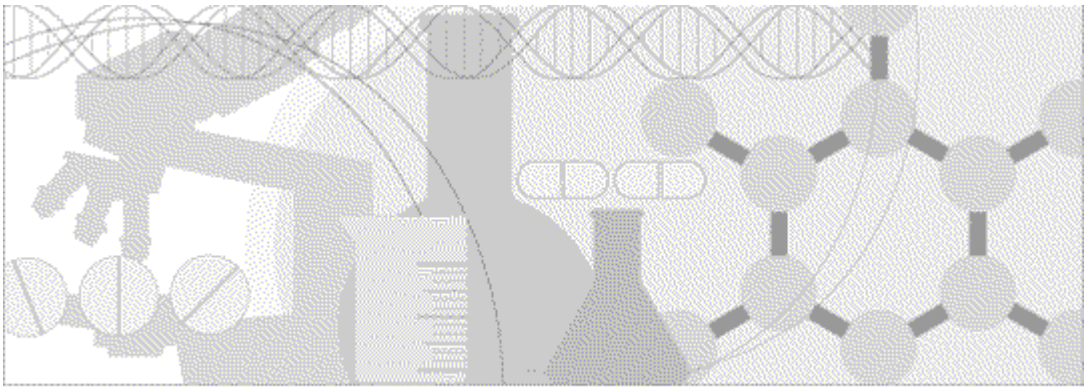
Rules Reference Guide

Oracle® Health Sciences Central Designer Release 2.0



ORACLE

Part Number: E37917-01

Contents

About this guide

In this preface

Overview of this guide

The *Rules Reference Guide* is a reference to the tools that are available for creating rule expressions, including:

- Study object properties.
- Functions.
- Constants.
- Data mappings.
- Methods, operators, and literals.

Prerequisites

You should have experience in one or more of the following clinical areas:

- Data management
- Data analysis
- Study design
- Library management
- Project management

Additionally, rule developers must have the following levels of programming expertise, depending on the types of rules they will be creating.

Type of rule	Required programming expertise
Simple edit checks using rule templates.	None.
Comparisons, calculations.	• Understanding of C# or similar (for example, Java, C, or C++) expression syntax. • Ability to incorporate functions in expressions.
User-defined functions.	• Ability to program in a .NET language (for example, C#) and create .NET assemblies.

Users

This guide is for users of the Central Designer application, including:

User	Description
Clinical data manager	A user who is involved in the study design process.
Library administrator	A user with library administration privileges for one or more clinical areas.
Central Designer administrator	An IT representative who supports and maintains the application from a technology and infrastructure perspective.
Study design team	Users who implement a study. This team includes several roles such as form designer, rule designer, and study workflow designer.
Medical project manager	A user who works with the CRF designer to create an annotated study protocol, usually based on existing CRFs for the therapeutic area. This user is concerned with the overall set of study questions and logic, not the individual details of form design.
Statistician	A user who interacts with study designers with respect to data collection needs for analysis.
Translator	A user who is responsible for translating text in clinical studies, forms, items, and rule queries to a specified language.

Related information

Documentation

All documentation is available from the Oracle Software Delivery Cloud (https://edelivery.oracle.com) and the Download Center (https://extranet.phaseforward.com).

All documents may not be updated for every Central Designer release. Therefore, the version numbers for the documents in a release may differ. For a complete list of the documents in this Central Designer release, their release version numbers, and part numbers, see the *Release Notes*.

Item	Description
Release Notes	The *Release Notes* document provides detailed information about the requirements, enhancements, and fixed issues in the current release.
Known Issues	The *Known Issues* document provides detailed information about the known issues in this release, along with workarounds, if available. **Note:** The most current list of known issues is available on the Extranet. To sign in to the Extranet, go to https://extranet.phaseforward.com.
Installation Guide	The *Installation Guide* provides system requirements and instructions for installing and upgrading the Oracle® Health Sciences Central Designer software and the Oracle® Health Sciences Central Designer Administrator software. This document is also available from the Documentation CD.
Administrator Guide	The *Administrator Guide* describes how to use the Oracle® Health Sciences Central Designer Administrator software to set up users, permissions, system configuration parameters, and catalog defaults. This document is also available from the Oracle® Health Sciences Central Designer Administrator application user interface (HTML format) and the Documentation CD (PDF format).
User Guide	The *User Guide* introduces the study design environment in the Oracle® Health Sciences Central Designer application and describes how to work as a study design team in that environment, including how to: • Work collaboratively. • Maximize study design efficiency by reusing study objects. • Manage collections of study objects. This document is also available from the Oracle® Health Sciences Central Designer application user interface (HTML format) and the Documentation CD (PDF format).
InForm Design Guide	The *InForm Design Guide* describes how to design a study for deployment to the InForm application. This document is also available from the Oracle® Health Sciences Central Designer application user interface (HTML format) and the Documentation CD (PDF format).

