject is assigned to. This value is defined as indicated in the Geography system code list on the Library page.
Region	Indicates the primary region of a country for a site that a subject is assigned to. A country's region is defined on the Study Settings tab for a study.

Handle partial dates in custom rules

Oracle Clinical One Platform handles dates differently depending on whether they are partial dates or not.

While writing a rule definition in the Rule editor you can retrieve the value of a date item into a variable. If the date is a full date, a standard JavaScript `Date` object is created to hold the variable. In the case for partial dates or time elements only, a custom object `C1Date` (defined only for Oracle Clinical One Platform) is created.

When it comes to using date variables with date and time helper functions, you must know that there are different types of helper functions and not all of them take partial dates. Only when a rule helper function is pure JavaScript, both rule variables and locally defined variables (either as `Date` or `C1Date` objects) can be used. The available pure JavaScript rule helper functions are:

- getDateDMYFormat( )
- getDatesCompareResult( )
- partialDateDiff( )

Other functions only take full JavaScript `Date` objects, meaning partial dates are not supported. You may refer to the whole list of available helper functions and their documentation in the Rules helper function reference section.

Usage tips

- Check your variable type to avoid rule failure. To know how your variable is created and which type of object it is, you can use the logMsg(). See Debug a rule.
- You can use the associated methods of each object type to gather specific elements of any date. This way you can manually evaluate and compare dates that may not be supported by existing rule helper functions.

Review more information on the available date objects:

- The standard JavaScript Date object
  The `Date` object is a standard built-in JavaScript object and supports full dates only. This object has built-in methods that can be used to retrieve specific date components.
- The custom C1Date object
  The `C1Date` object is defined only for Oracle Clinical One Platform and supports both full and partial dates.

The standard JavaScript Date object

The `Date` object is a standard built-in JavaScript object and supports full dates only. This object has built-in methods that can be used to retrieve specific date components.

> **Note:**
>
> If you need to work with partial dates see Handle partial dates in custom rules and The custom C1Date object.

Method	Description
`getFullYear()`	Get **year** as a four digit number (yyyy).
`getMonth()`	Get **month** as a number (0-11).
`getDate()`	Get **day** as a number (1-31).
`getDay()`	Get **weekday** as a number (0-6).
`getHours()`	Get **hour** (0-23).
`getMinutes()`	Get **minute** (0-59).
`getSeconds()`	Get **second** (0-59).

For more information, you may refer to any official JavaScript documentation resources.

The custom C1Date object

The `C1Date` object is defined only for Oracle Clinical One Platform and supports both full and partial dates.

Since this is a custom class, you need to know specific details about its constructors and methods. This information allows you to use `C1Date` objects to properly handle partial dates.

Constructors

The `C1Date` object has two possible constructors:

```
C1Date(date, day, month, year)
```

```
C1Date (date, day, month, year, hour, minute, second)
```

These two possible constructors use the following object parameters:

Parameter	Description
`date`	Takes a JavaScript `Date` object (full date).
`day`	Takes a numeric value for the day of the date.
`month`	Takes a numeric value for the month of the date.
`year`	Takes a 4-digit numeric value for the year of the date (yyyy).
`hour`	Takes a numeric value for the hours time element.
`minute`	Takes a numeric value for the minutes time elemnt.
`second`	Takes a numeric value for the seconds time element.

> **Note:**
>
> When a `date` is provided, following parameters are not needed.

Methods

The `C1Date` class has the following methods:

Method	Description
`isPartialDate()`	Returns `true` for partial dates or `false` for full dates.
`getDate()`	Returns a JavaScript `Date` object, or `null`.
`getDay()`	Returns the numeric value of the day, `null` or UNK.
`getMonth()`	Returns the numeric value of the month, `null` or UNK.
`getYear()`	Returns the 4-digit value of the year, `null` or UNK.
`getHour()`	Returns the numeric value of the hours time element, `null` or UNK.
`getMinute()`	Returns the numeric value of the minutes time element, `null` or UNK.
`getSecond()`	Returns the numeric value of the seconds time element, `null` or UNK.

2

Create and manage custom rules

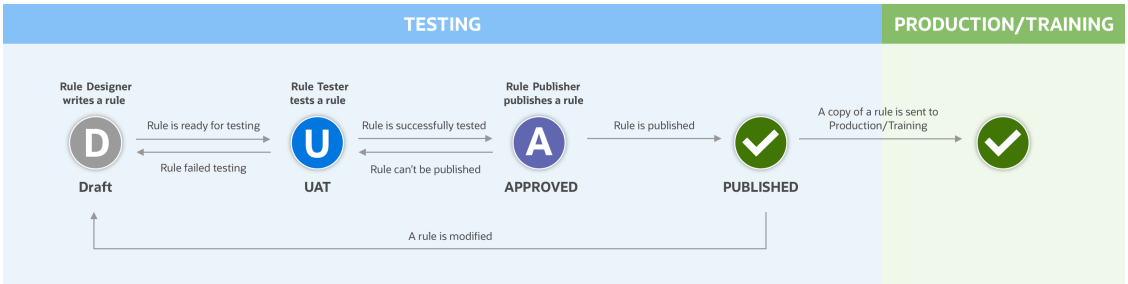
Oracle Clinical One Platform provides a user interface for creating custom rules using JavaScript. Rules are applied in all modes when published, but you can only create, test, edit, approve and publish them in Testing mode.

- Rule statuses and lifecycle
  During the lifecycle of the rule, you can create, test, approve, and publish in an iterative manner to help you refine your rules and implement them according to your product specification.
- Access the rules interface
  Oracle Clinical One Platform provides a user interface for you to create, test, approve, and publish rules. You access this dialog by navigating to a form within a visit.
- Create rules using the rule editor
  Define variables and add a Javascript expression in the rule editor to create your rules. You can create rules to calculate values, create autommated queries and send email notifications. If your rule fails to work as expected you can debug your rule to identify any errors and correct rule logic.
- Prepare your rule for testing and approval
  Your rule must be moved through each status in sequence and can be sent back for rework if later testing reveals an issue. As a rule designer, you should test your rule in Draft mode before moving it to UAT. This can help you minimize any future rework.
- Test and approve a rule
  You need to test and approve a rule before publishing it to Production. Your testing should be done before moving from Draft mode, and again before moving from UAT. This helps ensure that your rule is functioning as expected before moving to production.
- Publish rules
  Publish a rule once it is approved. A rule that reaches the Published status becomes active in all modes, regardless of the study version. You can publish a single rule or multiple rules at either form or study level.
- Modify and republish a published rule
  You can modify a published rule and re-publish it as needed using the same process you used to create the original rule. If you modify a published rule in Testing mode, changes won't appear in Production mode until you update that rule's status to **Published**.
- Disable a rule
  If you published a rule in error, users with the appropriate permission can disable that rule so it stops running in all study versions for a given mode.
- Access the Rule Management page
  Access the Rule Management page for a paticular study and manage all custom rules in different modes.
- Manage rules in Testing mode from the Rule Management page
  View and manage rules that are under development and already published in Testing mode. This includes rules in all states: *Draft*, *UAT*, *Approved*, *Published*, and *Invalid* .
- Manage published rules from the Rule Management page
  View and manage all published custom rules within a study, in either Production or Training mode.

Rule statuses and lifecycle

During the lifecycle of the rule, you can create, test, approve, and publish in an iterative manner to help you refine your rules and implement them according to your product specification.

Figure 2-1 A rule's lifecycle and what each user role has to do



A rule needs to be created in Testing mode, where it goes over different stages before it gets published. Rules are study version independent and apply to every mode once they reach the Published status. Different roles participate at the different stages of the rule's lifecycle.

Icon	Status	Description
	Draft	The rule is newly created and can be edited by a Rule Designer. Once drafted, Prepare your rule for testing and approval. If it has no syntax errors and works as expected move it to the next stage. Otherwise make edits to your rule still in draft.
	UAT	The rule has no syntax errors and is ready to be tested by a Rule Tester. Test and approve a rule for every possible scenario. If the rule fails testing you can take it back to draft mode so that the rule designer makes the necessary edits.
	Approved	The rule has no syntax errors, it has the expected results and is ready to be published by a Rule Publisher. At this point you can Publish a single rule, Publish multiple rules at the form level or Publish multiple rules at the study level.
	Published	The rule has been published by a Rule Publisher. Once a rule has reached the Published status, the rule becomes active in all modes regardless of the study version. If you edit a published rule in Testing mode, its state goes back to Draft, but this doesn't affect the published rule in Production and Training modes. To update in all modes you need to Modify and republish a published rule. If a rule is no longer needed you can Disable a rule.

Access the rules interface

Oracle Clinical One Platform provides a user interface for you to create, test, approve, and publish rules. You access this dialog by navigating to a form within a visit.

Access the rules dialog from the Oracle Clinical One Platform home page.

Before doing any rules work, you must have a study version in the Testing container that includes the following elements:

- Forms
- Visits (that contain forms)
- Subjects
- Sites associated with the study version placed in the Testing container

1. On the Home page, click the Testing Mode button () for the study you want to work on.
2. Along the top, make sure **Subjects** is selected.
3. If you have access to multiple sites for the study, select a site from the **Site** drop-down in the upper-right.
4. In the table, locate and click the visit card that you want to edit.
5. On the left side of the visit window, click the form where you want to apply your rules work.

   You can see the **Rules** pane on the right. Expand it to view and manage existing rules or create new ones.

> **Tip:**
>
> Use the **View all Rules in this form** toggle to manage all rules, including those added in other form questions.

From this interface, you can create, test, approve, and publish rules as needed.
To create a rule, select the question that must contain a rule, then click **+ Add Rule**. The Rule editor opens so you can start creating your rule, see:

- Create a rule for a calculated value
- Create a rule for an automated query
- Create a rule to send an email notification

Create rules using the rule editor

Define variables and add a Javascript expression in the rule editor to create your rules. You can create rules to calculate values, create autommated queries and send email notifications. If your rule fails to work as expected you can debug your rule to identify any errors and correct rule logic.

- Define rule variables
  As you create custom rules, define variables referring to collected data to use within your rule expression.

- Create a rule for a calculated value
  You can create a rule that enables the system to automatically calculate a value in a form. This allows you to automatically calculate certain values based on manually entered form data. reducing calculation errors and simplifying forms.
- Create a rule for an automated query
  You can create a rule that generates an automated query if the value entered by the site does not meet the acceptance criteria defined by the rule logic. The query message lets users know they should verify their entry and correct any errors.
- Create a rule to send an email notification
  You can notify designated team members by creating a rule that sends an automatic email notification when specific criteria is met. The usual steps for creating and managing custom rules still apply.
- Use predictive text to write rules
  Use predictive text to add suggested variable names, basic JavaScript syntax, Subject attributes and rule helper functions to your rule expression.
- Debug a rule
  Rule designers can debug rules by using logMsg() statements to obtain information regarding the rule logic and make sure it works as desired.

Define rule variables

As you create custom rules, define variables referring to collected data to use within your rule expression.

Custom rules are created from the Rule editor, see Access the rules interface.

As you create a rule you must enter a rule name and description before defining your rule variables, expression and action.

1. In the Rule editor, at the top, click the plus sign icon ( ) next to the Variables section title.

   A row with editable fields appears.
2. In the first field, enter a name for the variable.
3. Select a visit, a form, and a question from each drop-down. For example:

   For example:

   ```
   Var BP_Sys = Screening visit, Vital Signs form, Blood Pressure Systolic question.
   ```

Table 2-1 Variable set up for different use cases

Use case	Set up	Behavior
Retrieve data from the current visit	Select **-All Visits-** in the *visits* field.	In this scenario the variable value will be retrieved from the form in the current visit where rule is being run. This option only allows the rule to refer to forms in the same visit as the target form.

Table 2-1 (Cont.) Variable set up for different use cases

Use case	Set up	Behavior
Retrieve data from a specific visit	Select the specific visit in the *visits* field. For example the **Screening** visit.	If you select a specific 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.
Retrieve data from a form that is not in the current visit	Select **-Any Visit-**in the *visits* field.	In this scenario the variable value will be retrieved from a form that is not present in the same visit where the rule is being created. This type of variable can only be used in conjunction with getValues( ) helper function.
Retrieve visit date data as a variable	Select **Visit Date** as both *form* and *question*.	A visit's date is considered a separate form and is included in all visits. Because of this, rules that are configured with the visit date field as their target will run against all subject visits unless the rule logic dictates differently. **Caution:** Visit Date should only be used as a variable in a rule or as a rule's target when required. This is because Visit Date is a system item in its own form (not created by a study designer), and when used as a variable or a target, the system can take longer than usual to run rules. • If you need to create a custom rule to compare a Visit Date to another form question, the form question should be the target rather than the Visit Date. For example, you may need to create a custom rule that checks that the Date of Informed Consent is less than or equal to the Visit Date. Here, the Date of Informed Consent question would be the target. • Visit Date should only be used as a variable to trigger a calculation rule when no other form questions can be used as the trigger.

For each variable you want to create, repeat the steps above as needed.

- Create a rule for a calculated value
- Create a rule for an automated query

- Create a rule to send an email notification

Create a rule for a calculated value

You can create a rule that enables the system to automatically calculate a value in a form. This allows you to automatically calculate certain values based on manually entered form data. reducing calculation errors and simplifying forms.

Want to see how to perform this task? Watch the video below.

Video

You can also create rules that raise automatic queries or send an e-mail notification. See:

- Create a rule for an automated query
- Create a rule to send an email notification

You must have a study version in the Testing container that includes the required elements. You access the Rules interface from a specific study version and site as described in Access the rules interface.

1. Navigate to your desired study in Testing mode and select a site (if you have access to multiple sites).
2. In the table, locate and click the visit card that you want to edit.
3. On the left, click the form for which you want to create the rule.
4. Select the question that should contain a rule and where the calculated value should display.
5. On the right, expand the **Rules** pane, and click **Add Rules.**
6. In the **Rule editor** complete the following fields. Then click **Next**.

Field	Description
Rule Name	Enter a name for your rule. Each name must be unique within a study and its number of characters should not exceed **512**.
Description	Enter a short description of your rule that doesn't exceed **4000** characters. This field isn't mandatory. However, adding a description can help you distinguish between each rule and its purpose in a study. This is helpful when you want to reuse a rule.
Unblock Form	Turn this toggle on if you want to allow site users to edit and save forms without being blocked while the rule is running. **Note:** Complex rules take more time to run. Site users can still enter values and save the form while the rule is running. However, form updates generated by complex rules can be delayed and may only appear after a page refresh.

7. In the Variable section, Define rule variables.
8. In the Expression section, enter the JavaScript expression that will be evaluated to a value.

For example, to calculate BMI:

```
return weight/(height*height);
```

> **Note:**
>
> - If the action item is dynamic the rule expression should include logic to ensure the item is visible before the calculated value is populated. The calculated value will not be updated if the target item is hidden.
> - By default, the rule will run against every visit in the study that contains the form and in all study versions.
>   - To limit which study version the rule is run against, use the `isStudyVersion()` helper function.
>   - If you need your rule to apply and be executed only against an specific visit, you can use the `getCurrentVisitPropertyValue (Control)` helper function.

> **Tip:**
>
> A predictive text feature is available as you type, with available Subject attributes, rule helper functions, and more. See Use predictive text to write rules.

9. From the **Action** drop-down, choose **Calculate Value**.

   This allows the system to automatically calculate a value and populate a read-only item with the result. You must also

> **Note:**
>
> If the rule expression contains syntax errors the **Rule Editor** marks them for you to correct.

10. Select the **Answer Type** from nthe dropdown:
    - **Number**
    - **Text**
    - **Choice**
11. If your answer type is *Number*, select the **Data Format** for the calculated value.

    For example, you can calculate the Body Mass Index (BMI) based on a subject's height and weight and set the format to one decimal point as "*1.0*".
12. Click **Save**.

Your rule is now created and in Draft status.
To make your rule available in production. You must test, approve, and publish your rule. Rules are study version independent and will apply in every mode once they reach the Published state.

If you want to delete this rule and start over, click the menu icon (), select **Delete**.

For examples, and more information on developing custom rules, see:

- Rule statuses and lifecycle
- Test and approve a rule
- Publish a single rule
- Publish multiple rules at the form level
- Publish multiple rules at the study level
- Rules helper function reference
- Rules examples

Create a rule for an automated query

You can create a rule that generates an automated query if the value entered by the site does not meet the acceptance criteria defined by the rule logic. The query message lets users know they should verify their entry and correct any errors.

> **Note:**
>
> If the rule to create automated queries is published for all data, all published rules will be re-run against all data in the study. Meaning that all queries in that study that were previously closed by an user without data update, will be re-opened.
> To avoid this, make sure you publish the rule for future data only. For more information see Publish a single rule.

If a rule that triggers an assigned query is updated to add a new role in the assignment list, then this update will only become available when a new query is created. Existing open queries will not be impacted and they will not become assigned to the new role you add in the assignment list. If a query is closed and re-opened, that query becomes visible to the newly added roles.

You can also create a rule that calculates a value for use in a form or sends an e-mail notification. See:

- Create a rule for a calculated value
- Create a rule to send an email notification

You must have a study version in the Testing container that includes the required elements. You access the Rules interface from a specific study version and site as described in Access the rules interface.

1. Navigate to your desired study in Testing mode and select a site (if you have access to multiple sites).
2. In the table, locate and click the visit card that you want to edit.
3. On the left, click the form for which you want to create the rule.
4. Select the question that should contain the rule and to which the query will be raised upon if applicable.
5. On the right, expand the **Rules** pane, and click **Add Rules**.
6. In the **Rule editor** complete the following fields. Then click **Next**.

Field	Description
Rule Name	Enter a name for your rule. Each name must be unique within a study and its number of characters should not exceed **512**.
Description	Enter a short description of your rule that doesn't exceed **4000** characters. This field isn't mandatory. However, adding a description can help you distinguish between each rule and its purpose in a study. This is helpful when you want to reuse a rule.
Unblock Form	Turn this toggle on if you want to allow site users to edit and save forms without being blocked while the rule is running. **Note:** Complex rules take more time to run. Site users can still enter values and save the form while the rule is running. However, form updates generated by complex rules can be delayed and may only appear after a page refresh.

7. In the Variable section, Define rule variables.
8. In the Expression section, enter the JavaScript expression that will be evaluated to raise a query.

   For example:

```
if (diastolic>systolic){
    return false;        //query is raised when return false condition is
met
}
else{
    return true;
}
```

Note:

- If the action item is dynamic the rule expression should include logic to ensure the item is visible before the query is triggered.
- By default, the rule will run against every visit in the study that contains the form and in all study versions.
  - To limit which study version the rule is run against, use the `isStudyVersion()` helper function.
  - If you need your rule to apply and be executed only against an specific visit, you can use the `getCurrentVisitPropertyValue (Control)` helper function.

> **Tip:**
>
> A predictive text feature is available as you type, with available Subject attributes, rule helper functions, and more. See Use predictive text to write rules.