Item	Description
Rules Reference Guide	The *Rules Reference Guide* is a reference to the tools that are available for creating rule expressions, including: • Study object properties. • Functions. • Constants. • Data mappings. • Methods, operators, and literals. This document is also available from the Oracle® Health Sciences Central Designer application user interface (HTML format) and the Documentation CD (PDF format).
Secure Configuration Guide	The *Secure Configuration Guide* provides an overview of the security features provided with the Oracle® Health Sciences Central Designer application, including details about the general principles of application security, and how to install, configure, and use the Central Designer application securely.

If you need assistance

Oracle customers have access to support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info, or if you are hearing impaired, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs.

CHAPTER 1

Rule expressions

In this chapter

About the rule expression language

Use the rule expression language to create the expression component of a data-entry rule, workflow rule, or global condition.

You create rule expressions in:

- The Expression tab of the Rule Wizard.
- The Expression workspace of the dialog boxes used to create or edit a workflow rule or global condition.

As you type, a list of the rule model components that you can use in the expression (study objects and their properties, functions, constants, and data mappings) appears in the Expression workspace. When you select a rule model component, a tooltip appears to indicate its usage (for example, the parameters and their data types required for using a function). The list changes dynamically to support the contents of the expression as you create it.

You can also use the following methods for creating a rule expression:

- Drag rule model components into the Expression workspace from the tabs that appear on the right side of the workspace.
- Type an expression directly in the Expression workspace.

A rule expression can include any valid C# expression that can appear on the right side of the equals sign (=), including:

- Operators and literals.
- Study objects and their rule model properties.
- Functions.
- Constants defined for the study.
- Data mapping study objects and their rule model properties.
- Methods. You can use any method, including:
    - Math methods.
    - Data set methods.
    - Methods for repeating study objects.

A rule expression cannot include:

- Complex structures, such as if statements (conditionals are allowed) and looping statements.

  Use functions to build more complex rule expressions. For more information, see ***Functions tab of the Rule Wizard*** (on page 54) (in the *Rules Reference Guide*).

- Multi-line expressions.

  The rule expression does not allow flow-of-control operators, such as *like*, *if*, *then*, *for*, or *while*. You can use parentheses to provide better readability and grouping.

An expression must evaluate to one of the following types:

- Integer
- Float
- Boolean
- Text
- Date time

Note: A rule expression is similar to a switch statement in C++, C#, or Java.

Components of the rule expression language

Component	Description
Operators and literals (on page 135)	Operators and literals connect information in the rule expression. Use standard C# and Java operators and literals to create rule expressions.
Data (on page 18)	Data includes the following: • Values provided for study objects within the scope of the rule. • Values of rule model properties of study objects within the scope of the rule. • Methods for repeating study objects.

Component	Description
Functions (on page 54)	A function is a reusable piece of code that extends the behavior of a rule. Functions allow experienced programmers to build complex rule expressions and make them available in libraries or studies. A function can be part or all of a rule expression. If a function has parameters, you must specify their substitution values after you drag the function into the rule expression. You can use the following to specify parameter substitution values: ▪ Numeric values. ▪ Values of study objects and their rule model properties. ▪ Constants. ▪ Another function. ▪ Global study objects and their rule model properties.
Constants (on page 106)	A constant is a value that is defined in a library or study and that can be referenced by any rule.
Data Mappings (on page 114)	Study objects and properties that have a global scope for rule creation are called data mappings. The following data mappings can be used in any rule in the study in which you are working: • RefNames of the following study objects in the study or library: ▪ Mappings. ▪ Data sets. ▪ Data series. ▪ Items added to data series. • Methods for data sets. Methods are automatically available for all data sets. • Rule model properties for data series. If the data series contains one or more items that collect more than one value, then the rule model properties for data series appear so you can specify the value to use.
Methods (on page 128)	A method is a block of code that is called by a rule and that is used to manipulate data. You can use any method, including: • Math methods. • Methods for repeating study objects. • Data set methods.

Mappings in rules

About data mappings

Study objects and properties that are related to mappings are called data mappings. You add data mappings to rule expressions in the Rule Wizard using the dynamic expression prompts in the Expression tab or by dragging them from the Expression tab > Data Mappings tab. The Data Mappings tab lists:

- RefNames of the data mappings, data sets, and data series in the study or library.
- Rule model properties for data series.

  A data series has the properties of the item that is mapped to it. If a data series contains an item that collects more than one value, the rule model properties for repeating study objects appear so you can access an array of all of the values of the item.
- Methods for data sets.

  A method appears if you select the corresponding standard data dimension of the data set. You can use data set methods to return a subset of the data in the data set.
- Study events, forms, sections, and items that are mapped to each data set.

  Study objects appear if you select the corresponding standard data dimension of the data set. The properties of the study objects are used as parameters of data set methods.

Managing data mappings

You manage data mappings in the Project Explorer, where you create mappings, data sets, and data series and add items to data series. For more information, see *Data Mappings tab* (on page 114).

Mappings and data-entry rules

Adding items to a data series in a mapping allows you to:

- Use vector arithmetic to examine a range of values.

  When an item appears on multiple forms or is part of a repeating form or repeating study event, an array of values is collected for the item.
- Use the items in any data-entry rule in the study, regardless of the level at which the rule is created or the scope of the study object.

  To reuse the data-entry rule in another study, all items must be in the mapping in the other study.

Conversion to different units in rules

In the InForm application, when the base unit of an item is in one unit, such as kilograms, but the item allows users to enter different units, such as pounds, the value for the item is stored in the database in two ways—as the entered value (pounds) and the normalized value (pounds converted to kilograms).

For example, if a user types 154 and selects pounds, the value is stored as 154 pounds (for the entered value) and 70 kilograms (for the normalized value). The conversion process from the entered unit to the base unit is called normalization.

Some rules and functions require specific units for item values. For example, a BMI rule might require a weight value to be in kilograms. If a weight item uses pounds instead of kilograms as the base unit, a conversion from pounds to kilograms has to be done in the rule expression.

If the BMI rule is created on a VitalSigns form with Height, Weight, and BMI items, the rule expression with the conversion information could appear as follows:

(this.Weight.Value * 0.45359) / ((this.Height.Value) * (this.Height.Value))

Alternately, you can define a constant that performs the conversion. For example, you can define a **LbToKg** constant that equals **0.45359** and use the constant in the expression:

(this.Weight.Value * LbtoKg) / ((this.Height.Value) * (this.Height.Value))

Rules with multiple conditions

You can create a rule with multiple conditions using the **? :** operator. The operator allows you to mimic an if/then/else control structure and present multiple conditions that must all be true (or false) for a rule to pass or fail.

For example, a rule is used to determine the following information:

- Is the subject pregnant? (True or False)
- Has a severe or life-threatening adverse event occurred? (True or False)

If both are false, the rule passes. If one is true and the other is false, the rule passes. If both are true, the rule checks that the correct termination code was entered when the subject was terminated from the study.

- If the correct termination code was entered, the rule passes.
- If an incorrect termination code was entered, the rule fails, and a query is issued, indicating that the correct termination code must be selected.

The rule checks data on the following forms:

- **Demographics form**—To determine if the subject is pregnant.
- **AE (Adverse Events) form**—To determine if the adverse event that occurred was severe or life threatening.
- **Termination form**—To check the termination code that was used when the subject was terminated from the study.

This rule is created at the study design level because the three forms are in the scope of the study

design. The query is created on the Termination form.