9. From the **Action** drop-down select the query type you want.
   - **Create Query**: select this option to automatically generate a query each time the value returned by the rule expression is **False**.
   - **Create Assigned Query**: select this option to assign the query to a specific study role. For this option, you must click the field displayed and select one or more study roles from the roles drop-down list. The query will be assigned to the selected roles only.

   > **Note:**
   >
   > The Roles drop-down list contains only the study roles that have been created in the study. The template study roles are not included.

10. Enter a query message in the appropriate text-box.

    Both query types require you to add a query message.

11. Click **Save**.

Your rule is now created and in Draft status.
To make your rule available in production. You must test, approve, and publish your rule. Rules are study version independent and will apply in every mode once they reach the Published state.

If you want to delete this rule and start over, click the menu icon (), select **Delete**.

For examples, and more information on developing custom rules, see:

- Rule statuses and lifecycle
- Test and approve a rule
- Publish a single rule
- Publish multiple rules at the form level
- Publish multiple rules at the study level
- Rules helper function reference
- Rules examples

Create a rule to send an email notification

You can notify designated team members by creating a rule that sends an automatic email notification when specific criteria is met. The usual steps for creating and managing custom rules still apply.

You can also create a rule that calculates a value for use in a form or generates an automated query. See:

- Create a rule for a calculated value
- Create a rule for an automated query

> **Caution:**
>
> Make sure you do not include any personally identifiable information (PII) data in the body of your email notification.

> **Note:**
>
> In Draft mode, study designers can additionally create a rule to send notifications upon data entries and response changes. Reach out to your study design team or see Define a Send Notification rule.

You must have a study version in the Testing container that includes the required elements. You access the Rules interface from a specific study version and site as described in Access the rules interface.

To create a rule to send an automatic email notification:

1. Navigate to your desired study in Testing mode and select a site (if you have access to multiple sites).
2. In the table, locate and click the visit card that you want to edit.
3. On the left, click the form for which you want to create the rule.
4. Select the question that should contain a rule.
5. On the right, expand the **Rules** pane, and click **Add Rules**.
6. In the **Rule editor** complete the following fields. Then click **Next**.

Field	Description
Rule Name	Enter a name for your rule. Each name must be unique within a study and its number of characters should not exceed **512**.
Description	Enter a short description of your rule that doesn't exceed **4000** characters. This field isn't mandatory. However, adding a description can help you distinguish between each rule and its purpose in a study. This is helpful when you want to reuse a rule.
Unblock Form	Turn this toggle on if you want to allow site users to edit and save forms without being blocked while the rule is running. **Note:** Complex rules take more time to run. Site users can still enter values and save the form while the rule is running. However, form updates generated by complex rules can be delayed and may only appear after a page refresh.

7. In the Variable section, Define rule variables.
8. In the Expression section, enter the JavaScript expression that will be evaluated to send a notification.

> **Note:**
>
> By default, the rule will run against every visit in the study that contains the form and in all study versions.
>
> - To limit which study version the rule is run against, use the `isStudyVersion()` helper function.
> - If you need your rule to apply and be executed only against an specific visit, you can use the `getCurrentVisitPropertyValue (Control)` helper function.

> **Tip:**
>
> A predictive text feature is available as you type, with available Subject attributes, rule helper functions, and more. See Use predictive text to write rules.

9. From the **Action** drop-down, select **Send Notification**.
10. In the **Subject Line** field, write a subject for your notification email.
11. Type the email addresses you want to send this notification to in the text box underneath the **Action** drop-down.
12. To test the email notification, type the email addresses you want to send the notification to in the appropriate text box for testing email addresses.

> **Note:**
>
> Any email addresses will receive a notification when the rule is generated either in Testing or Production mode.

13. Write the notification message you want the users to receive.

    Notification details will always be included in the email below the notification message.
14. Click **Save**.

To make your rule available in production. You must test, approve, and publish your rule. Rules are study version independent and will apply in every mode once they reach the Published state.

If you want to delete this rule and start over, click the menu icon (), select **Delete**.

For examples, and more information on developing custom rules, see:

- Rule statuses and lifecycle
- Test and approve a rule
- Publish a single rule
- Publish multiple rules at the form level
- Publish multiple rules at the study level
- Rules helper function reference
- Rules examples

Use predictive text to write rules

Use predictive text to add suggested variable names, basic JavaScript syntax, Subject attributes and rule helper functions to your rule expression.

When you write rule expressions, a predictive text feature is available as you type, listing available Subject attributes and rule helper functions with their descriptions. This feature also suggests variable names and some standard JavaScript syntax, when applicable.

Predictive text is only available in the expression text box of the rule editor, when creating rules. See Create rules using the rule editor.

1. With the Rule Editor open, start typing in the Expression text box.

   For any string you enter, you will get a list of releted options.

2. Select a suggestion from the list, either using the mouse or the arrow keys on your keyboard.

   > **Tip:**
   >
   > A short description is available as you hover over a list item. To see the expanded description, including parameters description and return value details, if applicable, click on the right arrow icon (>)

   or use the right arrow key, then press the down arrow key on your keyboard.

Selecting an item from the list adds it to your expression, and for rule helper functions it includes its parameters as placeholders for you to complete.
For examples, and more information on developing custom rules, see:

- Create a rule for a calculated value
- Create a rule for an automated query
- Create a rule to send an email notification
- The Subject Object
- Rules helper function reference
- Rules examples

Debug a rule

Rule designers can debug rules by using logMsg() statements to obtain information regarding the rule logic and make sure it works as desired.

The Debug functionality is only available in Testing mode. You must have a study version in the Testing container that includes the required elements. You access the Rules interface from a specific study version and site as described in Access the rules interface.

> **Note:**
>
> Before you can debug a rule, test data must have been entered for the rule. The Debug button runs the rule against data entered for the subject for which the rule was accessed, this includes relevant data entered in other visits.

1. Navigate to your desired study in Testing mode and select a site.
2. In the table, locate and click the visit card that you want to edit.
3. On the left side of the visit window, click the form containing the rule you want to work with.
4. Select a question and expand the **Rules** pane on the right to view the rules it contains.

   > **Tip:**
   >
   > When you click on any question you can activate **View all Rules in this form** option to get a list of all the rules added to any question in that form

5. Open the Rule Editor:

   - **To edit an existing rule**: For the given rule click on the menu icon () then select **Edit**.
   - **To create a new rule**: See Create rules using the rule editor.

6. Add log statements where needed.

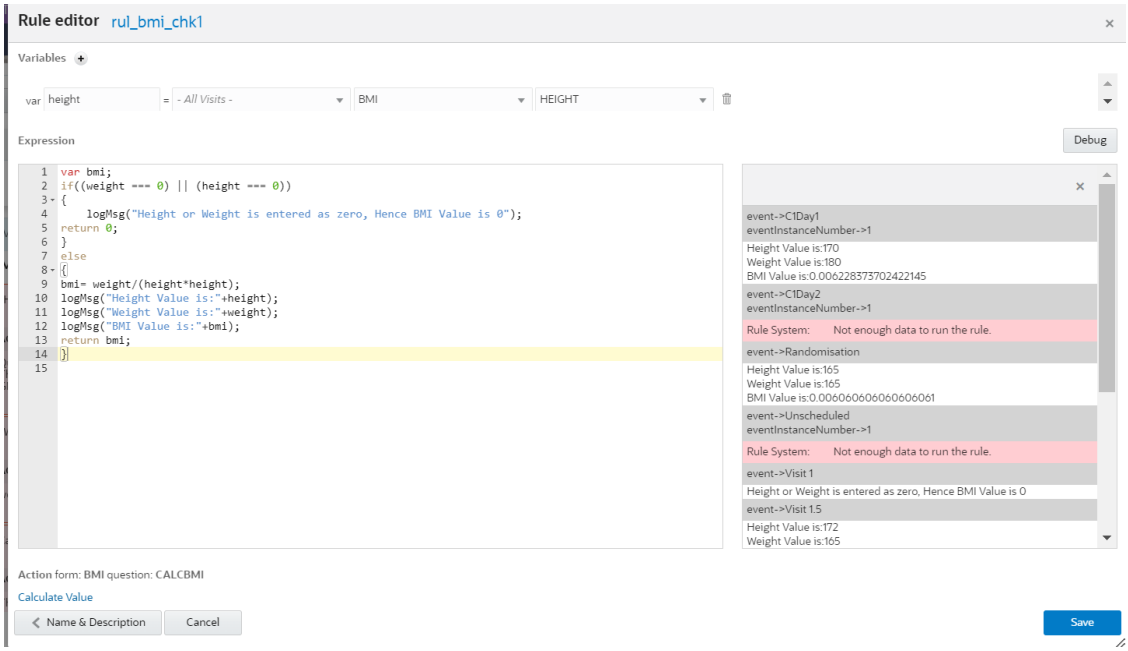
   For more information on how to use the log helper function properly see logMsg().

   > **Tip:**
   >
   > A predictive text feature is available as you type, with descriptions of this and all other rule helper functions. This allows you to select an existing function from the list and add it to your expression, including parameters as placeholders for you to complete.
   > Predictive text also suggests variable names and some standard JavaScript syntax.

7. Click **Debug**.

   Clicking **Debug** automatically saves changes to the JavaScript expression. All rule saving requirements must be fulfilled.

   > **Note:**
   >
   > Any compilation error or missing requirement will not allow the debug process to complete.



The log window appears on the right in the Rule editor.

Rule runs against data of the selected subject and outputs the calls made to the `logMsg()` helper function within the rule and for all instances. This means that, if the rule belongs to a form in multiple visits or a question in repeating sections, the rule is analyzed for all impacted visits and repeating sections. The output from each rule instance is listed in the log window.

If any exception is encountered, the stack trace details and error messages are displayed in the log window.

Review the log messages and make any necessary changes to the rule expression. Add additional log statements as required and repeat this process until the rule's path of execution is as expected. Then **Save** your changes and exit the rule editor.

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

After saving, you must test rule performance to be as expected. To make your rule available in Production, you must test, approve, and publish your rule.

For examples, and more information on developing custom rules, see:

- Rule statuses and lifecycle
- Prepare your rule for testing and approval
- Test and approve a rule
- Publish a single rule
- Publish multiple rules at the form level
- Publish multiple rules at the study level
- Rules helper function reference

- Rules examples

Prepare your rule for testing and approval

Your rule must be moved through each status in sequence and can be sent back for rework if later testing reveals an issue. As a rule designer, you should test your rule in Draft mode before moving it to UAT. This can help you minimize any future rework.

Before your rule can be promoted to UAT and Approved, a rule must contain a valid expression and at least one action. The validation in the Rules editor can help you ensure your rules are valid. For information on creating your rules, see:

- Create a rule for a calculated value
- Create a rule for an automated query
- Create a rule to send an email notification

For information on accessing the rules interface, see Access the rules interface.

1. Navigate to your desired study and select a site (if you have access to multiple sites).
2. In the table, locate and click the visit card that you want to edit.
3. On the left side of the visit window, click the form for which you want to test a rule.
4. Test your rule.

   Your rule should be in Draft at this time.

   a. Enter a value for the question containing the rule.

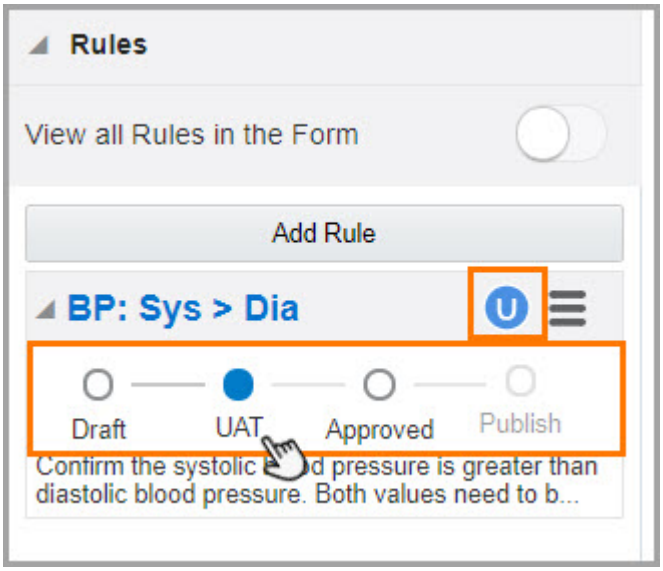
   b. Click **Save**.

   The rule will run against the value. You should evaluate the action taken by the rule and confirm that it is correct.
5. Repeat the testing steps for all possible scenarios and values and confirm your results.
6. Once you have confirmed that your rule is functioning as expected, you can move it to UAT.

   a. On the **Rules** pane, click the rule status icon next to the rule name to show the status slider.

   b. Move the slider from **Draft** to **UAT**.

Figure 2-2 The rule status slider when the rule is in UAT



If your rule is not functioning as expected, do not move it to UAT. See the following topics for examples and more information on developing rules:

- Debug a rule
- Rules helper function reference
- Rules examples

Test and approve a rule

You need to test and approve a rule before publishing it to Production. Your testing should be done before moving from Draft mode, and again before moving from UAT. This helps ensure that your rule is functioning as expected before moving to production.

Want to see how to perform this task? Watch the video below.

Video

Do I have to do anything before performing this task? Before you begin your testing, work with your user administrator to make sure you're assigned the Rule Tester role and make sure you have prepared the rule for testing and approval (see Prepare your rule for testing and approval).

1. Navigate to your desired study and select a site (if you have access to multiple sites).
2. In the table, locate and click the visit card that you want to edit.
3. On the left side of the visit window, click the form for which you want to test a rule.

   Your rule should be in UAT at this time.
4. Enter a value for the question containing the rule.
5. Click **Save**.

   The rule will run against the value. You should evaluate the action taken by the rule and confirm that it is correct.
6. Repeat the testing steps for all possible scenarios and values and confirm your results.
7. If the rule generates the expected result for all scenarios:

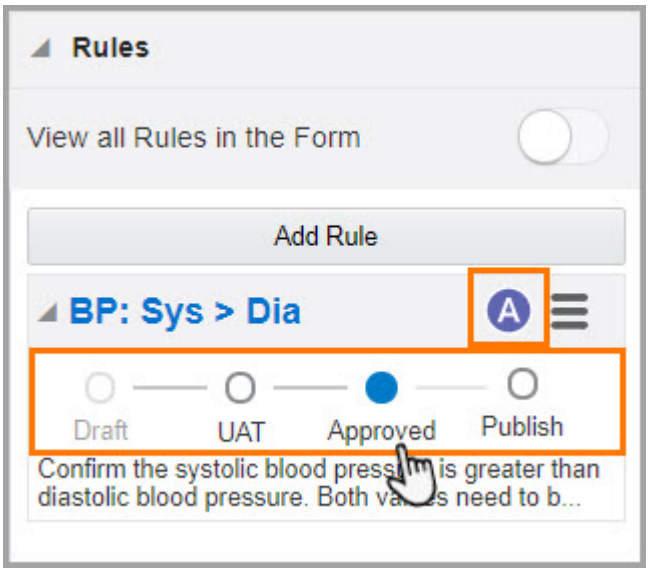
a. On the **Rules** pane, next to the rule's name, click the rule status icon to show the status slider.

b. Move the slider from **UAT** to **Approved**.

> **Tip:**
>
> Make sure you move a Form or a Question in the the **Approved** container. The rule will show only in the Design and Testing tabs. If you omit to move them, the Production and Training tabs will show the forms and questions incorrect as N/A.

> **Note:**
>
> Click the menu icon ( ) and select **View** to see a rule's details in read-only mode.

Figure 2-3 Figure 3-2 The rule status slider when a rule is approved



> **Note:**
>
> You can move the slider back to **Draft** at anytime.

If the rule doesn't generate the expected result, return the status slider to **Draft** and notify the rule designer that rework is needed.

If the rule works as intended, you can move on to publishing. See:

- Publish a single rule
- Publish multiple rules at the form level
- Publish multiple rules at the study level

Publish rules

Publish a rule once it is approved. A rule that reaches the Published status becomes active in all modes, regardless of the study version. You can publish a single rule or multiple rules at either form or study level.

- Publish a single rule
  Publish a rule to make it available in every mode. You can publish rules one at a time using the rules slider in the Rules interface.
- Publish multiple rules at the form level
  Publish rules to make them available in every mode. The rules interface gives you the ability to publish all of the approved rules included in a form at the same time. This bulk approval functionality saves you time when you have a number of rules that need to be published.
- Publish multiple rules at the study level
  Publish rules to make them available in every mode. The rules interface gives you the ability to publish all approved rules within a study at the same time. This bulk approval functionality saves you time when you have a number of rules that need to be published.

Publish a single rule

Publish a rule to make it available in every mode. You can publish rules one at a time using the rules slider in the Rules interface.

> **Note:**
>
> Rules are study version independent and will apply in every mode once they reach the Published state.

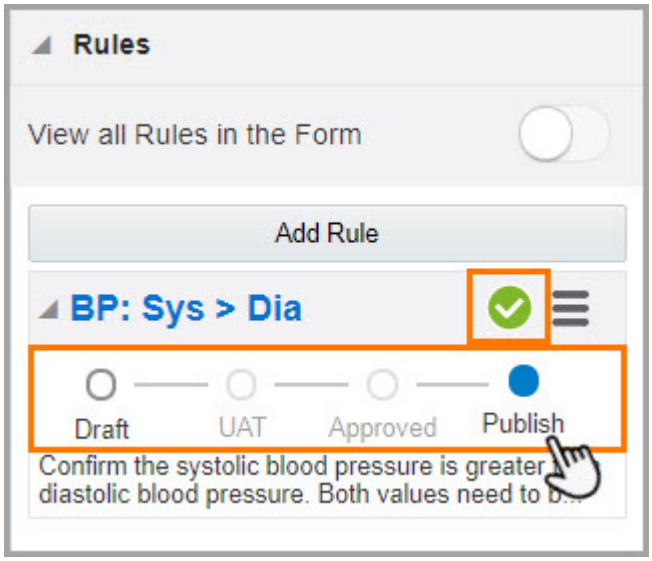
You can also publish multiple rules at the same time. See Publish multiple rules at the form level or Publish multiple rules at the study level. Publishing a single rule or publishing multiple rules at the form level allows you more control over how newly published rules are run. Publishing multiple rules at the study level may run rules against existing data and for newly entered data, too.

Work with your user administrator to make sure you're assigned the Rule Publisher role. Before publishing a rule, it must be in the Approved state. See Test and approve a rule.

1. Navigate to your desired study and select a site (if you have access to multiple sites).
2. In the table, locate and click the visit card that you want to edit.
3. On the left side of the visit window, click the form that includes the rules that you want to publish.
4. On the Rules pane, select the rule you want to publish.
5. Click the rule status icon to open the status slider.
6. Click the circle icon to move the rule to **Publish**.

> **Note:**
>
> Once you move the rule to **Publish** you can't manually move it back to a previous status in the rules status slider.

Figure 2-4 Figure 3-4 The rule status slider when publishing a rule



7. In the confirmation window, select whether you want to run the rule **On Future Data** only or **On All Data**:
   - **On All Data**: runs the rule on exisiting subjects and visits, as well as those entered in the future.
   - **On Future Data**: runs the rule only on subjects and visits entered from the point forward the rule is published.

Once a rule is published, it cannot be retracted. The only way to change a published rule is to edit it and re-publish the new version.

If you publish a rule by mistake, see Disable a rule.

Publish multiple rules at the form level

Publish rules to make them available in every mode. The rules interface gives you the ability to publish all of the approved rules included in a form at the same time. This bulk approval functionality saves you time when you have a number of rules that need to be published.

> **Note:**
>
> Rules are study version independent and will apply in every mode once they reach the Published state.

Want to see how to perform this task? Watch the video below.

Video

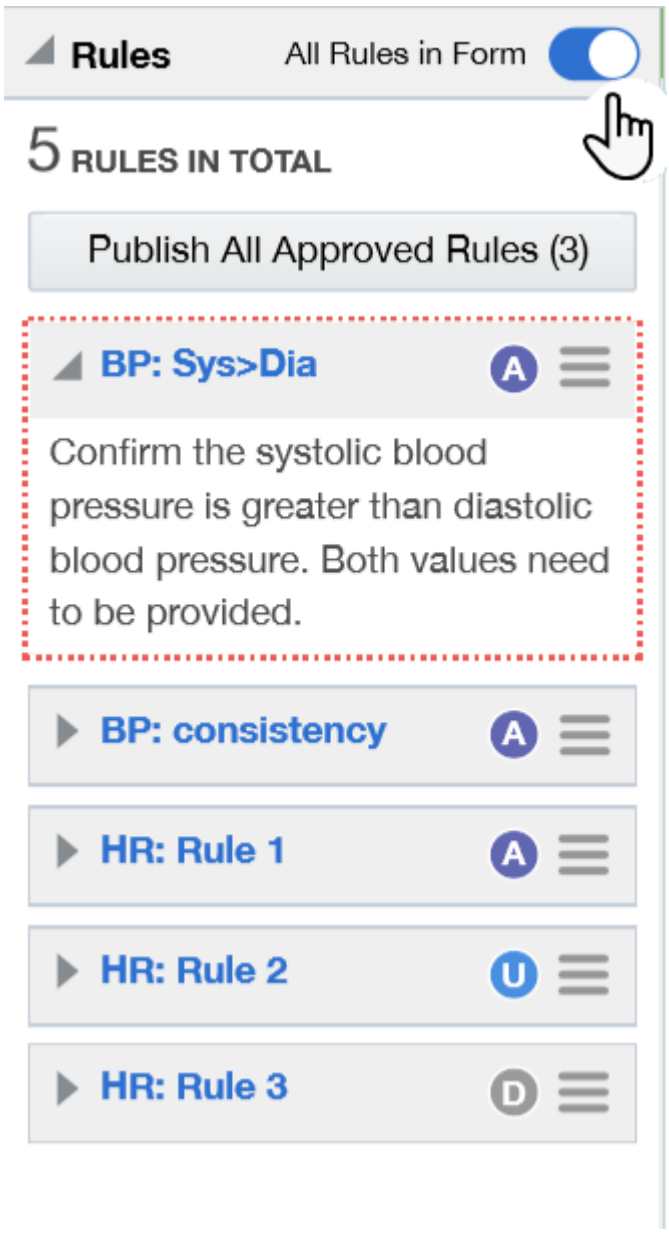
You can also publish a single rule or multiple rules at the same time at the study level. See Publish a single rule or Publish multiple rules at the study level.

Work with your user administrator to make sure you're assigned the Rule Publisher role. Before publishing a rule, it must be in the Approved state. See Test and approve a rule.

1. Navigate to your desired study and select a site (if you have access to multiple sites).
2. In the table, locate and click the visit card that you want to edit.
3. On the left side of the visit window, click the form that includes the rules that you want to publish.
4. At the top of the **Rules** pane, click the **All Rules in the Form** toggle.

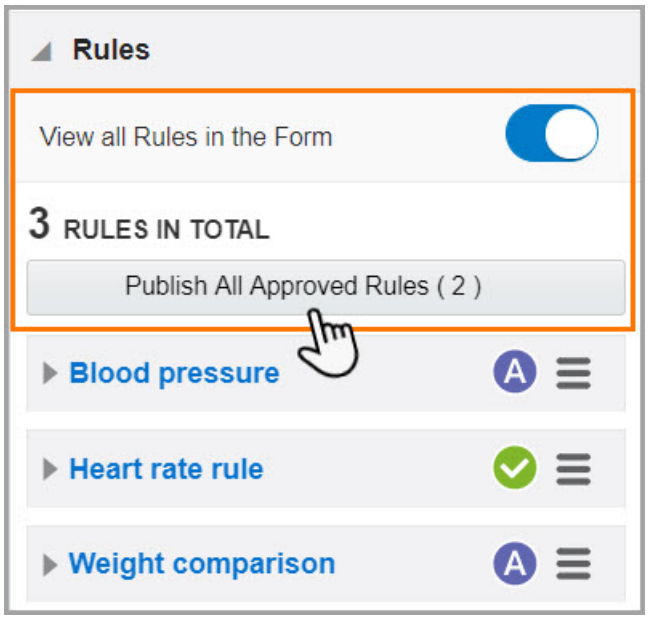
   You can now view all rules included in the form and their associated statuses.

Figure 2-5 How to show all rules in a form



5. On the Rules pane, click the **View all Rules in the Form** toggle button.
6. Click **Publish All Approved Rules**.

Figure 2-6 Figure 3-5. Publish all approved rules button



Once a rule is published, it cannot be retracted. The only way to change a published rule is to edit it and re-publish the new version.

If you publish a rule by mistake, see Disable a rule.

Publish multiple rules at the study level

Publish rules to make them available in every mode. The rules interface gives you the ability to publish all approved rules within a study at the same time. This bulk approval functionality saves you time when you have a number of rules that need to be published.

Rules are study version independent and will apply in every mode once they reach the Published state.

> **Note:**
>
> When multiple published rules run or re-run at a study level, they run on all subject data, both existing and new.

You can also publish a single rule or multiple rules at the same time at the form level. See Publish a single rule or Publish multiple rules at the form level.

Work with your user administrator to make sure you're assigned the Rule Publisher role. Before publishing a rule, it must be in the Approved state. See Test and approve a rule.

1. Select the **Home** tab.
2. On the Home page, click the pencil button ( ) for the study you are working on.

3. Drag the study from the **Testing** container to **Approved**.

   All rules with an **Approved** status are published in Production.

> **Note:**
>
> You can publish rules in Production by following these steps only if the study is already in the Approved container. If not, you need to move the study from Testing to Approved to publish all rules in Production.

Once a rule is published, it cannot be retracted. The only way to change a published rule is to edit it and re-publish the new version.

If you publish a rule by mistake, see Disable a rule.

Modify and republish a published rule

You can modify a published rule and re-publish it as needed using the same process you used to create the original rule. If you modify a published rule in Testing mode, changes won't appear in Production mode until you update that rule's status to **Published**.

> **Note:**
>
> Rules are study version independent and will apply in every mode once they reach the Published state.

Want to see how to perform this task? Watch the video below.

Video

As a rules designer you can edit a rule in the same rules interface you used to create the rule.

Modify the study version in the Testing container. For information on accessing the rules interface, see Access the rules interface.

1. Navigate to your desired study and select a site (if you have access to multiple sites).
2. In the table, locate and click the visit card that you want to edit.
3. On the left side of the visit window, click the form for which you want to create the rule.
4. Select the question that should contain a rule.
5. On the right, expand the **Rules** pane, and click the menu icon ( ).
6. Select **Edit**.
7. In the Rule editor, make your changes.

   You can:

   - Create a rule for a calculated value
   - Create a rule for an automated query
   - Create a rule to send an email notification
8. Click **Save**.

The rule's status is updated to **Draft**. The rule currently published in Production isn't impacted by this change.

> **Note:**
>
> If you want to start over, you can always delete a rule with a **Draft** status.

To make your updated rule available in Production, you must test, approve, and publish the newly modified rule. However, as a Rule Designer, when you publish a modified rule, a confirmation message appears asking you to choose if you want to execute the rule only on future data or on all data including data that was already collected. When you click **On All Data**, the rule re-runs for all subjects and visits for which it has previously run, as well as for data that is collected from that moment forward. For example, if the rule is a calculation for Body Mass Index (BMI) and has previously calculated the value for two subjects, then the rule will re-run and re-calculate BMI for these two subjects. The rule will also run for data collected in the future. When you click **On Future Data Only**, the rule runs only on data entered from that point forward. It is not re-run on previously collected data.

Republished rules overwrite existing rules for all study versions currently running in Production.

See:

- Prepare your rule for testing and approval
- Test and approve a rule
- Publish a single rule
- Publish multiple rules at the form level
- Publish multiple rules at the study level

Disable a rule

If you published a rule in error, users with the appropriate permission can disable that rule so it stops running in all study versions for a given mode.

> **Note:**
>
> When you disable a rule, it becomes inactive only in the mode where you disabled it. For instance, if you disable a rule in Production mode, you deactivate it only in Production but you can still test it in Testing mode.

On the Rule Management page, you can view all your study's custom rules by mode. From that interface you can also disable (and enable) rules, individually and in bulk, for any mode.

- See Manage rules in Testing mode from the Rule Management page to view and manage rules under development and already published in Testing mode.
- See Manage published rules from the Rule Management page to view and manage active rules in Production and Training modes.

1. Access your study in a specific mode.
2. If you have access to multiple sites for the study, select a site from the **Site** drop-down.
3. In the table, locate and click the visit card that you want to edit.

4. Select the question that contains the rule that you want to disable.
5. On the right-side panel, expand the **Rules** section.
6. Click the menu icon for the rule that you want to disable and select **Disable**.

The rule is now disabled in all study versions running for the given mode.
To disable a rule in any other mode, access you study in that specific mode and repeat the steps above.

Access the Rule Management page

Access the Rule Management page for a paticular study and manage all custom rules in different modes.

Once a rule is published, there are three versions of the rule (Testing, Training, Production) and all can be managed separately.

- See Manage rules in Testing mode from the Rule Management page to view and manage rules under development and already published in Testing mode.
- See Manage published rules from the Rule Management page to view and manage active rules in Production and Training modes.

> **Note:**
>
> Access to the Rule Management page and the corresponding tabs is based on your permissions assigned on each mode. Reach out to you Oracle Project Manager or user administrator to make sure you have the appropriate permissions.

1. On the Home page, locate your study and click the pencil button ().
2. Click **Rule Management**.
3. On the new screen, navigate to the appropriate tab.
   - Go to the **Design & Testing** tab to manage rules in Testing mode. This includes rules in all states (*Draft*, *UAT*, *Approved*, *Published* and *Invalid*).
   - Go to the **Production and Training** tab to manage rules in Production and Training modes.

Manage rules in Testing mode from the Rule Management page

View and manage rules that are under development and already published in Testing mode. This includes rules in all states: *Draft*, *UAT*, *Approved*, *Published*, and *Invalid* .

Once a rule is published, there are three versions of the rule (Testing, Training, Production) and all can be managed separately.

All rules in Testing mode are listed with the following details:

- Rule name and description.
- Target form.
- Target question.
- Rule state.

- If rule is enabled or disabled in Testing mode.
- If, when published, rule will run on all data or future data only.

> **Note:**
>
> Only applies to the version of the rule in Testing mode.

- Last modified date and user.

> **Tip:**
>
> If you can't see all columns, make sure you scroll all the way to the right.

See Manage published rules from the Rule Management page to view and manage active rules in Production and Training modes.

1. Access the Rule Management page.
2. Make sure you are on the **Design & Testing** tab.
3. Choose how to view your rules:

   By default, rules are ordered by target form and question.

   - To order by a different attribute, hover over the given column header and click the arrow ( ) to the right of the column name.
   - To search for a specific rule, type its name in the search bar and press Enter on your keyboard..
4. To manage rules in Testing mode you have the following options:

Option	Description
Enable/Disable a rule	a. Locate the rule. b. Click **Enabled** or **Disabled** in the *Testing Mode* column. A blue background indicates the current status.
Enable/Disable multiple rules	*All rules are enabled by default.* a. Use the checkbox to the left of each rule to select all the rules you want to enable or disable. b. Click the **Enable/Disable** drop-down. c. Select **Enable** or **Disable** accordingly.

Option	Description
Update on what data the rule will run when published	*Set to All Data by default.* a. Locate the rule. b. Click **All Data** or **Future Data** in the *When published, Run on* column. A blue background indicates the current selection. ***Note**: This setting only applies when the rule gets published by moving the study version (where the rule exists in an approved state) from Testing to Approved container. If a rule gets published using the slider in testing mode, the selection made then to run either on All Data or Future Data Only controls the rule's behavior in active mode, regardless of the configuration set in the Rule Management page. Once a rule is published and active, modifying this setting only applies to the version of the rule in the Testing mode.*
View rule in the Rule Editor	a. Locate the rule. b. Click the **Action** drop-down to the right. c. Select **View Rule**.
Locate a published rule on the Production & Training tab	*Only available for published rules.* a. Locate the rule. b. Click the **Action** drop-down to the right. c. Select **Find in Production & Training**.

Manage published rules from the Rule Management page

View and manage all published custom rules within a study, in either Production or Training mode.

Once a rule is published, there are three versions of the rule (Testing, Training, Production) and all can be managed separately.

All active rules (in Production and Training modes) are listed with the following details:

- Rule name and description.
- Target form.
- Target question.
- If rule is enabled or disabled in Production mode.
- If rule is enabled or disabled in Training mode.
- Last published date and user.

> **Note:**
>
> This field displays the details of when the rule became active in poduction, this is when study version with the rule in approved or published state got moved to the Approved container.

> **Tip:**
>
> If you can't see all columns, make sure you scroll all the way to the right.

See Manage rules in Testing mode from the Rule Management page to view and manage rules under development and already published in Testing mode.

1. Access the Rule Management page.
2. Navigate to the **Production & Training** tab.
3. Choose how to view your rules:

   By default, rules are ordered by target form and question.

   - To order by a different attribute, hover over the given column header and click the arrow ( ) to the right of the column name.
   - To search for a specific rule, type its name in the search bar and press Enter on your keyboard..
4. To manage rules in Production and training modes you have the following options:

Option	Description
Enable/Disable a rule for a given mode	a. Locate the rule. b. Click **Enabled** or **Disabled** in the respective column: • *Production Mode* • *Training Mode* A blue background indicates the current status.
Enable/Disable multiple rules	a. Use the checkbox to the left of each rule to select all the rules you want to enable or disable. b. Click the **Enable/Disable** drop-down. c. Select **Enable** or **Disable** accordingly. d. Select on which mode do you want to apply the changes: • **Production & Training Mode** • **Production Mode** • **Training Mode**
View rule in the Rule Editor	a. Locate the rule. b. Click the **Action** drop-down to the right. c. Select **View Rule**.

Option	Description
Locate a published rule on the Design & Testing tab	a. Locate the rule. b. Click the **Action** drop-down to the right. c. Select **Find in Design & Testing**.

3

Rules helper function reference

General expressions

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

- Comparison
  Use this expression when you want to compare two variables and generate a query when the result returns **false**.
- Conversion
  Automatically convert values such as Fahrenheit into Celsius or Celsius into Fahreinheit degrees.
- 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.
- Choice
  Search for one of the answers in a choice question.
- Compare dates with different formats
  Compare two dates with different formats. This can help confirm or resolve date inconsistencies.
- 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.

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

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 3-1 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 3-2 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 3-3 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.

Example 3-4 Temperature range check

In this example, we are looking for temperatures outside of a certain range.

```
//validate if a given a temperature (temp_f) is outside of a specified range

var upper = 99.1;
var lower = 97.8;
if (temp_f > lower && temp_f < upper){
  return false;
}
```

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.

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.
- dateDiffInDays( )
  Calculate the date difference between two dates measured in days.
- timeDiffInHours( )
  Calculate the time difference between two date or date/time values in hours.
- timeDiffInMinutes( )
  Calculate the time difference between two date or date/time values in minutes.
- timeDiffInSeconds( )
  Calculate the time difference between two date or date/time values in seconds.
- areDatesEqual( )
  Compare two dates to determine if they are equivalent.
- isDateInRange( )
  Verify if a date falls within a given range.
- areDateTimesEqual( )
  Compare two date or date/time values to determine if they are equivalent.
- isTimeInRange( )
  Verify if a date or date/time value falls within a given range.
- addDays( )
  Add a specific number of days to a date value.
- addTimeInHours( )
  Add a specific number of hours to a date or date/time value.
- addTimeInMinutes( )
  Add a specific number of minutes to a date or date/time value.
- getDateDMYFormat( )
  Return a date or datetime in DD-Mon-YYYY format, including time elements if applicable. This function supports 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.

The `dateDiffInYears( )` helper function is invoked with starting and ending dates passed in as parameters. The function returns a negative or positive number value indicating the difference between the two dates in years.

> **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.

This function is only used to compare variables of type date that do not contain time elements and do not include partial dates. When using a date/time type parameter, function considers only date part and ignores time elements.

> **Tip:**
>
> - 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.

Syntax

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

Parameters

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

Parameter	Required/Optional	Description
`toDate`	Required	Ending date value.
`fromDate`	Required	Starting date value.

Return value

Number that represents the difference between the passed-in dates in years. This number can be positive or negative.

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

Examples

Example 3-5 Difference between two Date items

```
// Given 2 form questions of type Date (with no time elements) are defined in
the rule as variables:
return dateDiffInYears(dateItem1, dateItem2);
```

Example 3-6 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.

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

> **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.

This function is only used to compare variables of type date that do not contain time elements and do not include partial dates. When using a date/time type parameter, function considers only date part and ignores time elements.

> **Tip:**
>
> - 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.

Syntax

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

Parameters

Parameter	Required/Optional	Description
`toDate`	Required	Ending date value.
`fromDate`	Required	Starting date value.

Return value

Number that represents the difference between the passed-in dates in days. This number can be positive or negative.

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

Examples

Example 3-7 Difference between two Date items

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

Example 3-8 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 date or date/time values in hours.

The `timeDiffInHours( )` helper function is invoked with starting and ending dates passed in as parameters. The function returns a negative or positive number value indicating the difference between the two dates in hours.

> **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.

When using date type variables with no time elements, function considers time as '00:00:00'.

Syntax

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

Parameters

Parameter	Required/Optional	Description
`toDate`	Required	Ending date value.
`fromDate`	Required	Starting date value.

Return value

Number that represents the difference between the passed-in dates in hours. This number can be positive or negative.

- If the number value returned is a negative or zero value, means that `toDate` is before or the same as the `fromDate`.

- If the function returns a positive value, `toDate` is after the `fromDate`.