Example of a rule with multiple conditions

The rule could use the following syntax.

evaluate on Form Submission

value = this.InitialVisit.Pregnancy.Pregnant.Value == this.InitialVisit.Pregnancy.Pregnant.YesNoCodes.YesNoCode1 && (this.AECM.AdverseEvents.Current().Severity.Value == this.AECM.AdverseEvents.Current().Severity.SeverityCodes.SeverityCode3 || this.AECM.AdverseEvents.Current().Severity.Value == this.AECM.AdverseEvents.Current().Severity.SeverityCodes.SeverityCode4) ? this.Termination.TerminationForm.TerminationReason.Value == this.Termination.TerminationForm.TerminationReason.TerminationCodes.TerminationCodePREGSEV : true

when value is false

issue query on this.Termination.TerminationForm.TerminationReason: If the patient is pregnant and a severe or life-threatening AE occurs, Termination Reason must be filled out with a value of "Severe AE during Pregnancy"

In the previous example, the values collected for items (such as the Pregnancy item) are compared to the codelist item names, not the actual values of the codelist items. You can simplify the rule expression by providing the codelist item values in place of the codelist item names (shown in the following example).

The following codes are used in the study:

- In the YesNo codelist:
  - Yes = 1.
- In the SeverityCodes codelist:
  - Severe = 3.
  - Life-Threatening = 4.
- In the TerminationCodes codelist:
  - Severe AE during Pregnancy = PREGSEV

Example of a simplified rule expression in a rule with multiple conditions

Using these values, you could simplify the rule syntax:

evaluate on Form Submission

value = this.InitialVisit.Pregnancy.Pregnant.Value == 1 &&
(this.AECM.AdverseEvents.Current().Severity.Value == 3 ||
this.AECM.AdverseEvents.Current().Severity.Value == 4) ?
this.Termination.TerminationForm.TerminationReason.Value == PREGSEV : true

when value is false

issue query on this.Termination.TerminationForm.TerminationReason: If the patient is pregnant and a severe or life-threatening AE occurs, Termination Reason must be filled out with a value of "Severe AE during Pregnancy"

Dynamic prompts in the Expression workspace

When you create an expression, you can type directly in the Expression workspace, and you can drag rule model objects from the tabs at the right side of the windows where you create a data-entry rule, workflow rule, or global condition.

When you type in the Expression workspace, prompts appear dynamically as you type, listing the rule model objects that are available for you to use based on the scope of the rule and the content of the expression. As an alternative to dragging rule model objects from the tabs, you can select the objects from the prompts.

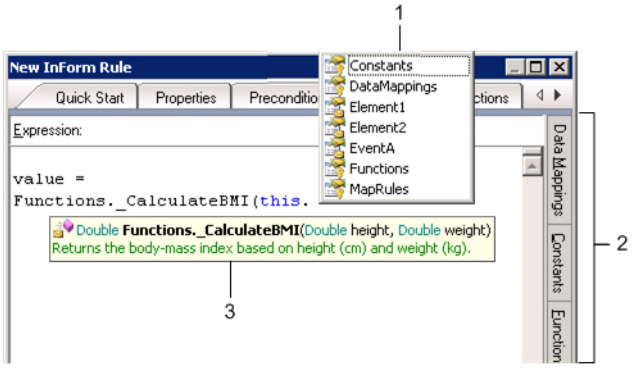
A dynamic expression prompt appears when you type a period. The prompt contains a list of rule model objects that you can select.

- To use the value of a study object or rule model property, type **this.**, and select the study object or property from the dynamic expression prompt.
- To use a function, type **Functions.**, and select the function from the dynamic expression prompt.
- To use a constant, type **Constants.**, and select the constant from the dynamic expression prompt.
- To use a data mapping study object or a rule model property, type **DataMappings.**, and select the study object or rule model property from the dynamic expression prompt.

When you select a rule model object that requires parameter values (for example, a function or method), an open parenthesis mark follows the name of the rule model object, and a tooltip indicates the parameters and their data types. The tooltip also appears when you point to the name of the rule model object.

The following illustration shows a dynamic expression prompt and tooltip that appear when you create a rule expression with a function on a study event that includes several forms. The Rule Wizard tabs containing the same options are visible on the right.

Dynamic expression prompt and tooltip



1—Dynamic expression prompt. Constants, functions, data mappings, and child study objects are available for selection.