Examples

Example 3-9 Difference between two date/time items

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

Example 3-10 Difference between two hard-coded date/time 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
```

Example 3-11 Difference between 2 time items

```
var toDate = new Date( '01-Jan-0001 ' + ruleTimeItem.getHour() + ':' +
ruleTimeItem.getMinute() + ':' + ruleTimeItem.getSecond() );
var fromDate = new Date( '01-Jan-0001 ' + ruleTimeItem2.getHour() + ':' +
ruleTimeItem2.getMinute() + ':' + ruleTimeItem2.getSecond() );
return timeDiffInHours(toDate, fromDate);
```

Example 3-12 Difference between two partial date items

```
var toDate = new Date( ruleTimeItem.getYear() + '-' +
ruleTimeItem.getMonth() + '-' + ruleTimeItem.getDay() + ' ' +
ruleTimeItem.getHour() + ':' + ruleTimeItem.getMinute() + ':' +
ruleTimeItem.getSecond() );
var fromDate = new Date( ruleTimeItem.getYear() + '-' +
ruleTimeItem.getMonth() + '-' + ruleTimeItem.getDay() + ' ' +
ruleTimeItem.getHour() + ':' + ruleTimeItem.getMinute() + ':' +
ruleTimeItem.getSecond() );
return timeDiffInHours(toDate, fromDate);
```

timeDiffInMinutes( )

Calculate the time difference between two date or date/time values in minutes.

The `timeDiffInMinutes( )` helper function is invoked with starting and ending dates passed in as parameters. The function returns a negative or positive number value indicating the difference between the two dates in minutes.

> **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.

When using date type variables with no time elements, function considers time as '00:00:00'.

Syntax

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

Parameters

Parameter	Required/Optional	Description
`toDate`	Required	Ending date value.
`fromDate`	Required	Starting date value.

Return value

Number that represents the difference between the passed-in dates in minutes. This number can be positive or negative.

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

Examples

Example 3-13 Difference between two date/time items

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

Example 3-14 Difference between two hard-coded date/time 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
```

Example 3-15 Difference between 2 time items

```
var toDate = new Date( '01-Jan-0001 ' + ruleTimeItem.getHour() + ':' +
ruleTimeItem.getMinute() + ':' + ruleTimeItem.getSecond() );
var fromDate = new Date( '01-Jan-0001 ' + ruleTimeItem2.getHour() + ':' +
ruleTimeItem2.getMinute() + ':' + ruleTimeItem2.getSecond() );
return timeDiffInMinutes(toDate, fromDate);
```

Example 3-16 Difference between two partial date items

```
var toDate = new Date( ruleTimeItem.getYear() + '-' +
ruleTimeItem.getMonth() + '-' + ruleTimeItem.getDay() + ' ' +
ruleTimeItem.getHour() + ':' + ruleTimeItem.getMinute() + ':' +
ruleTimeItem.getSecond() );
var fromDate = new Date( ruleTimeItem.getYear() + '-' +
```

```
ruleTimeItem.getMonth() + '-' + ruleTimeItem.getDay() + ' ' +
ruleTimeItem.getHour() + ':' + ruleTimeItem.getMinute() + ':' +
ruleTimeItem.getSecond() );
return timeDiffInMinutes(toDate, fromDate);
```

timeDiffInSeconds( )

Calculate the time difference between two date or date/time values in seconds.

The `timeDiffInSeconds( )` helper function is invoked with starting and ending dates passed in as parameters. The function returns a negative or positive number value indicating the difference between the two dates in seconds.

> **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.

When using date type variables with no time elements, function considers time as '00:00:00'.

Syntax

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

Parameters

Parameter	Required/Optional	Description
`toDate`	Required	Ending date value.
`fromDate`	Required	Starting date value.

Return value

Number that represents the difference between the passed-in dates in seconds. This number can be positive or negative.

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

Examples

Example 3-17 Difference between two date/time items

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

Example 3-18 Difference between two hard-coded date/time 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
```

Example 3-19 Difference between two time items

```
var date1 = new Date( '01-Jan-0001 ' + ruleTimeItem.getHour() + ':' +
ruleTimeItem.getMinute() + ':' + ruleTimeItem.getSecond() );
var date2 = new Date( '01-Jan-0001 ' + ruleTimeItem2.getHour() + ':' +
ruleTimeItem2.getMinute() + ':' + ruleTimeItem2.getSecond() );
return areDateTimesEqual(date1, date2);
```

Example 3-20 Compare two partial date items

```
var date1 = new Date( ruleTimeItem.getYear() + '-' +  ruleTimeItem.getMonth()
+ '-' + ruleTimeItem.getDay() + ' ' + ruleTimeItem.getHour() + ':' +
ruleTimeItem.getMinute() + ':' + ruleTimeItem.getSecond() );
var date2 = new Date( ruleTimeItem.getYear() + '-' +  ruleTimeItem.getMonth()
+ '-' + ruleTimeItem.getDay() + ' ' + ruleTimeItem.getHour() + ':' +
ruleTimeItem.getMinute() + ':' + ruleTimeItem.getSecond() );
return areDateTimesEqual(date1, date2);
```

areDatesEqual( )

Compare two dates to determine if they are equivalent.

This function is used to compare variables of type date that do not contain time elements and do not include partial dates.. When using a date/time type parameter, function considers only date part and ignores time elements.

> **Tip:**
>
> - You can use the areDateTimesEqual( ) helper function to compare two date and time items.
> - If the date question contains partial date elements then use the getDatesCompareResult( ) helper function.

Syntax

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

Parameters

Parameter	Required/Optional	Description
date1	Required	First date value to compare.

Parameter	Required/Optional	Description
date2	Required	Second date value to compare.

Return value

Boolean (true or false) value:

- **True** if dates are equal.
- **False** if dates are different.

Examples

Example 3-21 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 3-22 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 given range.

This function is used to compare variables of type date that do not contain time elements and do not include partial dates.. When using a date/time type parameter, function considers only date part and ignores time elements.

> **Tip:**
>
> - You can use the isTimeInRange( ) helper function to compare two date and time items.
> - If the date question contains partial date elements then use the getDatesCompareResult( ) helper function.

Syntax

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

Parameters

Parameter	Required/Optional	Description
`dateToCheck`	Required	Date value to evaluate if it is within range.
`dateFrom`	Required	Starting date value for the date range to evaluate.
`dateTo`	Required	Ending date value for the date range to evaluate.

`inclusive`	Required	Determines whether the range limits are included or not for the in-range evaluation. Can take any of the following options:
		• **`'both'`**: include both date limits, `dateTo` and `dateFrom`, 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 date limits, `dateTo` nor `dateFrom`, in the range check.
		`dateFrom < dateToCheck && dateToCheck < dateTo`

Return value

Boolean (true or false) value:

- **True** if date is within range.
- **False** if date is out of range.

Examples

Example 3-23 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 3-24 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 or date/time values to determine if they are equivalent.

When using date type variables with no time elements, function considers time as '00:00:00'.

Syntax

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

Parameters

Parameter	Required/Optional	Description
date1	Required	First date/time value to compare.
date2	Required	Second date/time value to compare.

Return value

Boolean (true or false) value:

- **True** if date/times are equal.
- **False** if date/times are different.

Examples

Example 3-25 Compare two date/time 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 3-26 Compare two hard-coded date/time 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.
```

Example 3-27 Compare two time items

```
var date1 = new Date( '01-Jan-0001 ' + ruleTimeItem.getHour() + ':' +
ruleTimeItem.getMinute() + ':' + ruleTimeItem.getSecond() );
var date2 = new Date( '01-Jan-0001 ' + ruleTimeItem2.getHour() + ':' +
ruleTimeItem2.getMinute() + ':' + ruleTimeItem2.getSecond() );
return areDateTimesEqual(date1, date2);
```

Example 3-28 Compare two partial date items

```
var date1 = new Date( ruleTimeItem.getYear() + '-' +  ruleTimeItem.getMonth()
+ '-' + ruleTimeItem.getDay() + ' ' + ruleTimeItem.getHour() + ':' +
ruleTimeItem.getMinute() + ':' + ruleTimeItem.getSecond() );
var date2 = new Date( ruleTimeItem.getYear() + '-' +  ruleTimeItem.getMonth()
+ '-' + ruleTimeItem.getDay() + ' ' + ruleTimeItem.getHour() + ':' +
ruleTimeItem.getMinute() + ':' + ruleTimeItem.getSecond() );
return areDateTimesEqual(date1, date2);
```

isTimeInRange( )

Verify if a date or date/time value falls within a given range.

When using date type variables with no time elements, function considers time as '00:00:00'.

Syntax

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

Parameters

Parameter	Required/Optional	Description
`dateToCheck`	Required	Date/time value to evaluate if it is within range.
`dateFrom`	Required	Starting date/time value for the date range to evaluate.
`dateTo`	Required	Ending date/time value for the date range to evaluate.

Parameter	Required/Optional	Description

`inclusive`	Required	Determines whether the range limits are included or not for the in-range evaluation. Can take any of the following options:
		• **`'both'`**: include both date limits, `dateTo` and `dateFrom`, 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 date limits, `dateTo` nor `dateFrom`, in the range check.
		`dateFrom < dateToCheck && dateToCheck < dateTo`

Return value

Boolean (true or false) value:

- **True** if date is within range.
- **False** if date is out of range.

Examples

Example 3-29 Check a date/time 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 3-30 Compare three hard-coded date/time 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)
```

Example 3-31 Compare two time items

```
var dateToCheck= new Date( '01-01-001 ' + ruleTimeItem.getHour() + ':' +
ruleTimeItem.getMinute() + ':' + ruleTimeItem.getSecond() );
return isTimeInRange(dateToCheck, dateFrom, dateTo, "both");
```

addDays( )

Add a specific number of days to a date value.

> **Tip:**
>
> This function is useful when you need to evaluate if a date is exceeded by a specific amount of time.

This function is used with variables of type date that do not contain time elements. When using a date/time type parameter, function considers only date part and ignores time elements.

> **Tip:**
>
> You can use the addTimeInHours( ) or addTimeInMinutes( ) helper functions to work with date and time elements.

Syntax

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

Parameters

Parameter	Required/Optional	Description
`startTime`	Required	Date or date/time value to add days to.
`numberOfDays`	Required	Number of days to be added to the passed in date.

Return value

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

Examples

Example 3-32 date1 cannot be greater than 7 days 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 or date/time value.

> **Tip:**
>
> This function is useful when you need to evaluate if a date is exceeded by a specific amount of time.

Syntax

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

Parameters

Parameter	Required/Optional	Description
`startTime`	Required	Date or date/time value to add hours to.
`numberOfHours`	Required	Number of hours to be added to the passed in date.

Return value

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

Examples

Example 3-33 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
```

Example 3-34 Add hours to time items

```
var dateTime1= new Date( '01-Jan-0001 ' + ruleTimeItem.getHour() + ':' +
ruleTimeItem.getMinute() + ':' + ruleTimeItem.getSecond() );
return addTimeInHours(dateTime1, 30);
```

Example 3-35 Add hours to partial date items

```
var dateTime1= new Date( ruleTimeItem.getYear() + '-' +
ruleTimeItem.getMonth() + '-' + ruleTimeItem.getDay() + ' ' +
ruleTimeItem.getHour() + ':' + ruleTimeItem.getMinute() + ':' +
ruleTimeItem.getSecond() );
return addTimeInHours(dateTime1, 30);
```

addTimeInMinutes( )

Add a specific number of minutes to a date or date/time value.

> **Tip:**
>
> This function is useful when you need to evaluate if a date is exceeded by a specific amount of time.

Syntax

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

Parameters

Parameter	Required/Optional	Description
`startTime`	Required	Date or date/time value to add minutes to.
`numberOfMinutes`	Required	Number of minutes to be added to the passed in date.

Return value

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

Examples

Example 3-36 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
```

Example 3-37 Add minutes to Time Items

```
var dateTime1= new Date( '01-Jan-0001 ' + ruleTimeItem.getHour() + ':' +
ruleTimeItem.getMinute() + ':' + ruleTimeItem.getSecond() );
return addTimeInMinutes(dateTime1, 30);
```

Example 3-38 Add minutes to partial date items

```
var dateTime1= new Date( ruleTimeItem.getYear() + '-' +
ruleTimeItem.getMonth() + '-' + ruleTimeItem.getDay() + ' ' +
ruleTimeItem.getHour() + ':' + ruleTimeItem.getMinute() + ':' +
ruleTimeItem.getSecond() );
return addTimeInMinutes(dateTime1, 30);
```

getDateDMYFormat( )

Return a date or datetime in DD-Mon-YYYY format, including time elements if applicable. This function supports partial dates.

Syntax

```
getDateDMYFormat(vDate, [timeFormat])
```

Parameters

Parameter	Required/Optional	Description
`vDate`	Required	Rule variable of date or date/time type to be retrieved in DMY format.

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

Parameter	Required/Optional	Description
`timeFormat`	Optional	Specifies the output format for the time elements present. This can be one of the following values (case-sensitive), all in a 24-hour format: • `"HH:mm:ss"` • `"HH:mm"` • `"HH"` If `timeFormat` is not supplied, the return value will include all available time elements, suppressing any ending zero values. For example, if `vDate` contains a full time including hours, minutes and seconds the output will have a "HH:mm:ss" time format, and if it only contains hours then output will have a "HH" format. However, if time equals `"07:45:00"`, output will omit the seconds and return `"07:45"`, but would not remove the minutes in `"07:00:45"` as the ending value present is "45" for the seconds time element.

> **Note:**
>
> Previously, this function used to take an optional boolean value as the second parameter `isPartial`, to specify whether the function is expecting a partial date or not. This function still accepts **`true`** and **`false`** as the second parameter, instead of the `timeFormat` parameter, to continue supporting existing rules.

Return value

Date in DD-Mon-YYYY, for example **UNK-Jan-2025** or **01-Feb-2021 23:45:00**.

Examples

Example 3-39 Return a partial date in DD-Mon-YYYY format

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


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

getDatesCompareResult( )

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

As far as this function may contain partial date components, dates are compared up to the first defined part for both dates (second/minute/hour/day/month/year). For example, if the following two dates are compared:

- `01-Jun-2011 11:12:14`
- `02-Jan-2011 17:UNK:UNK`

The first defined part is *hour*, so dates will be compared as:

- `01-Jun-2011 11`
- `02-Jan-2011 17`

> **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. The comparison will always be in the format *<date1> <operation> <date2>*.

Syntax

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

Parameters

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

Parameter	Required/Optional	Description
`date1`	Required	First rule variable of date or date/time type to be compared. Supports Date, Datetime and Time type variables, either full or with partial components.
`isPartial1`	Required	Boolean value (true or false) to indicate if `date1` is partial or not.
`date2`	Required	Second rule variable of date or date/time type to be compared. Supports Date, Datetime and Time type variables, either full or with partial components.
`isPartial2`	Required	Boolean value (true or false) to indicate if `date2` is partial or not.
`operation`	Required	The operation you want to use to compare date1 and date2. For example: • ">" • ">=" • "<" • "<=" • "===" • "!=="

Return value

Boolean (true or false) value as a result of the given comparison operation.

Usage tips

- Since this function compares values, your rule expression may need to include a check to ensure the variables being passed are not null.

Examples

Example 3-40 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
```

Example 3-41 Compare time part of time and datetime variables using time components only

```
//compare time part of time (time1) and datetime (datetime1) components
var cdate1 = new C1Date (null, null, null, null, time1.getHour(),
time1.getMinute(), time1.getSecond());
var cdate2 = new C1Date (null, null, null, null, datetime1.getHour(),
datetime1.getMinute(), datetime1.getSecond());
return getDatesCompareResult(cdate1, true, cdate2, true, '===');
```

partialDateDiff( )

Find the difference between two dates.

As far as this function may contain partial date components, dates are compared up to the first defined part for both dates (second/minute/hour/day/month/year). For example, if the following two dates are compared:

- `01-Jun-2011 11:12:14`
- `02-Jan-2011 17:UNK:UNK`

The first defined part is *hour*, so dates will be compared as:

- `01-Jun-2011 11`
- `02-Jan-2011 17`

Syntax

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

Parameters

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

Parameter	Required/Optional	Description
`date1`	Required	First rule variable of date or date/time type to be compared. Supports Date, Datetime and Time type variables, either full or with partial components.

Parameter	Required/Optional	Description
`isPartial1`	Required	Boolean value (true or false) to indicate if `date1` is partial or not.
`date2`	Required	Second rule variable of date or date/time type to be compared. Supports Date, Datetime and Time type variables, either full or with partial components.
`isPartial2`	Required	Boolean value (true or false) to indicate if `date2` is partial or not.
`Datepart`	Required	String that specifies the part of the variable to compare. May contain one of the following values: • `'Day'` • `'Year'` • `'Hour'` • `'Minute'` • `'Second'`

Note:

When datepart is `'Day'` or `'Year'`, time elements are not considered in difference calculation

Return value

Number that represents the difference between the two dates in units of the given date part selected. For example if you added `'Day'` as the datepart to be compared, the difference is in days.

Differences are returned using the closest integer less than or equal to the exact value. For example, if the difference between 2 dates is 1.5 hours, then 1 hour is returned as the difference in hours.

Examples

Example 3-42 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 across repeating form instances for a given item. The data is identified by a form ID which has duplicate item values for the search keys provided. The rule target should be on the corresponding repeating section item.
- FindDuplicateRepeatingFormWithinRange( )
  Detect duplicate data across repeating form instances for a given item and within the same date range. The data is identified by a form ID which has duplicate item values for the search keys provided. The rule target should be on the corresponding repeating section item.
- FindMinInRepeatingForms( )
  Find the minimum value of a given number type 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 number 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.

- getPrevRepeatValue()
  Fetch a value from the previous non-deleted row within the same instance, where the question of interest is entered. This function is available for repeating forms and repeating sections of two-section forms.
- getRFValues( )
  Retrieves the current values for specified items on repeating form instances.

FindDuplicateRepeatingForm( )

Detect duplicate data across repeating form instances for a given item. The data is identified by a form ID which has duplicate item values for the search keys provided. The rule target should be on the corresponding repeating section item.

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

- 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.

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

Syntax

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

Parameters

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`variable(s)`	Required	Item variable to check, passed in using single quotes.

Return value

Boolean (true or false) value:

- **True** if duplicate values are found.
- **False** if duplicate values are not found.

Examples

Example 3-43 Check to see if any 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 across repeating form instances for a given item and within the same date range. The data is identified by a form ID which has duplicate item values for the search keys provided. The rule target should be on the corresponding repeating section item.

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

- 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.

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

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`startDateVariable`	Required	Date item on a repeating form, used as the start date for the date range to evaluate.

Parameter	Required/Optional	Description
`endDateVariable`	Required	Date item on a repeating form, used as the end date for the date range to evaluate.
`variable(s)`	Optional	Item variable to check, passed in using single quotes.

Return value

Boolean (true or false) value:

- **True** if duplicate values are found.
- **False** if duplicate values are not found.

Examples

Example 3-44 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 number type item in all instances of a repeating form. The data is retrieved by form ID and works only for numeric fields.

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

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

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`variable`	Required	Item variable to search, passed in using single quotes.

Return value

Numeric value that constitutes the minimum value across all instances or `"0"` if no minimum can be found.

Examples

Example 3-45 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 number item in all instances of a repeating form. The data is retrieved by form ID and works only for numeric fields.

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

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

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`variable`	Required	Item variable to search, passed in using single quotes.

Return value

Numeric value that constitutes the maximum value across all instances or `"0"` if no maximum can be found.

Examples

Example 3-46 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.

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

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`variable`	Required	Item variable to search, passed in using single quotes.

Parameter	Required/Optional	Description
`DateMask`	Optional	Flag to specify how partial dates should be handled: • If **`null`**, ignores partial dates when doing comparisons. • If a **value** is provided (in string format), replaces the UNK component in the partial date with the specified value. For example: entering `'10-Apr'` substitutes with `'10'` every `UNK` value of Day (DD) component and with `'Apr'` every `UNK` value of Month (MMM) component. ***Note:*** *Use mask only for date elements and do not use it for time elements. Any missing value in the time part is considered as 00.*

Return value

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

Usage tips

- 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.

- To convert return value to JavaScript date object, extra formatting should be done. For example:

```
vExample = '10-Jul-2022 10:UNK:UNK'
new Date(vExample.replace(/UNK/g, "00"))
```

Examples

Example 3-47 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.

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

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`variable`	Required	Item variable to search, passed in using single quotes.
`DateMask`	Optional	Flag to specify how partial dates should be handled: • If **`null`**, ignores partial dates when doing comparisons. • If a **value** is provided (in string format), replaces the UNK component in the partial date with the specified value. For example: entering `'10-Apr'` substitutes with `'10'` every `UNK` value of Day (DD) component and with `'Apr'` every `UNK` value of Month (MMM) component. ***Note:*** *Use mask only for date elements and do not use it for time elements. Any missing value in the time part is considered as 00.*

Return value

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

Usage tips

- 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.

- To convert return value to JavaScript date object, extra formatting should be done. For example:

```
vExample = '10-Jul-2022 10:UNK:UNK'
new Date(vExample.replace(/UNK/g, "00"))
```

Examples

Example 3-48

```
// 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.

- 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. You can use partial dates in the following formats:
  - **<dd-mmm-yyyy hh:mm>**

- **<dd-mmm-yyyy hh>**
- **<dd-mmm-yyyy>**
- **<mmm-yyyy>**
- **<yyyy>**

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

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`variable(s)`	Required	Item variable to search, passed in using single quotes.
`value(s)`	Required	Value for the given variable to search. These values must be hard-coded and cannot be rule variables: • Dates must be provided inside the string **`'Date(dd-mmm-yyyy hh:mm:ss)'`**. • You can use partial dates in the following formats: – **<dd-mmm-yyyy hh:mm>** – **<dd-mmm-yyyy hh>** – **<dd-mmm-yyyy>** – **<mmm-yyyy>** – **<yyyy>** • Times must be provided inside the string **`'Time(hh:mm:ss)'`**. • You can use partial times in the following formats: – **<hh:mm>** – **<hh>**

Return value

Number (**>0**) that represents the index of the form instance where the matching value was found.

- If multiple instances are found, only the first index is returned.
- Returns **-1** if no matches are found.

> **Note:**
>
> In dates, **UNK** values are considered to match any other value. For example: `'Date(01-Feb-2022)'` and `'Date(20-Feb-2022)'` are both considered as a match of an entry with `UNK-Feb-2022` date value.

Examples

Example 3-49 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.

- 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.
- This function also considers entries where the dates are **null**.
  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.
  - If the end date is `null` then it is assumed to be 01-Dec-3099.
- If a variable is designed to hold a partial date then provide the value for that parameter in the same partial date format. You can use partial dates in the following formats:
  - **<dd-mmm-yyyy hh:mm>**
  - **<dd-mmm-yyyy hh>**
  - **<dd-mmm-yyyy>**
  - **<mmm-yyyy>**
  - **<yyyy>**

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

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`startDateVariable`	Required	Date item on a repeating form, used as the start date for the date range to evaluate.
`startDateValue`	Required	Date value for the start date of the date range. This must be provided in a string format, must be be hard-coded and cannot be rule variables: • Dates must be provided inside the string **`'Date(dd-mmm-yyyy hh:mm:ss)'`**. • You can use partial dates in the following formats: – **<dd-mmm-yyyy hh:mm>** – **<dd-mmm-yyyy hh>** – **<dd-mmm-yyyy>** – **<mmm-yyyy>** – **<yyyy>** • Times must be provided inside the string **`'Time(hh:mm:ss)'`**. • You can use partial times in the following formats: – **<hh:mm>** – **<hh>**
`endDateVariable`	Required	Date item on a repeating form, used as the end date for the date range to evaluate.
`endDateValue`	Required	Date value for the end date of the date range. Consider same requirements as in *startDateValue* paraemter.
`variable(s)`	Optional	Item variable to search, passed in using single quotes.
`value(s)`	Optional	Search value(s) for the given variables. These must be hard-coded and cannot be rule variables.

Return value

Number (**>0**) that represents the index of the form instance where the matching value was found.

- If multiple instances are found, only the first index is returned.
- Returns **-1** if no matches are found.

> **Note:**
>
> In dates, **UNK** values are considered to match any other value. For example: `'Date(01-Feb-2022)'` and `'Date(20-Feb-2022)'` are both considered as a match of an entry with `UNK-Feb-2022` date value.

Examples

Example 3-50 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"
```

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 (=, >, <, >=, <=).

- This function does not support choice questions such as drop-down, radio button, or check box.
- Values in the custom function should be JavaScript variables or direct values. Do not use operand variables directly in the custom function expression.
- 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.

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

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`DateMask`	Optional	Flag to specify how partial dates should be handled: • If **`null`**, ignores partial dates when doing comparisons. • If a **value** is provided (in string format), replaces the UNK component in the partial date with the specified value. For example: entering `'10-Apr'` substitutes with `'10'` every `UNK` value of Day (DD) component and with `'Apr'` every `UNK` value of Month (MMM) component. ***Note:*** *Use mask only for date elements and do not use it for time elements. Any missing value in the time part is considered as 00.*
`variable(s)`	Required	Item variable to search, passed in using single quotes.
`compareOperator`	Required	Operator to use for the comparison of the provided variable and value: • = • >= • < • >= • <=
`value`	Required	Value to compare variable with as a string. • Date values can be provided as a Javascript Date variable or as a string in the following formats: – `'Date(dd-mmm-yyyy)'` – `'Date(dd-mmm-yyyy hh:mm:ss)'` – `'dd-mmm-yyyy'` – `'dd-mmm-yyyy hh:mm:ss'` • Time values can be provided as `'Time(hh:mm:ss)'`

Return value

Number (**>0**) that represents the index of the form instance where the matching value was found.

- If multiple instances are found, only the first index is returned.
- Returns **-1** if no matches are found.

Usage tips

- 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.

Examples

Example 3-51 Raise a query if there is an instance of pulse greater than a given value

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

Example 3-52 Raise query if Date variable is on or after a given date value

```
// 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;
```

Example 3-53 Raise query if Datetime variable is on or after a given datetime value

```
// Raise a query if there's an instance where onDateTime is >= 10-Jun-2010
11:12:15
//(onDateTime is partial DD-MMM-YYYY UNK:UNK:UNK)
return (FindRFInstance('10-Jun', 'onDateTime' , '>=', '10-Jun-2010 11:12:15')
> 0)?false:true;

//or
return (FindRFInstance('10-Jun', 'onDateTime' , '>=', 'Date(10-Jun-2010
11:12:15)') > 0)?false:true;
```

Example 3-54 Raise query if Time variable is on or after a given time value

```
// Raise a query if there's an instance where onTime is >= 11:12:15
//(onTime is partial HH:UNK:UNK)
return (FindRFInstance(null, 'onTime' ,'>=', 'Time(11:12:15)') > 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.

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

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`variable`	Required	Item variable to search, passed in using single quotes.
`includeDeleted`	Optional	Indicates whether deleted records are included in the output: • **0** - Do not include deleted Repeating instances in the return array. This is the default for when no value is provided. • **1** - Include deleted Repeating instances in the array count.

Return value

An array of repeating form instance numbers.

Examples

Example 3-55 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.

Example 3-56 Raise a query if given form instance number does not exist, using variables

```
// Raise a query if current form instance number does not exist
var curInst = GetCurrentRFInstance();
var arrAE = ListRFInstances('onDate', 0);
return (arrAE.indexOf(curInst.intValue()) == -1)?false:true;
```

> **Note:**
>
> Any variables passed when using the JavaScript `indexOf()` method with `ListRFInstances()`, should be converted for integer using `intValue()` to ensure the search works correctly.

GetCurrentRFInstance( )

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

Syntax

```
GetCurrentRFInstance()
```

Parameters

None.

Return Value

Number that represents the repeating form instance number where the rule is being run.

Examples

Example 3-57 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.

- Can accept choice controls (radio controls, check box controls, and dropdowns) but can only be searched by label, not value.
- Only one option can be provided as search text for choice controls.
- Dates must be provided as a string using the following format: **`'Date(dd-mmm-yyyy hh:mm:ss)'`**.
- You can use partial dates in the following formats:

  - **<dd-mmm-yyyy hh:mm>**
  - **<dd-mmm-yyyy hh>**
  - **<dd-mmm-yyyy>**
  - **<mmm-yyyy>**
  - **<yyyy>**
- Times must be provided as a string using the following format: `'Time(hh:mm:ss)'`.
- You can use partial times in the following formats:
  - **<hh:mm>**
  - **<hh>**

Syntax

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

Parameters

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`variable`	Required	Rule variable to evaluate, passed in using single quotes.
`value`	Required	Value to search for.

Return value

Number that represents the count of repeating form instances that match the passed-in value for the given variable.

> **Note:**
>
> In dates, **`UNK`** values are considered to match any other value. For example: `'Date(01-Feb-2022)'` and `'Date(20-Feb-2022)'` are both considered as a match of an entry with `UNK-Feb-2022` date value.

Examples

Example 3-58 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;
```

getPrevRepeatValue()

Fetch a value from the previous non-deleted row within the same instance, where the question of interest is entered. This function is available for repeating forms and repeating sections of two-section forms.

> **Tip:**
>
> You can use this function to get the value form a previous row for either the same or a different question.

Syntax

```
getPrevRepeatValue('ruleVariable', [isNullConsidered])
```

Parameters

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`ruleVariable`	Required	Name of the rule variable to get its value, passed in using single quotes. The rule variable is defined for a specific visit, form and item.
`isNullConsidered`	Optional	Boolean parameter to indicate if null values should be included in the search. • `true` - previous value is returned even if it is null. This parameter is set to `true` by default. • `false` - only returns value if it is not null. When `false` is set for this parameter, system will keep looking backwards until either it finds the closest not-null value or the entirety of rows have been searched.

Return value

Returns the value for the specified item in the immediate previous row. Depending on the optional parameter configuration, whether null is considered or not, can return any value (including null) from the immediate previous row, or the closest previous not null value, if the immediate previous row is null for the given item.

If the variable is a choice control (checkbox, radio or drop-down), the return value is the string in JSON format:

```
("[{\"value\":\"3\",\"label\":\"TestLabel\"}]")
```

> **Tip:**
>
> This can be parsed using `JSON.parse(result)` or the helper function `parseChoice(result)`.

Examples

Example 3-59 Get value of a given variable in the previous row

```
var prevValue= getPrevRepeatValue('vValue', false); //returns the first
previous not null value
var prevValue= getPrevRepeatValue('vValue');  //returns the first previous
value
```

getRFValues( )

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

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

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

Syntax

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

Parameters

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`formInstance`	Required	The instance number of the form you're retrieving values from. This parameter can be a JavaScript variable or it can be a number.
`isNullConsidered`	Optional	Item variable names, passed in using single quotes, for the values to retrieve.

Return value

Returns a JSON object containing the variables (of the same name as passed in the second parameter) 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.

Examples

Example 3-60 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( )
  Detect duplicate data across two-section form instances for a given item either within the flat section or the repeating section. The data is identified by a form ID which has duplicate

item values for the search keys provided. The rule target should be on the corresponding repeating section item.

- findDuplicate2SFormWithinRange( )
  Detect duplicate data across two-section form instances for a given item within the same date range, either within the flat section or the repeating section. The data is identified by a form ID which has duplicate item values for the search keys or date ranges provided. The rule target should be on the corresponding repeating section item.
- 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( )

Detect duplicate data across two-section form instances for a given item either within the flat section or the repeating section. The data is identified by a form ID which has duplicate item

values for the search keys provided. The rule target should be on the corresponding repeating section item.

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

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

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`formInstance`	Optional	Indicates the form instance in which the search for a duplicate is to be performed. • 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 the given two-section form instance.
`variable(s)`	Required	Item variable to check, passed in using single quotes.

Return value

Boolean (true or false) value:

- **True** if duplicate values are found.
- **False** if duplicate values are not found.

Examples

Example 3-61 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 (findDuplicate2sForm(null,'itmLab', 'itmTestName')) {
  return false;
} else {
  return true;
}
```

Example 3-62 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

return findDuplicate2SForm(null, 'txt');

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

var arrAE = findDuplicate2SForm(2, "txt");
return (arrAE.length <= 1)?false:true;
```

findDuplicate2SFormWithinRange( )

Detect duplicate data across two-section form instances for a given item within the same date range, either within the flat section or the repeating section. The data is identified by a form ID which has duplicate item values for the search keys or date ranges provided. The rule target should be on the corresponding repeating section item.

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

- 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.

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','endDateVariab
le','variable1',...)
```

Parameters

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`formInstance`	Optional	Indicates the form instance in which the search is to be performed. • 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, used as the start date for the date range to evaluate.
`endDateVariable`	Required	Date item on a repeating form, used as the end date for the date range to evaluate.
`variable(s)`	Optional	Item variable to check, passed in using single quotes.

Return value

Boolean (true or false) value:

- **True** if duplicate values are found.
- **False** if duplicate values are not found.

Examples

Example 3-63 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','it
mSymptom') === 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.

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(formInstance,'variable')
```

Parameters

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`formInstance`	Optional	Indicates the form instance in which the search for the minimum value is to be performed. • If `formInstance` is `null` and variable is in flat section, the flat section of all instances is searched. • If `formInstance` is `null` and variable is in a table row, all table rows in all instances are searched. • If `formInstance` value is provided, the search is across the table rows of the specified instance only.
`variable(s)`	Required	Item variable to check, passed in using single quotes.

Return value

Numeric value that constitutes the minimum value across all instances or `"0"` if no minimum can be found.

Examples

Example 3-64 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(null,'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.

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(formInstance,'variable')
```

Parameters

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`formInstance`	Optional	Indicates the form instance in which the search for the maximum value is to be performed. • If `formInstance` is `null` and variable is in flat section, the flat section of all instances is searched. • If `formInstance` is `null` and variable is in a table row, all table rows in all instances are searched. • If `formInstance` value is provided, the search is across the table rows of the specified instance only.
`variable(s)`	Required	Item variable to check, passed in using single quotes.

Return value

Numeric value that constitutes the maximum value across all instances or `"0"` if no maximum can be found.

Examples

Example 3-65 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(null,'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.

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

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`variable`	Required	Item variable to search, passed in using single quotes.
`DateMask`	Optional	Flag to specify how partial dates should be handled: • If **`null`**, ignores partial dates when doing comparisons. • If a **value** is provided (in string format), replaces the UNK component in the partial date with the specified value. For example: entering `'10-Apr'` substitutes with `'10'` every `UNK` value of Day (DD) component and with `'Apr'` every `UNK` value of Month (MMM) component. ***Note:*** *Use mask only for date elements and do not use it for time elements. Any missing value in the time part is considered as 00.*

Return value

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

Usage tips

- 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.

- To convert return value to JavaScript date object, extra formatting should be done. For example:

```
vExample = '10-Jul-2022 10:UNK:UNK'
new Date(vExample.replace(/UNK/g, "00"))
```

Examples

Example 3-66 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.

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

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`variable`	Required	Item variable to search, passed in using single quotes.
`DateMask`	Optional	Flag to specify how partial dates should be handled: • If **`null`**, ignores partial dates when doing comparisons. • If a **value** is provided (in string format), replaces the UNK component in the partial date with the specified value. For example: entering `'10-Apr'` substitutes with `'10'` every `UNK` value of Day (DD) component and with `'Apr'` every `UNK` value of Month (MMM) component. ***Note:*** *Use mask only for date elements and do not use it for time elements. Any missing value in the time part is considered as 00.*

Return value

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

Usage tips

- 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.

- To convert return value to JavaScript date object, extra formatting should be done. For example:

```
vExample = '10-Jul-2022 10:UNK:UNK'
new Date(vExample.replace(/UNK/g, "00"))
```

Examples

Example 3-67 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.

- 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. You can use partial dates in the following formats:
  - **<dd-mmm-yyyy hh:mm>**
  - **<dd-mmm-yyyy hh>**
  - **<dd-mmm-yyyy>**
  - **<mmm-yyyy>**
  - **<yyyy>**

> **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

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`formInstance`	Optional	Indicates the form instance in which the search is to be performed. • If **null**, the search considers 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 considers an array of existing table rows on the specified instance of the two-section form.
`variable(s)`	Required	Item variable to check, passed in using single quotes.
`value(s)`	Required	Value for the given variable to search. These valuesmust be hard-coded and cannot be rule variables: • Dates must be provided inside the string **`'Date(dd-mmm-yyyy hh:mm:ss)'`**. • You can use partial dates in the following formats: – **<dd-mmm-yyyy hh:mm>** – **<dd-mmm-yyyy hh>** – **<dd-mmm-yyyy>** – **<mmm-yyyy>** – **<yyyy>** • Times must be provided inside the string **`'Time(hh:mm:ss)'`**. • You can use partial times in the following formats: – **<hh:mm>** – **<hh>**

Return value

Number (**>0**) that represents the index of the instance where the matching value was found.

- When searching across all instances for the two-section form (when `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.
- If multiple instances are found, only the first index is returned.
- Returns **-1** if no matches are found.

> **Note:**
>
> In dates, **`UNK`** values are considered to match any other value. For example: `'Date(01-Feb-2022)'` and `'Date(20-Feb-2022)'` are both considered as a match of an entry with `UNK-Feb-2022` date value.

Examples

Example 3-68 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.

- 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.
- This function also considers entries where the dates are **`null`**.
  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.

  - If the end date is `null` then it is assumed to be 01-Dec-3099.
- If a variable is designed to hold a partial date then provide the value for that parameter in the same partial date format. You can use partial dates in the following formats:
  - **<dd-mmm-yyyy hh:mm>**
  - **<dd-mmm-yyyy hh>**
  - **<dd-mmm-yyyy>**
  - **<mmm-yyyy>**
  - **<yyyy>**

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

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`startDateVariable`	Required	Date item on a repeating form, used as the start date for the date range to evaluate.
`startDateValue`	Required	Date value for the start date of the date range. This must be provided in a string format, must be be hard-coded and cannot be rule variables: • Dates must be provided inside the string **`'Date(dd-mmm-yyyy hh:mm:ss)'`**. • You can use partial dates in the following formats: – **<dd-mmm-yyyy hh:mm>** – **<dd-mmm-yyyy hh>** – **<dd-mmm-yyyy>** – **<mmm-yyyy>** – **<yyyy>** • Times must be provided inside the string **`'Time(hh:mm:ss)'`**. • You can use partial times in the following formats: – **<hh:mm>** – **<hh>**
`endDateVariable`	Required	Date item on a repeating form, used as the end date for the date range to evaluate.

Parameter	Required/Optional	Description
`endDateValue`	Required	Date value for the end date of the date range. Consider same requirements as in *startDateValue* paraemter.
`variable(s)`	Optional	Item variable to search, passed in using single quotes.
`value(s)`	Optional	Search value(s) for the given variables. These must be hard-coded and cannot be rule variables.

Return value

Number (**>0**) that represents the index of the instance where the matching value was found.

- If multiple instances are found, only the first index is returned.
- Returns **-1** if no matches are found.

> **Note:**
>
> In dates, **`UNK`** values are considered to match any other value. For example: `'Date(01-Feb-2022)'` and `'Date(20-Feb-2022)'` are both considered as a match of an entry with `UNK-Feb-2022` date value.

Examples

Example 3-69 Raise a query if any instances exist where a Date variable is between a given range and there is any other given condition that matches