2—Standard Rule Wizard tabs.

3—Tooltip showing format and data types of required parameters for the function.

Selecting a rule model object from a dynamic expression prompt

When a dynamic expression prompt appears:

- Double-click a rule model object.

  or

  Select a rule model object, and press the **Tab** or **Enter** key.

> **Note:** You can navigate the list in the dynamic expression prompt with arrow keys or by typing the first letter of a rule model object.

Rule templates

To simplify the process of creating rules, experienced programmers can create rule templates to define the most common rules. Non-programmers can then use the rule templates to create rules. A rule created using a rule template is called an intrinsic rule. Intrinsic rules are typically used for simple, item-level checks.

A rule template is a function that is defined on a study object, study object template, or study object type and can be used as the expression clause of a rule.

A rule template defines only the expression clause of an intrinsic rule. You must supply the precondition clause, action clause, and parameters for an intrinsic rule in the Rule Wizard to form a complete rule.

Rule templates are especially useful when they are part of a study object template or type because the rule template is part of all study objects created from the template or type. For example, a rule template that tests the value of an item against a minimum or maximum value can be part of the integer item type. Then, when you create an integer item, the rule template is part of the item. A user who works on the rule can create an intrinsic rule based on the rule template.

Rule Templates tab

Rule templates are created and managed using the Rule Templates tab. The Rule Templates tab displays a list of all rule templates that are attached to the study object and a toolbar for adding, modifying, or deleting a rule template.

The Rule Templates tab appears in the Form Editor, Section Editor, and Item Editor. The tab provides a summary of all the rule templates that are attached to the selected study object. You can create rule templates for:

- Forms
- Form templates
- Sections
- Section templates
- Items
- Item templates
- Item types

Creating, modifying, and deleting a rule template

You cannot delete a *rule template* (on page 11) if an existing rule is based on the template.

To create a rule template:

1 In the Project Explorer, select a form, section, or item.

The editor for the study object appears in the workspace.

2 Select the **Rule Templates** tab.

3 Click **Add**.

The New Rule Template dialog box appears.

4 Fill in the fields in the dialog box.

5 Click **OK**.

To modify a rule template:

1 In the Project Explorer, select a form, section, or item.

The editor for the study object appears in the workspace.

2 Select the **Rule Templates** tab.

3 In the grid, select the rule template to modify.

4 Click **Edit**.

The Edit Rule Template dialog box appears.

5 Modify the template as necessary.

To delete a rule template:

1 In the Project Explorer, select a form, section, or item.

The editor for the study object appears in the workspace.

2 Select the **Rule Templates** tab.

3 In the grid, select the rule template to delete.

4 Click **Delete**.

The template is removed from the form or item.

For more information, see **Option descriptions for the New Rule Template dialog box** (on page 154).

Default rule templates

_CheckTextLength

Characteristic	Description
Availability	Text items
Description	Checks whether the value entered in the item is less than the maximum length. The maximum length is 255 by default, but you can specify any value when you create a rule based on the template.
Returns	• **True**—The value entered contains fewer than 255 characters. • **False**—The value entered contains 255 characters or more.
Display text	this.Value.Length < MaxLength

_DateTimeRangeCheck

Characteristic	Description
Availability	Date time items
Description	Checks whether a date falls within a specified range of another date. When you create a rule based on the template, you specify the two date time items, the date part to compare (by default, days), and the minimum and maximum range.
Returns	• **True**—Date1 is within the specified range of Date2. • **False**—Date1 is not within the specified range of Date2.
Display text	_CompareDatesWithRange(date1,date2,datePart,rangeMin,rangeMax)

_FutureDateCheck

Characteristic	Description
Availability	Date time items
Description	Checks whether the specified date and time is in the future, as compared to the site date and time. Site information is based upon the time zone of the data-entry site rather than the time zone of the InForm server.
Returns	• **True**—The entered date and time are in the future, compared to the site date and time. • **False**—The entered date and time is empty or is in the past, compared to the site date and time.
Display text	!Empty && _CompareDates(Value, GetSiteTime()) > 0

_PartialCompletenessCheck

Characteristic	Description
Availability	Compound and blood pressure items
Description	Checks whether child items within a compound item are partially complete. A partially complete compound item consists of one or more empty child items and one or more complete child items.
Returns	• **True**—Child items within a compound item are partially complete (one or more child items are empty, and one or more child items are complete). • **False**—Child items within a compound item are all either empty or complete.
Display text	!this.Empty && !this.Complete

_RangeCheck

Characteristic	Description
Availability	Integer and float items
Description	Checks whether the value for an integer or float item is greater than a minimum value and less than a maximum value. By default, the minimum value is: • 0 for an integer. • 0.0 for a float. By default, the maximum value is: • 100 for an integer. • 100.0 for a float item. You can change these values when you create a rule based on the template.
Returns	• **True**—The value of the item is greater than the minimum value and less than the maximum value. • **False**—The value of the item is less than the minimum value or greater than the maximum value.
Display text	Value must be between {MinValue} and {MaxValue}

_RangeCheckInclusive

Characteristic	Description
Availability	Integer and float items
Description	Checks whether the value of a float or integer item is greater than or equal to a minimum value and less than or equal to a maximum value. By default, the minimum value is: • 0 for an integer. • 0.0 for a float. By default, the maximum value is: • 100 for an integer. • 100.0 for a float item. You can change these values when you create a rule based on the template.

Characteristic	Description
Returns	• **True**—The value of the item is greater than or equal to the minimum value and less than or equal to the maximum value. • **False**—The value of the item is less than the minimum value or greater than the maximum value.
Display text	Value must be greater than or equal to {MinValue}, and less than or equal to {MaxValue}

ValidBPCheck

Characteristic	Description
Availability	Blood pressure items
Description	Checks whether the entered systolic value is greater than the entered diastolic value.
Returns	• **True**—The systolic value is greater than the diastolic value. • **False**—The systolic value is less than the diastolic value.
Display text	this.SystolicVariable.Value > this.DiastolicVariable.Value

CHAPTER 2

Data

In this chapter

Data tab

A rule expression can include data from a study, including:

- The values of study objects.
- The values of the rule model properties of study objects.

  Data collected in the InForm application has properties, such as Empty and Required, that you can select. These and other properties, collectively called rule model properties, are visible in the Rule Wizard and can be used in rule expressions.
- Review states and review stages.
- A method used to identify an instance of a repeating study object.

Using data in rules and rule templates

To add data to rule expressions, use one of the following:

- **For workflow rules**—Workflow Expression Editor dialog box.
- **For rule templates**—Edit Rule Template dialog box > Definition tab.
- **For data-entry rules**—Rule Wizard Expression tab > Data tab.
- **For global conditions**—Edit Global Conditions dialog box > Data tab.

Note: Rule expressions use RefNames, not titles. Therefore, RefNames appear in the Rule Wizard and in the rule expression.

In the Data tab (in the Expression tab of the Rule Wizard), the RefName of the study object to which you are adding a rule always appears at the top of the tree. The information that appears below the study object depends on whether **Show all** is selected.

- **Show all not selected.**

  The RefNames of all children of the study object are listed below the study object. Only study objects within the scope of the rule are listed.

  You can use the values of any of the study objects in the rule expression.

- **Show all selected.**

  The following information appears:

  - RefNames of all children of the study object.

    Only study objects within the scope of the rule are listed. You use the values of the study objects and properties in the rule expression.

  - Rule model properties of the children of the study object. Rule model properties are available for study events, forms, sections, and items.
  - Methods for repeating study objects.

To use the value of a study object or the value of a rule model property in a rule expression:

- Drag the study object or property from the Data tab to the Expression workspace.

Icons used on the Data tab

The icons that appear depend on the level on which a rule is created and whether **Show all** is selected.

Icon	Description
	Study design.
	Study element.
	Study event.
	Form.
	• DateTime item. or • Rule model property with a DateTime return type.
	• Float item. or • Rule model property with a Float return type.
	• Integer item. or • Rule model property with an Integer return type.
	• Text item. or • Rule model property with a Text return type.
	Compound item.
	Method.
	Used for some rule model properties for items.
	Rule model property with a Boolean return type.
	[Name of codelist] rule model property.
	[Name of codelist item] rule model property.