```
// Given 5 repeating form instances with items "onsetStart", "onsetEnd" and
"itmSymptom"
//Raises query if symptom = "headache" AND onSet date is between Jan 1 2020
and March 1 2020
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"



//if rule variable (datetimeVar) is datetime componentif
(findMatching2SFormWithinRange(2, 'onsetStart', 'Date(01-JAN-2020)',
'onsetEnd', 'Date(01-MAR-2020)', 'datetimeVar', 'Date(10-Jun-2010 11:12:15)')
> 0) {
return false;
} else {
return true;
}
```

```
//if rule variable (timeVar) is time componentif
(findMatching2SFormWithinRange(2, 'onsetStart', 'Date(01-JAN-2020)',
'onsetEnd', 'Date(01-MAR-2020)', 'timeVar', 'Time(11:12:15)') > 0) {
return false;
} else {
return true;
}
```

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 (=, >, <, >=, <=).

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(formInstance, DateMask, 'variable1', 'compareOperator1',
value1, 'variable2', ...)
```

Parameters

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`formInstance`	Optional	Indicates the form instance in which the search is to be performed. • If `formInstance` is `null` and variable is in flat section, the flat portion across all instances is searched. • If `formInstance` is `null` and variable is in a table row, the search is across all table rows in all instances. • If `formInstance` value is provided, the search is across the table rows of the specified instance.

Parameter	Required/Optional	Description
`DateMask`	Optional	Flag to specify how partial dates should be handled: • If **`null`**, ignores partial dates when doing comparisons. • If a **value** is provided (in string format), replaces the UNK component in the partial date with the specified value. For example: entering `'10-Apr'` substitutes with `'10'` every `UNK` value of Day (DD) component and with `'Apr'` every `UNK` value of Month (MMM) component. ***Note:*** *Use mask only for date elements and do not use it for time elements. Any missing value in the time part is considered as 00.*
`variable(s)`	Required	Item variable to search, passed in using single quotes.
`compareOperator`	Required	Operator to use for the comparison of the provided variable and value: • = • >= • < • >= • <=
`value`	Required	Value to compare variable with as a string. • Date values can be provided as a Javascript Date variable or as a string in the following formats: – `'Date(dd-mmm-yyyy)'` – `'Date(dd-mmm-yyyy hh:mm:ss)'` – `'dd-mmm-yyyy'` – `'dd-mmm-yyyy hh:mm:ss'` • Time values can be provided as `'Time(hh:mm:ss)'`

Return value

Number (**>0**) that represents the index of the instance where the matching value was found.

- If multiple instances are found, only the first index is returned.
- Returns **-1** if no matches are found.

Usage tips

- Values in the custom function should be JavaScript variables or direct values. Do not use operand variables directly in the custom function expression.
- 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.
- 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.

Examples

Example 3-70 Raise a query if there is an instance of pulse greater than a given value

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

Example 3-71 Raise a query if there is an instance of a date variable on or after a given date value

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

Example 3-72 Raise a query if there is an instance of a datetime variable on or after a given datetime value

```
return (find2SFormInstance(2, '10-Jun', onDateTime , '>=', '10-Jun-2010
11:12:15') > 0)?false:true;
//or
return (find2SFormInstance(2, '10-Jun', 'onDateTime' , '>=', 'Date(10-
Jun-2010 11:12:15)') > 0)?false:true;
```

Example 3-73 Raise a query if there is an instance of a Time variable on or after a given time value

```
return (find2SFormInstance(2, null, 'onTime' ,'>=', 'Time(11:12:15)') > 0)?
false:true;
```

list2SInstances( )

List all instance numbers for a two-section form.

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', formInstance, includeDeleted)
```

Parameters

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`variable`	Required	Item variable to search, passed in using single quotes.
`formInstance`	Optional	Indicates the form instance in which the search is to be performed. • If **null**, the search considers an array of existing instances of two-section forms. Ideally this should be used with variables in the flat section of the form. • If a value is provided, the search considers an array of existing table rows on the specified instance of the two-section form.
`includeDeleted`	Optional	Indicates whether deleted records are included in the output: • **0** - Do not include deleted Repeating instances in the return array. This is the default for when no value is provided. • **1** - Include deleted Repeating instances in the array count.

Return value

An array of two-section form instance numbers.

Examples

Example 3-74 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.

Syntax

```
getCurrent2SFormInstance()
```

Parameters

None.

Return value

Number that indicates the two-section form instance number where the rule is being run.

Examples

Example 3-75 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.

Syntax

```
getCurrent2STableInstance()
```

Parameters

None.

Return value

Number that represents the table row where the rule is currently being run:

- A number greater than or equal to **1** refers to a table instance number.
- **-1** indicates it is not a two-section form.

Examples

Example 3-76 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.

- Can accept choice controls (radio controls, check box controls, and dropdowns) but can only be searched by label, not value.
- Only one option can be provided as search text for choice controls.

- Dates must be provided inside the string `'Date(dd-mmm-yyyy hh:mm:ss)'`.
- You can use partial dates in the following formats:
  - **<dd-mmm-yyyy hh:mm>**
  - **<dd-mmm-yyyy hh>**
  - **<dd-mmm-yyyy>**
  - **<mmm-yyyy>**
  - **<yyyy>**
- Times must be provided inside the string `'Time(hh:mm:ss)'`.
- You can use partial times in the following formats:
  - **<hh:mm>**
  - **<hh>**

Syntax

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

Parameters

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`formInstance`	Optional	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`	Required	Rule variable to evaluate, passed in using single quotes.
`value`	Required	Value to search for.

Return value

Number that represents the 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.

> **Note:**
>
> In dates, **`UNK`** values are considered to match any other value. For example: `'Date(01-Feb-2022)'` and `'Date(20-Feb-2022)'` are both considered as a match of an entry with `UNK-Feb-2022` date value.

Examples

Example 3-77 Raise a query if there is more than one instance where AE Outcome is "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',
'[{"value":"3","label":"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

> **Note:**
>
> It is allowed to reuse variables passed into this function elsewhere in the rule expression, however you must add the variable as a parameter using single quotes.

Parameter	Required/Optional	Description
`formInstance`	Required	The instance number of the form you're retrieving values from. This parameter can be a JavaScript variable or it can be a number.
`tableRowInstance`	Optional	Instance number of the table of the two-section form to retrieve values from. This parameter should be **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.
`isNullConsidered`	Optional	Item variable names, passed in using single quotes, for the 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.

Examples

Example 3-78 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( )
  Obtain the property value of a given property name for the current visit. A visit property may be its title, ID or event type.
- 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

Parameter	Required/Optional	Description
`operator`	Required	Operator to compare the current study version with a given value. Can be any of the following: • **'<'** • **'>'** • **'='** • **'<='** • **'>='**
`version`	Required	Version number to compare the current study version with.
`variable`	Optional	Variables that can be used in the rule expression given a **true** condition (if the comparison scenario is true).

Return value

Boolean (true or false) value, depending on the comparison result.

Examples

Example 3-79 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 3-80 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( )

Obtain the property value of a given property name for the current visit. A visit property may be its title, ID or event type.

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

For more information see Define rule variables.

Syntax

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

Parameters

Parameter	Required/Optional	Description
`propertyName`	Required	Name of the visit property you want to retrieve. May be any of the following (including quotes is required): • `'title'` • `'visitid'` • `'eventtype'`

Return value

Returns the current visit's given property.

Event type property values can be one of following based on visit design:

- ScreeningVisit
- ScheduleAbleVisit
- SubjectWithdrawalVisit
- SubjectCompletionVisit
- UnScheduleAbleVisit
- Event
- AdverseEvent

Examples

Example 3-81 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","SubjectCompleti
onVisit","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

Parameter	Required/Optional	Description
`argument`	Required	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. **Tip:** 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 value of the argument passed to the `logMsg()` helper function will be returned and displayed in the log window when debugging.

Usage tips

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

Examples

Example 3-82 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 3-83 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 3-84 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,"treatment
Arm":null},
{"visitName":"SCR","deleted":false,"tableRowInstance":null,"branchName":null,"
eventType":"ScreeningVisit","formRepeatNumber":2,
"value":"2022-03-09T00:02:00.000Z","cycleNumber":null,"empty":false,"treatment
Arm":null},
{"visitName":"SCR","deleted":false,"tableRowInstance":null,"branchName":null,"
eventType":"ScreeningVisit","formRepeatNumber":3,
"value":"2022-03-09T06:00:00.000Z","cycleNumber":null,"empty":false,"treatment
Arm":null}]
```

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).
- [Deprecated] - clearChoice( )
  Use this helper function to clear values in multiple choice type questions.
- getArrayFromChoice( )
  Convert the selected choice labels or codes from a multiple choice question (drop-down, radio button, check box) into an array.
- getStringFromChoice( )
  Convert selected choice labels or codes from a multiple-choice question (drop-down, radio button, check box) into a comma-separated string.

[Deprecated] - getArrayFromDropdown( )

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

> **Note:**
>
> As this function is deprecated, we recommend you use getArrayFromChoice( ) instead, which is used for the same purpose and supports all choice-type controls. However, rules that call `getArrayFromDropdown()` can still work when used with drop-downs.

Syntax

```
getArrayFromDropdowm(variable)
```

Parameters

Parameter	Required/Optional	Description
`variable`	Required	Rule variable, corresponding to a drop-down value, that you want to retrieve.

Return value

This function returns an array with the labels of the selected drop-down options. If no values are selected it returns an empty array.

Examples

Example 3-85 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:**
>
> As this function is deprecated, we recommend you use getStringFromChoice( ) instead, which is used for the same purpose and supports all choice-type controls. However, rules that call `getStringFromDropdown()` can still work when used with drop-downs.

Syntax

`getStringFromDropdown(variable)`

Parameters

Parameter	Required/Optional	Description
`variable`	Required	Rule variable, corresponding to a drop-down value, that you want to retrieve.

Return value

This function returns a comma-separated string with the labels of the selected drop-down options. If no values are selected it returns an empty string.

Examples

Example 3-86 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 [Deprecated] - clearChoice( ).

If you want to set the value instead of the label see setChoiceValue( ).

Syntax

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

Parameters

Parameter	Required/Optional	Description
`labelStr`	Required	String value of the label you want to set for the given choice type field.
`variable`	Required	Rule variable, corresponding to a choice type field, that you want to set.

Return value

This function returns a JSON object with the string array of the labels of the selected choice options. If no values are selected it returns an empty object.

Examples

Example 3-87 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 [Deprecated] - clearChoice( ).

If you want to set the label instead of the value see setChoiceLabel( ).

Syntax

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

Parameters

Parameter	Required/Optional	Description
`labelStr`	Required	String value you want to set for the given choice type field.
`variable`	Required	Rule variable, corresponding to a choice type field, that you want to set.

Return value

This function returns a JSON object with the string array of the labels of the selected choice options. If no values are selected it returns an empty object.

Examples

Example 3-88 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
```

[Deprecated] - clearChoice( )

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

> **Note:**
>
> As this function is deprecated, we recommend you use clearValue( ) instead, which is used for the same purpose and also supports text and number questions. However, rules that call `clearChoice()` still work to clear values in multiple choice questions.

To properly clear a rule target, this function needs to be used as the rule's return statement. See examples below.

For more information about clearing data, see Understanding data clearing.

Syntax

```
clearChoice()
```

Parameters

None.

Return value

This function is used as the return value of a calculation rule. It returns an indicator to the rule process to start data clear in the back end.

Examples

Example 3-89 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
```

Related Topics

- Create a rule for a calculated value
  You can create a rule that enables the system to automatically calculate a value in a form. This allows you to automatically calculate certain values based on manually entered form data. reducing calculation errors and simplifying forms.

getArrayFromChoice( )

Convert the selected choice labels or codes from a multiple choice question (drop-down, radio button, check box) into an array.

Syntax

```
getArrayFromChoice(variable, [option])
```

Parameters

Parameter	Required/Optional	Description
`variable`	Required	Rule variable, corresponding to a choice type field, that you want to retrieve.
`option`	Optional	Defines which element of a choice control value to return (including quotes is required): • `"label"`: returns the selected choice control label. This is the default option if no option is provided. • `"code"`: returns the selected choice control code if the question choice comes form a codelist.

Return value

This function returns an array with the labels of the selected choice control options. If no values are selected it returns an empty array.

Examples

Example 3-90 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"

// Return the first selected code from choice item dd2:
return getArrayFromChoice(dd2, "code")[0];
// returns C1
```

getStringFromChoice( )

Convert selected choice labels or codes from a multiple-choice question (drop-down, radio button, check box) into a comma-separated string.

Syntax

```
getStringFromChoice(variable, [option])
```

Parameters

Parameter	Required/Optional	Description
`variable`	Required	Rule variable, corresponding to a choice type field, that you want to retrieve.

Parameter	Required/Optional	Description
`option`	Optional	Defines which element of a choice control value to return (including quotes is required): • `"label"`: returns the selected choice control label. This is the default option if no option is provided. • `"code"`: returns the selected choice control code if the question choice comes form a codelist.

Return value

This function returns a comma-separated string with the labels of the selected choice control options. If no values are selected it returns an empty string.

Examples

Example 3-91 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"


// return a code from choice:
return getStringFromChoice(dd2, "code");
// returns C1
```

Example 3-92 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

- String value that constitues the current branch's ID.
- Empty string if the visit is not a branch visit.

Usage tips

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

Examples

Example 3-93 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

Parameter	Required/Optional	Description
branchShortName	Required	The short ID for the branch you are querying for.

Return value

Boolean value:

- **True** if the branch with the given short name contains any visits for the subject where data has been entered.
- **False** if no visits are initialized in that branch for that subject.

Usage tips

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

Examples

Example 3-94 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()
```

Parameters

None.

Return value

- String value that indicates the current treatment arm short name for the subject.

- Empty string, if the subject hasn't been randomized and the treatment arm for them does not exist.

Examples

Example 3-95 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

> **Note:**
>
> If optional parameters are missing, the event instance number and form instance number are taken from rule target context.

Parameter	Required/Optional	Description
`ruleVariable`	Required	Variable to be retrieved. Must be a rule variable (added globally in the rule editor).
`eventInstanceNumber`	Optional	Event instance number/cycle instance number. To reference non-repeating visits, pass an empty string.

Parameter	Required/Optional	Description
`formInstanceNumber`	Optional	Form instance number. To reference a non-repeating form, pass an empty string.
`repeatFormNumber`	Optional	Repeat form number to reference items on a two section form.

Return value

The question value in the corresponding data type, or null if there is no value.

> **Note:**
>
> The value is returned even for deleted instances.

Question type	Returned data type details.
Date	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. For additional information see Handle partial dates in custom rules. **Note:** When using date items, a null check must be included. See the example for more information.
Choice control: • Checkbox • Radio button • Dropdown	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))`

Examples

Example 3-96 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

Syntax

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

Parameters

Parameter	Required/Optional	Description
`variable(s)`	Required	Item variable to retrieve.

Return value

The rule returns an array of data element arrays that contain data collection information about all existing instances for each variable. Includes visit or branch short name.

Examples

Example 3-97 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

Number that indicates the cycle intsnace number:

- A positive number refers to the current cycle instance number.
- **-1** if the target form is not on a cycle visit.

Examples

Example 3-98 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

Parameter	Required/Optional	Description
`branchShortName`	Required	The short ID for the branch you are querying for.

Return value

Number that indicates the branch cycle:

- A positive number refers to the current cycle number.

- **-1** if the subject is not on the given branch.

Usage tips

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

Examples

Example 3-99 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

Parameter	Required/Optional	Description
visitShortName	Required	The short ID for the visit you are querying for.

Return value

Numeric value:

- The number of cycles where the visit is in the Completed status.
- **-1** if the given visit is not a cycle visit.

Usage tips

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

Examples

Example 3-100 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**.
- getValues( )
  Use this helper function to fetch values for one or more variables across multiple visits, in an array format ordered by visits.
- clearValue( )
  Use this helper function to clear values in a target populated by a calculation rule, whether it is a text, number, or multiple choice question.

setQueryMessage( )

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

The message you set for the query can be a dynamically generated string. You can do this by using variables or functions that return strings. Refer to the examples section below.

The default query message provided on rule creation is used when no dynamic query message is set upon running the rule. If you use the `setQueryMessage()` function within your logic, the dynamic query message is not set if its value is null, undefined, an empty string, or contains a space only.

> **Note:**
>
> This function cannot be used for rules that send email notifications.

Syntax

```
setQueryMessage(strMessage)
```

Parameters

Parameter	Required/Optional	Description
strMessage	Required	A string containing the query message. This string can be dynamically generated within the rule expression.

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

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.

Examples

Example 3-101 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."
```

getValues( )

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

This is an aggregation function. The rule is run for each form instance in the case where the target is on a repeating form. For empty rows in repeating forms and two-section forms, only rows that were entered and then cleared will be included. Rows that never had data entered into it, won't be returned.

Syntax

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

Parameters

Parameter	Required/Optional	Description
`'var1', 'var2', ...`	At least one is required.	Rule variable names that define the visit, form, or item to be retrieved.

Return value

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

Also, during the call to this function, passed in rule variables (var1, var2, …) are redefined with the gotten value. As a result, each variable holds an array with attributes.

> **WARNING:**
>
> Rule variables will be redefined to `null` if there is no data elements related to this variable. For this reason you should always validate if the variable contains any array elements before you use it for processing data.

Each item in the array, for example `var1[0]`, contains the following information as attributes:

Attribute	Description
`var1[0].visitName`	The name of the visit.
`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.
`var1[0].branchName`	Name of branch.
`var1[0].eventType`	Type of visit: • Scheduled Visit • Unscheduled Visit • Event
`var1[0].formRepeatNumber`	The repeating form instance number in case form is a repeating form.

Attribute	Description

var1[0].value	Data element value, or **null** if values were cleared.
	• **Date elements**:
	– The returned variable value is a `Date` object in the case where the variable is the full date.
	– The returned variable value is a `C1Date` object in the case where a variable is a partial date.
	* For `C1Date` objects (partial dates), date values 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`
	`}`
	`]`
	
	• **Choice control elements:** 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( )`.

var1[0].cycleNumber	The cycle number of the visit in case it is in a cycle.
var1[0].empty	True if value was cleared or never entered in repeating form.
var1[0].cleared	True if repeating forms instance is cleared.
var1[0].treatmentArm	The treatment arm.

Usage tips

- If you are working with variables defined for *-Any Visit-* you must use this helper function to get the values and use them in your expression. You can't work with this type of variables

directly in your logic. Use the following format, where `'variable'` is your *Any Visit* type variable:

```
getValues('variable')
```

This type of variable definition allows you to work with data collected in forms that are not present in the same visit as the target form in which you are creating your rule. For more information see Define rule variables.

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

Examples

Example 3-102 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 3-103 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 3-104 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;
}
```

Example 3-105 Find the previous/next value with target and operands on the same form

Rule Variables:

- `brDateForFunc` - rule variable which references date.
- `brDate` - rule variable which references date (the same as the first one just with different name in case the same items on different visits should be compared).

```
var visitName = getCurrentVisitPropertyValue("visitid");
var currCycle = getCurrentCycle();var prevValue = null;
var prevValueFound = false;

function getPrevValue(item){
    if(prevValueFound) return;
    if(item.visitName==visitName && item.cycleNumber==currCycle){
        prevValueFound = true;        return;
    }
    if(item.eventType!='UnScheduleAbleVisit' && item.value!=null)
        prevValue = item.value;
}

var res = getValues('brDateForFunc');
if(res){
    brDateForFunc.forEach(getPrevValue);
         //in case the next visit should be found the array should be
reverted:
         //brDateForFunc.reverse().forEach(getPrevValue);

//here should go the necessary logic to compare brDate with previous value
which is now in variable prevValue
}
```

Example 3-106 Find the last or closest next value with target and operands on different forms

Rule variables:

- `aeDate`- rule variable which references date on AE form.
- `cycleDate1` - rule variable which references date on the first cycle visit.
- `cycleDate2` - rule variable which references date on the second cycle visit.
- `cycleCheck1` - rule variable which references some condition on the first cycle visit.
- `cycleCheck2` - rule variable which references some condition on the second cycle visit.
- `vsd` - visit start date with All Visits option

```
var lastStartedlVisit = null;
var lastVisitFound = false;
var cycleDate = null;
var cycleCheck = null;
var cycleNumber = null;

function getLastStartedlVisit(item){
    if(lastVisitFound || item.eventType=='UnScheduleAbleVisit') return;
```

```
      //condition may depend on where the point of comparison is, at some
situation it can be <if(item.value==null)> or something else
    if(item.eventType=='AdverseEvent'){
        lastVisitFound = true;
        return;
    }
if(item.value!=null) {
        lastStartedlVisit = item;
   }
}

function findItem(item, index){
    if(item.cycleNumber === cycleNumber){
        if(this.valueOf()=='cycleDate') cycleDate = item.value;
        if(this.valueOf()=='cycleCheck') cycleCheck = item.value;
    }
}

var res = getValues('vsd');
if(res){
    //the first visit before AE
    vsd.forEach(getLastStartedlVisit);

    //in case the next visit should be found the array should be reverted:
    //vsd.reverse().forEach(getLastStartedlVisit);

    cycleNumber = lastStartedlVisit.cycleNumber;
    res = getValues('cycleDate1');
    if( cycleDate1[cycleDate1.length-1].visitName ==
lastStartedlVisit.visitName){
       cycleDate1.forEach(findItem, 'cycleDate');
       res = getValues('cycleCheck1');
       cycleCheck1.forEach(findItem, 'cycleCheck');
   }
   else{
       res = getValues('cycleDate2');
       if( cycleDate2[cycleDate2.length-1].visitName ==
lastStartedlVisit.visitName){
             cycleDate2.forEach(findItem, 'cycleDate');
             res = getValues('cycleCheck2');
             cycleCheck2.forEach(findItem, 'cycleCheck');
       }
    }
}
//logMsg(JSON.stringify(aeDate))
//logMsg(JSON.stringify(cycleDate));
//logMsg(JSON.stringify(cycleCheck));
//logic to compare cycleDate cycleCheck and aeDate
```

Example 3-107 Compare date with next visit start date

Rule variables:

- `brDate` - rule variable which references date.

- `vsd` - rule variable which references visit start date with All Visits option.

```
var visitName = getCurrentVisitPropertyValue("visitid");
var currCycle = getCurrentCycle();
//logMsg(visitName);
//logMsg(currCycle);
//logMsg(brDate);
var nextValue = null;
var nextValueFound = false;

function getNextValue(item){
    if(nextValueFound) return;
    if(item.visitName==visitName && item.cycleNumber==currCycle){
       nextValueFound = true;
       return;
    }
    if(item.eventType!='UnScheduleAbleVisit' && item.value!=null)
          nextValue = item.value;
}

var res = getValues('vsd');
if(res){
      vsd.reverse().forEach(getNextValue);
      //logMsg(nextValue);
      //logMsg(brDate);
      //compare next vsd with cycle date: vsd
}
```

clearValue( )

Use this helper function to clear values in a target populated by a calculation rule, whether it is a text, number, or multiple choice question.

To properly clear a rule target, this function needs to be used as the rule's return statement. See examples below.

> **Note:**
>
> Even if you clear a question's data, the whole answer history is preserved. All actions performed on a question are listed on the Answer & Visit History sidebar and in the audit trail. In there you can also see which actions result from rule execution. For more information see How does clearing data impact my study?.

Syntax

```
clearValue()
```

Parameters

None.

Return value

This function itself is used as the return value of a calculation rule. It returns an indicator to the rule process to start data clear in the back end.

Examples

Example 3-108 Depending on a control variable, either clear or return calculated value

```
logMsg(txtRuleVariable);
if(txtRuleVariable=='clear')
    return clearValue();
return txtRuleVariable;
```

Example 3-109 Given a drop-down (choice) control, either clear the control or set label

```
if(txtRuleVariable=='clear')
    return clearValue();
return setChoiceLabel("Yes");
```

Related Topics

- Create a rule for a calculated value
  You can create a rule that enables the system to automatically calculate a value in a form. This allows you to automatically calculate certain values based on manually entered form data. reducing calculation errors and simplifying forms.
- About data clear and data removal
- How does clearing data impact my study?
- What happens when one of the rule's inputs is cleared?
  When a question used as a rule's input is updated (cleared, removed or replaced with a new value), the system processes the change causing the impacted rule or rules to be re-run. The end result depends on the rule logic and is applicable to all types of custom rules.

4

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

1. Using a subject for testing, go to the given visit and form containing the value to check, in this example the *oral temperature value* <`tempval`>.
2. Update the form items `tempval` and `tempunit` as in the following table and verify the result is as listed:

tempval	tempunit	Notes	Result
35.0	C	`tempval` matches the lower range limit for °C temperatures (35.0 - 40.6).	No query
34.9	C	`tempval` is lower than the lower range limit for °C temperatures (35.0).	Query
35.1	C	`tempval` is in range for °C temperatures (35.0 - 40.6).	No query
40.6	C	`tempval` matches the higher range limit for °C temperatures (40.6).	No query
40.5	C	`tempval` is in range for °C temperatures (35.0 - 40.6).	No query
40.7	C	`tempval` is higher that the higher range limit for °C temperatures (40.6).	Query
40.7	F	`tempval` is lower than the lower range limit for °F temperatures (95 - 105).	Query
94.0	F	`tempval` is lower than the lower range limit for °F temperatures (95 - 105).	Query
95.0	F	`tempval` matches the lower range limit for °F temperatures (95 - 105).	No query
96.0	F	`tempval` is in range for °F temperatures (95 - 105).	No query
105.0	F	`tempval` matches the higher range limit for °F temperatures (95 - 105).	No query
104.0	F	`tempval` is in range for °F temperatures (95 - 105).	No query
106.0	F	`tempval` is higher than the higher range limit for °F temperatures (95 - 105).	Query
103.0	F	`tempval` is in range for °F temperatures (95 - 105).	No query
103.0	C	`tempval` is higher that the higher range limit for °C temperatures (40.6).	Query

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

Other examples

Example 4-1 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

1. Using a subject for testing, go to the given visit and form containing the iems to check, in this example the *Date of discontinuation* <`dt`> and *Reason for withdrawal* <`reason`>.
2. Update the form items `dt` and `reason` as in the following table and verify the result is as listed:

Step	dt	reason	Result
Add date in `dt` only.	Value	Null	Query
Add `reason`.	Value	Value	No query
Clear `reason` only.	Value	Null	Query
Add new `reason`.	Value	Value	No query
Clear `dt` only.	Null	Value	No query
Clear `reason`.	Null	Null	No query

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

Other examples

Example 4-2 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);}
     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

1. Using a subject for testing, go to the given visit and form containing the items to check, in this example the *height value* <hght>, *height unit* <hghtunt>, *weight value* <wght>, and *weight unit* <wghtunt>.
2. Update the form items hght, hghtunt, wght and wghtunt as in the following table and verify the result is as listed:

hght	hghtunt	wght	wghtunt	Result
175	cm	50	kg	16.3
175	cm	50	lb	7.4
175	cm	78.0	lb	11.5
175	cm	78.0	kg	25.5
72	cm	78.0	kg	150.5
72	in	78.0	kg	23.3
0.0	in	78.0	kg	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

1. Using a subject for testing, go to the given visit and form containing the items to map, in this example the dropdown question *administration route* <ROUTE>, and *specify* <ROUTEOTHR> displayed when selecting "Other" as an answer.
2. Update the form items ROUTE, and ROUTEOTHR when prompted, as in the following table and verify the result is as listed:

ROUTE	ROUTEOTHR	Result
'Oral'	NA.	Target item is populated as 'Oral'.
'Topical'	NA.	Target item is populated as 'Topical'.
'IM'	NA.	Target item is populated as 'IM'.
'Other'	Null	Target item is populated as 'Other'.
'Other'	'Unknown'	Target item is populated as 'Other: Unknown'.
Null	Null	Target item is cleared.

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

Other examples

Example 4-3 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

1. Using a subject for testing, go to the given visit and form containing the iems to check, in this example the *Injection site for study vaccine administration* <`INJSITELOC`> .
2. Update the form item `INJSITELOC` as in the following table and verify the result is as listed:

Step	Result
a. Select `INJSITELOC` as 'Other'.	Query
b. Select `INJSITELOC` as any value other than 'Other'.	No query
c. Select `INJSITELOC` as 'Other'.	Query
d. Clear `INJSITELOC`.	No query
e. Select `INJSITELOC` as any value other than 'Other' and different to the one selected in step b.	No query

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

Other examples

Example 4-4 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 4-5 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 4-6 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 4-7 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 4-8 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 4-9 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

1. Using a subject for testing, go to the given visit and form containing the items to check, in this example the *systolic blood pressure* <`SYS`> and *diastolic blood pressure* <`DIA`>.
2. Update the form items `SYS` and `DIA` as in the following table and verify the result is as listed:

SYS	DIA	Result
120	`Null`	No query
120	120	Query
120	119	No query
120	121	Query
123	121	No query
115	121	Query
`Null`	121	No query
125	121	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.

1. Using a subject for testing, go to the given visit and form containing the iems to check, in this example the *subject initials* <`txtitem1`> .
2. Update the form item `txtitem1` as in the following table and verify the result is as listed:

txtitem1	Result
'ABC'	No query
'abc'	No query
'AbC'	No query
'A-b'	No query
'A-A'	No query
'a-z'	No query
'A'	Query
'AB'	Query
'Ab'	Query
'A_B'	Query
'123'	Query
'A13'	Query
'AB@'	Query
'AB$'	Query
'AB!'	Query
'AB&'	Query
'A B'	Query
'Abc'	No query

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

Other examples

Example 4-10 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

1. Using a subject for testing, go to the given visit and form containing the items to check, in this example the *date of birth* <dob> and *date of inform consent signed* <infconst>.
2. Update the form items dob and infconst as in the following table and verify the result is as listed:

dob	infconst	Result
2-Jan-1942	2-Jan-2021	79
3-Jan-1942	2-Jan-2021	78
1-Jan-1942	2-Jan-2021	79
1-Jan-1993	2-Jan-2021	28
3-Jan-1993	2-Jan-2021	27
3-Jan-1993	3-Jan-2021	28
3-Jan-1993	Null	Null

Date examples

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

1. Using a subject for testing, go to the given visit and form containing the date items to compare, in this example the *onset date* <`onstdt`> and *date of completion* <`compdt`>.
2. Update the form items `onstdt` and `compdt` as in the following table and verify the result is as listed:

onstdt	compdt	Result
10-May-2021	`Null`	No query
10-May-2021	10-May-2021	No query
11-May-2021	10-May-2021	Query
09-May-2021	10-May-2021	No query
09-Jun-2021	10-May-2021	Query
11-Apr-2021	10-May-2021	No query
`Null`	10-May-2021	No query
12-May-2021	10-May-2021	Query
12-May-2021	14-May-2021	No query

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

Other examples

Example 4-11 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 4-12 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 4-13 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 4-14 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

1. Using a subject for testing, go to the given visit and form containing the date items to compare, in this example the *end date* <`enddt`> and *start date* <`stdt`>.
2. Update the form items `enddt` and `stdt` as in the following table and verify the result is as listed:

enddt	stdt	Result
Null	10-May-2021 11:00 AM	No query
10-May-2021 11:00 AM	10-May-2021	No query
10-May-2021 11:00 AM	10-May-2021 11:01 AM	Query
10-May-2021 11:00 AM	10-May-2021 10:59 AM	No query

enddt	stdt	Result
10-May-2021 11:00 AM	11-May-2021 10:59 AM	Query
10-May-2021 11:00 AM	09-May-2021 10:59 AM	No query
10-May-2021 11:00 AM	Null	No query
12-May-2021	10-May-2021 11:00 PM	Query
10-May-2021 11:05PM	10-May-2021 11:00 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

1. Using a subject for testing, go to the given visit and form containing the date items to compare, in this example the *date of study completion* <DSENDT1> and *'V5C' Visit date* <VISDAT>.
2. Update the form items `DSENDT1` and `VISDAT` as in the following table and verify the result is as listed:

DSENDT1	VISDAT	Result
Null	10-May-2021	No query
10-May-2021	10-May-2021	No query
10-Jun-2021	10-May-2021	Query
09-Jun-2021	10-May-2021	No query
10-May-2022	10-May-2021	Query
11-May-2021	10-May-2021	No query
11-May-2021	05-May-2022	Query
11-May-2021	11-May-2021	No query
11-May-2021	Null	No query
11-May-2021	06-May-2022	Query

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

Other examples

Example 4-15 '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

1. Using a subject for testing, go to the given visit and form containing the date items to compare, in this example the *Day 1 visit start date* <DOV> and *date of measurement* <MEASDT>.
2. Update the form items DOV and MEASDT as in the following table and verify the result is as listed:

DOV	MEASDT	Result
Null	10-May-2021	No query
10-May-2021	10-May-2021	Query
11-May-2021	10-May-2021	No query
09-May-2021	10-May-2021	Query
12-Apr-2022	10-May-2021	Query
12-Apr-2022	11-Apr-2021	No query
12-Apr-2022	12-Apr-2022	Query
12-Apr-2022	14-Mar-2021	Query
12-Apr-2022	15-Mar-2021	No query
13-Apr-2021	15-Mar-2022	Query
13-Apr-2022	Null	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

```
return getDateDMYFormat(dt2,"HH:mm");
```

Definitions

getDateDMYFormat( )
Returns a date or datetime in DD-Mon-YYYY format, including time elements if applicable.

dt2
Corresponds to *DateTime2* in rule description.

"HH:mm"
Time format passed as a parameter for the `getDateDMYFormat()` helper function to define the output format of the time elements present.

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

1. Using a subject for testing, go to the given visit and form containing the date item to map, in this example the *date time in form 2* <`dt2`> to be mapped into *date time in form 1*.
2. Update the form item `dt2` as in the following table and verify the result mapped to the target item in form 1 is as listed:

dt2	**Mapped value in *form 1***
'30-Oct-2021 01:23'	'30-Oct-2021 01:23'
'31-Oct-2021 23:59'	'31-Oct-2021 23:59'
`Null`	`Null`

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

Other examples

Example 4-16 Map date of visit in status form as visit date

```
return getDateDMYFormat(date);
```

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

1. Using a subject for testing, go to the given visit and form containing the date items to compare, in this example the *AE start date* <`aestdt`> and *date of informed consent* <`infconsdt`>.
2. Update the form items `aestdt` and `infconsdt` as in the following table and verify the result is as listed:

aestdt	infconsdt	Result
Null	02-Dec-2021	No query
02-Dec-2021	02-Dec-2021	No query
01-Dec-2021	02-Dec-2021	Query
UNK-Dec-2021	02-Dec-2021	No query
UNK-Nov-2021	02-Dec-2021	Query
03-Dec-2021	02-Dec-2021	No query
03-Dec-2021	05-Dec-2021	Query
03-Dec-2021	02-Dec-2021	No query
03-Dec-2021	01-Jan-2022	Query
03-Dec-2021	04-Dec-2021	Query
03-Dec-2021	Null	No query
03-Dec-2021	02-Dec-2021	No query
01-Dec-2021	02-Dec-2021	Query

Note:

Repeat the above steps if the form is present in multiple visits.

Other examples

Example 4-17 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 4-18 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 4-19 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

1. Using a subject for testing, go to the given visit and form containing the date item to check, in this example the *date of initial diagnosis* <`DIADT`> .
2. Update the form item `DIADT` as in the following table and verify the result is as listed:

DIADT	Result
10-May-2021	No query
UNK-UNK-2021	Query
UNK-May-2021	No query
05-Jun-2021	No query
UNK-UNK-2021	Query
`Null`	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

1. Using a subject for testing, go to the given visit and form containing the date items to compare, in this example the *date of informed consent* <`icdat`> and *visit date* <`vstdt`>.
2. Update the form items `icdat` and `vstdt` as in the following table and verify the result is as listed:

icdat	vstdt	Result
10-May-2021	Null	No query
10-May-2021	10-May-2021	No query
11-May-2021	10-May-2021	Query. Verify correct date values are populated in the Query Text.
09-May-2021	10-May-2021	No query
09-Jun-2021	10-May-2021	Query. Verify correct date values are populated in the Query Text.
11-Apr-2021	10-May-2021	No query
Null	10-May-2021	No query
12-May-2021	10-May-2021	Query. Verify correct date values are populated in the Query Text.
12-May-2021	14-May-2021	No query

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

Other examples

Example 4-20 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 4-21 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 4-22 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=getDateDMYFormat(colldt,"HH:mm");
var dt2=getDateDMYFormat(vaccdt,"HH:mm");
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
}
```

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.

getDateDMYFormat( )
Returns a date or datetime in DD-Mon-YYYY format, including time elements if applicable.

`"HH:mm"`
Time format passed as a parameter for the `getDateDMYFormat()` helper function to define the output format of the time elements present.

setQueryMessage( )

Specify dynamic query text using the `setQueryMessage( )` function.

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

1. Using a subject for testing, go to the given visit and form containing the date items to compare, in this example the *sample collection date* <`colldt`> and *study vaccine administration date* <`vaccdt`>.
2. Update the form items `colldt` and `vaccdt` as in the following table and verify the result is as listed:

colldt	vaccdt	Result
10-May-2021 10:00	`Null`	No query
10-May-2021 10:00	10-May-2021 10:00	No query
10-May-2021 10:01	10-May-2021 10:00	Query. Verify correct date values are populated in the Query Text.
10-May-2021 09:59	10-May-2021 10:00	No query
11-Jun-2021 10:00	10-May-2021 10:00	Query. Verify correct date values are populated in the Query Text.
11-Apr-2021 07:01	10-May-2021 10:00	No query
`Null`	10-May-2021 10:00	No query

Other examples

Example 4-23 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=getDateDMYFormat(ecgdt,"HH:mm");
var dt2=getDateDMYFormat(compdt);
var qtstr="Date is "+dt1+" after Date of Study Completion or Discontinuation
"+dt2+". Please correct or confirm date(s).";
setQueryMessage(qtstr);
return false;
}
```

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

1. Using a subject for testing, go to the given visit and form containing the date items to compare, in this example the *AE stop date* <`aeenddt`> and *date of informed consent* <`infconsdt`>.
2. Update the form items `aeenddt` and `infconsdt` as in the following table and verify the result is as listed:

aeenddt	infconsdt	Result
Null	02-Dec-2021	No query
02-Dec-2021	02-Dec-2021	No query
01-Dec-2021	02-Dec-2021	Query.
UNK-Dec-2021	02-Dec-2021	No query
UNK-Nov-2021	02-Dec-2021	Query.
03-Dec-2021	02-Dec-2021	No query
03-Dec-2021	05-Dec-2021	Query
03-Dec-2021	02-Dec-2021	No query
03-Dec-2021	01-Jan-2022	Query
03-Dec-2021	04-Dec-2021	Query
03-Dec-2021	Null	No query
03-Dec-2021	02-Dec-2021	No query
01-Dec-2021	02-Dec-2021	Query

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

Other examples

Example 4-24 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 4-25 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) repeating 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

1. Using a subject for testing, go to the given visit and form containing the iems to check, in this example the *indication* <INDICAT> on the Concomitant Medications form.
2. Update the form item INDICAT as in the following table and verify the result is as listed:

Step	Result
a. Assign the INDICAT item to an adverse event.	Query
b. Go to the first repeating form instance of the Adverse Event form and complete all items of the repeating form for which it is required to check atleast one non deleted instance is present: AESER, AESERY, AESERYOTH, AESTDT, AEONGO, AEENDT, and AEOUT.	No query
c. Clear all the item in the first repeating form instance completed in previous step.	No query
d. Delete the first repeating form instance.	Query
e. Update the INDICAT item so it is not assigned to an adverse event.	No query
f. Update the INDICAT item so it is assigned to an adverse event again.	Query
g. Create a second repeating form instance 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
h. Create third repeating form instance and complete the remaining required items from previous step in that third repeating form instance.	Query
i. Complete the remaining required items in the second repeating form instance.	No query
j. Delete the third repeating form instance.	No query

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

Other examples

Example 4-26 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

1. Using a subject for testing, go to the given visit and form containing the iems to check, in this example the *RECIST evaluation number* <`TLFEVAL`> in the specified repeating form instance.
2. Update the form item `TLFEVAL` as in the following table and verify the result is as listed:

Step	Notes	Result
a. In the first repeating form (1RF) instance enter the `TLFEVAL` item as *value 1* (input value is as per item type i.e. Text/Date/Number).	Only one repeating form instance.	No query.
b. Go to a second repeating form (2RF) instance and enter the `TLFEVAL` item as *value 1* (same as in previous step).	Matching values in 1RF and 2RF: • 1RF - *value 1* • 2RF -*value 1*	Query in 1RF and 2RF.
c. Update `TLFEVAL` item in 2RF instance as *value 2* (different to *value 1*).	Different values in 1RF and 2RF: • 1RF - *value 1* • 2RF -*value 2*	No queries.
d. Update `TLFEVAL` item in 1RF instance as *value 2* (same as in previous step).	Matching values in 1RF and 2RF: • 1RF - *value 2* • 2RF -*value 2*	Query in 1RF and 2RF.
e. Clear `TLFEVAL` item in 2RF.	Different values in 1RF and 2RF: • 1RF - *value 2* • 2RF -`Null`	No queries.
f. Enter `TLFEVAL` item in 2RF as *value 3* (different to those entere before).	Different values in 1RF and 2RF: • 1RF - *value 2* • 2RF -*value 3*	No queries.
g. Go to a third repeating form (3RF) instance and enter the `TLFEVAL` item as *value 2*.	Matching values in 1RF and 3RF: • 1RF - *value 2* • 2RF -*value 3* • 3RF - *value 2*	Query in 1RF and 3RF. No query in 2RF.

Step	Notes	Result
h. Update the `TLFEVAL` item in 3RF as *value 1*.	Different values in 1RF, 2RF and 3RF: • 1RF - *value 2* • 2RF -*value 3* • 3RF - *value 1*	No queries.
i. Update the `TLFEVAL` item in 3RF as *value 3*.	Matching values in 2RF and 3RF: • 1RF - *value 2* • 2RF -*value 3* • 3RF - *value 3*	Query in 2RF and 3RF. No query in 1RF.
j. Delete the 2RF instance.	Different values in 1RF and 3RF, and no 2RF present: • 1RF - *value 2* • 3RF - *value 3*	No queries.

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

Other examples

Example 4-27 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

1. Using a subject for testing, go to the given visit and form containing the iems to check, in this example the *Adverse Event number* <`aenumval`> and *severity* <`sevval`>in the specified repeating form instance.
2. Update the form items `aenumval` and `sevval` as in the following table and verify the result is as listed:

Step	Notes	Result
a. In the first repeating form (1RF) instance enter the `aenumval` item as '**1**' (AE1) and select `sevval` as '**Grade 1**'.	Only one repeating form instance.	No query.
b. Go to a second repeating form (2RF) instance and enter the `aenumval` item as '**1**' (AE1) and select `sevval` as '**Grade 1**' (same as in previous step).	Matching values in 1RF and 2RF: • 1RF: – AE1 – Severity Grade 1 • 2RF: – AE1 – Severity Grade 1	Query in 2RF.
c. Update `sevval` item in 2RF instance as '**Grade 2**'.	Different values in 1RF and 2RF: • 1RF: – AE1 – Severity Grade 1 • 2RF: – AE1 – Severity Grade 2	No queries.
d. Update `sevval` item in 2RF instance back as '**Grade 1**'.	Matching values in 1RF and 2RF: • 1RF: – AE1 – Severity Grade 1 • 2RF: – AE1 – Severity Grade 1	Query in 2RF.
e. Update `aenumval` item in 2RF instance as '**2**' (AE2).	Different values in 1RF and 2RF: • 1RF: – AE1 – Severity Grade 1 • 2RF: – AE2 – Severity Grade 1	No queries.
f. Create a thrid repeating form (3RF) instance and enter `aenumval` item as '**1**' (AE1) and `sevval` as '**Grade 1**'.	Matching values in 1RF and 3RF: • 1RF: – AE1 – Severity Grade 1 • 2RF: – AE2 – Severity Grade 1 • 3RF: – AE1 – Severity Grade 1	Query in 3RF.

Step	Notes	Result
g. Update `aenumval` in 3RF as '**3**' (AE3).	Different values in all repeating form instances: • 1RF: – AE1 – Severity Grade 1 • 2RF: – AE2 – Severity Grade 1 • 3RF: – AE3 – Severity Grade 1	No queries.
h. Update the `aenumval` item in 3RF as '**2**' (AE2).	Matching values in 2RF and 3RF: • 1RF: – AE1 – Severity Grade 1 • 2RF: – AE2 – Severity Grade 1 • 3RF: – AE2 – Severity Grade 1	Query in 3RF.
i. Update the `sevval` item in 2RF as '**Grade 3**'.	Different values in all repeating form instances: • 1RF: – AE1 – Severity Grade 1 • 2RF: – AE2 – Severity Grade 3 • 3RF: – AE2 – Severity Grade 1	No queries.

Other examples

Example 4-28 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

1. Using a subject for testing, go to the given visit and form containing the iems to check, in this example the *Does the subject have any relevant Medical History?* question <MHYes> .
2. Update the form item MHYes as in the following table and verify the result is as listed:

Step	Result
a. In the flat section of a first instance of a two-section form (*Form1*), enter the MHYes item as '**Yes**'.	Query
b. In *Form1*, create a first table instance and complete all items.	No query
c. In *Form1*, clear all items in the first table instance.	No query
d. In *Form1*, delete the first table instance.	Query
e. In the flat section of *Form1*, update the MHYes item as '**No**'.	No query
f. In the flat section of *Form1*, update the MHYes item as '**Yes**'.	Query
g. In *Form1*, create a new first table instance and complete some items.	No query
h. Create a second two-section form instance (*Form2*) and enter the MHYes item as '**Yes**' in the flat section.	Query in *Form2*
i. In *Form2*, create a first table instance and complete some items.	No queries
j. Delete *Form2*.	No query

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

Other examples

Example 4-29 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(getStringFromChoice(PEDT)==='Yes')
{
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

1. Using a subject for testing, go to the given visit and form containing the iems to check, in this example the *Lesion ID* <`lesid`> in the *Target Lesion* form.
2. Create form instances updating the form item `lesid` as in the following table and verify the result is as listed:

Step	Result
a. Create a first instance of a two-section form (*Form1*) and enter the `lesid` item with any value'.	No query
b. Create a second instance of a two-section form (*Form2*) and enter the `lesid` item with any value'.	No query
c. Create a third instance of a two-section form (*Form3*) and enter the `lesid` item with any value'.	No query
d. Create a fourth instance of a two-section form (*Form4*) and enter the `lesid` item with any value'.	No query
e. Create a fifth instance of a two-section form (*Form5*) and enter the `lesid` item with any value'.	No query
f. Create a sixth instance of a two-section form (*Form6*) and enter the `lesid` item with any value'.	Query in all six instances.
g. Delete *Form2*.	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

1. Using a subject for testing, go to the given visit and form containing the iems to check, in this example the *Lesion ID* <`lesid`> in the specified two-section form instance.
2. Update the form item `lesid` as in the following table and verify the result is as listed:

Step	Notes	Result
a. In the first two-section form instance (*Form1*) enter the `lesid` item as '**1**'.	Only one two-section form instance.	No query.
b. Create a second two-section form instance (*Form2*), and enter the `lesid` item as '**1**'.	Matching values in *Form1* and *Form2*: • *Form1* - **1** • *Form2* - **1**	Query in *Form1* and *Form2*.
c. Update `lesid` item in *Form2* as '**2**'.	Different values in *Form1* and *Form2*: • *Form1* - **1** • *Form2* -**2**	No queries.

Step	Notes	Result
d. Update `lesid` item in *Form1* as '**2**'.	Matching values in *Form1* and *Form2*: • *Form1* - **2** • *Form2* -**2**	Query in *Form1* and *Form2*.
e. Clear `lesid` item in *Form2*.	Different values in *Form1* and *Form2*: • *Form1* - **2** • *Form2* -`Null`	No queries.
f. Update `lesid` item in *Form2* as '**3**'.	Different values in *Form1* and *Form2*: • *Form1* - **2** • *Form2* -**3**	No queries.
g. Create a third two-section form instance (*Form3*), and enter the `lesid` item as '**2**'.	Matching values in *Form1* and *Form3*: • *Form1* - **2** • *Form2* -**3** • *Form3* - **2**	Query in *Form1* and *Form3*. No query in *Form2*.
h. Update `lesid` item in *Form3* as '**1**'.	Different values in *Form1*, *Form2* and *Form3*: • *Form1* - **2** • *Form2* -**3** • *Form3* - **1**	No queries.
i. Update `lesid` item in *Form3* as '**3**'.	Matching values in *Form2* and *Form3*: • *Form1* - **2** • *Form2* -**3** • *Form3* - **3**	Query in *Form2* and *Form3*. No query in *Form1*.
j. Delete *Form2*.	Different values in *Form1* and *Form3*, and no *Form2* present: • *Form1* - **2** • *Form3* - **3**	No queries.

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

Other examples

Example 4-30 Method of Assessment should stay the same across all records

```
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

1. Using a subject for testing, go to the given visit and form containing the iems to check, in this example the *Abnormality/Condition* <`MHCondition`> in the specified table instance of the *Medical History* two-section form.
2. Update the form item `MHCondition` as in the following table and verify the result is as listed:

Step	Notes	Result
a. In the first two-section form instance (*Form1*), create a first table instance (*Row1)* and enter the `MHCondition` item as '*value 1*'.	Only one two-section form instance.	No query.
b. In *Form1*, create a second table instance (*Row2)* and enter the `MHCondition` item as '*value 1*'.	Matching values in *Row1* and *Row2* of *Form1*: • *Form1*: – *Row1 - value 1* – *Row2 - value 1*	Query.
c. Update `MHcondition` item in *Row2* of *Form1* as '*value 2*'.	Different values in *Row1* and *Row2* of *Form1*: • *Form1*: – *Row1 - value 1* – *Row2 - value 2*	No query.
d. Update `MHcondition` item in *Row1* of *Form1* as '*value 2*'.	Matching values in *Row1* and *Row2* of *Form1*: • *Form1*: – *Row1 - value 2* – *Row2 - value 2*	Query.
e. Clear `MHCondition` item in *Row2* of *Form1*.	Different values in *Row1* and *Row2* of *Form1*: • *Form1*: – *Row1 - value 2* – *Row2* - `Null`	No queries.
f. Update `MHcondition` item in *Row2* of *Form1* as '*value 3*'.	Different values in *Row1* and *Row2* of *Form1*: • *Form1*: – *Row1 - value 2* – *Row2 - value 3*	No queries.
g. In *Form1*, create a third table instance (*Row3)* and enter the `MHCondition` item as '*value 2*'.	Matching values in *Row1* and *Row3* of *Form1*: • *Form1*: – *Row1 - value 2* – *Row2 - value 3* – *Row3 - value 2*	Query.
h. Update `MHcondition` item in *Row3* of *Form1* as '*value 1*'.	Different values in *Row1*, *Row2* and *Row3* of *Form1*: • *Form1*: – *Row1 - value 2* – *Row2 - value 3* – *Row3 - value 1*	No queries.
i. Update `MHcondition` item in *Row3* of *Form1* as '*value 3*'.	Matching values in *Row2* and *Row3* of *Form1*: • *Form1*: – *Row1 - value 2* – *Row2 - value 3* – *Row3 - value 3*	Query.

Step	Notes	Result
j. Create a second two-section form instance (*Form2*) and create a first table instance (*Row1)* and enter the `MHCondition` item as '*value 3*'.	Matching values in Matching values in *Row2* and *Row3* of *Form1*: • *Form1*: – *Row1 - value 2* – *Row2 - value 3* – *Row3 - value 3* • *Form2*: – *Row1 - value 3*	Query in *Form1*. No query in *Form2*.
k. In *Form2*, create a second table instance (*Row2)* and enter the `MHCondition` item as '*value 3*'.	Matching values in Matching values in *Row2* and *Row3* of *Form1* and in *Row1* and *Row2* of *Form2*: • *Form1*: – *Row1 - value 2* – *Row2 - value 3* – *Row3 - value 3* • *Form2*: – *Row1 - value 3* – *Row2 - value 3*	Query in *Form1* and *Form2*.
l. Delete *Row2* in *Form2*.	Matching values in Matching values in *Row2* and *Row3* of *Form1*: • *Form1*: – *Row1 - value 2* – *Row2 - value 3* – *Row3 - value 3* • *Form2*: – *Row1 - value 3*	Query in *Form1*. No query in *Form2*

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

Other examples

Example 4-31 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 4-32 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.

5

Frequently Asked Questions (FAQs)

What if my JavaScript expression does not return a value for a calculation?

A JavaScript function always returns a value, if it is properly defined.

If you do not specify a return value, it remains undefined and can result in unexpected behavior. Therefore, make sure you include a `return` statement on your rule to specify the value your JavaScript expression should return. For step-by-step instructions, see Create a rule for a calculated value.

What happens when one of the rule's inputs is cleared?

When a question used as a rule's input is updated (cleared, removed or replaced with a new value), the system processes the change causing the impacted rule or rules to be re-run. The end result depends on the rule logic and is applicable to all types of custom rules.

> **Tip:**
>
> You can view the resulting actions in the Answer & Visit History sidebar, each stated as the result of rule execution.

- For rules that populate calculated values, if a question used as an input for the calculation is removed or cleared, the rule re-runs with an empty value as the input. This would typically result in the calculated value also getting removed or cleared (respectively). However, you should keep in mind that the output always depends on what the rule logic dictates.

> **Note:**
>
> Clearing or removing a value makes it null and it is not the same as making it zero.

- If a question with an opened automated query is cleared or removed, either directly or as a result of calculation rule, the query will behave according to what the rule dictates. Typically, the automated query would get **Closed**, but this is only if the condition for the query no longer applies and as long as the rule logic doesn't dictate differently.
- For rules that raise queries or send notifications, the result of clearing the input depends on how an empty input would trigger the rule. So, only for the case of rules that raise queries or send notifications when detecting empty values, clearing a value results on a notification being sent or a query being raised against the question.

What if I published a rule by mistake?

If you published a rule in error, depending on your role, you can disable that rule so it stops running in all study versions in Production. For more information see Disable a rule.

Can I publish a single rule in Production?

Yes you can. To publish one rule at a time in Production, all you have to do is move the status slider from Approved to Published, while the study is still in Testing mode.

If you want to publish multiple rules at a time, all you have to do is make sure they have a status of Approved. When you move the study from the Testing to the Approved container, all rules with a status of **Approved** will be published in Production.

This task can be performed if you you're assigned the **Rule Publisher** role by the user administrator.

6
Revision history

Table 6-1 Revision History

Date	Part Number	Description
27-September-2024	F99798-02	Formatting updates were made throughout the publication.
30-August-2024	F99798-01	Original version of the document.