Rule model properties for study events, forms, and sections

You add rule model properties to an expression in the Rule Wizard in the Expression tab > Data tab.

Note: The types for rule model properties are .NET base types.

Note: All arrays start at 0.

[Name of review stage]

Characteristic	Description
Icon	
Availability	Appears for forms when at least one review state has been defined in the study.
Description	Review stage in a custom review state. You can get the review stage of a form using the GetReviewState method. You can check for the review stage of a form using the HasReviewState method.

[Name of review state]

Characteristic	Description
Icon	
Availability	Appears for forms. This property is a container for the stages defined for a custom review stage and cannot be dragged to the Expression workspace. You can drag the named review stages that appear below the property.

CurrentIndex

Characteristic	Description
Icon	
Availability	Repeating study events, repeating forms, and repeating sections. **Note:** To view this property, you must select the parent study object. For example, to view the property on a study event, you must select its parent (either a study element or a study design).
Return type	Int32 (for Central Designer Integer and YesNo types).
Description	The index of the current study events, form, or section. The index starts at 0.

Characteristic	Description
Purpose	Use this property to determine the section, form, or study event for which someone is currently entering or modifying data. For example, you might want to perform a rule action for the first form for which data was entered and another rule action for subsequent instances of the form.

Count

Characteristic	Description
Icon	
Availability	Repeating study events, repeating forms, and repeating sections. **Note:** To view this property, you must select the parent study object. For example, to view the property on a study event, you must select its parent (either a study element or a study design).
Return type	Int32 (for Central Designer Integer and YesNo types).
Description	The current number of instances of the repeating study event, form, or section.
Purpose	Use this property to determine the number of sections, forms, or study events in a study. For example, you might want to know the number of adverse events that exist for a subject.

HasData

Characteristic	Description
Icon	
Availability	Non-repeating forms and sections.
Return type	Boolean.
Description	True or False. When True, a value for the item has been provided.
Purpose	Use this property to determine whether a form has data on it. For example, if an Adverse Events form indicates that a subject was hospitalized, you could check whether the hospitalization form has been filled out yet. `value = this.AE.Current().itmHospitalized == 'Y' ? this.Hospitalization.HasData : true when value is false issue query on this.AE.Current().itmHospitalized: If patient was hospitalized, Hospitalization form must be filled out.`
Notes	**Note:** The functionality available with HasData is now available with HasState. If you are using HasData, consider converting it to HasState(Constants.FormStates.HasData).

IsDeleted

Characteristic	Description
Icon	
Availability	Repeating forms and sections.
Return type	Boolean.
Description	True or False. When True, the form or section has been deleted.
Purpose	Use this property to determine whether an itemset (in the Central Designer application, a section) or a form has been deleted in the InForm application.
Notes	**Note:** The functionality available with IsDeleted is now available with HasState. If you are using HasData, consider converting it to HasState(Constants.FormStates.Deleted).

RelatedData[]

Characteristic	Description
Icon	
Availability	Repeating, associated form.
Return type	• Form instances for the associated form. or • An array of item values from associated forms.
Description	Returns instances of associated forms, sorted by form index, and allows access to arrays of item values from the associated forms.
Example	If the AE and CM forms are associated, and you are creating a rule on a study event that contains both of them: • To return an array of associated AE forms: **this.CM.Current().RelatedData** • To return an array of associated CM forms: **this.AE.Current().RelatedData** • To return the value of the Verbatim item on the first AE form: **this.CM.Current().RelatedData[0].Verbatim.Value** • To return an array of values for the drugName item from all instances of the CM form that are associated with the current instance of the AE form: **this.AE.Current().RelatedData.drugName.Values**

Characteristic	Description
Notes	Trigger dependencies are created in the InForm application for a data-entry rule using associated forms. • When the expression of a data-entry rule references the RelatedData rule model property, a trigger dependency is created for both the item that is explicitly referenced and the item that is referenced through the RelatedData property. The rule runs whenever either of the associated forms is submitted. • Additionally, when you associate or unassociate two forms in the InForm application, a trigger dependency is created, so the rules are run.

ReviewStates

Characteristic	Description
Icon	
Availability	Appears for forms when at least one custom review state has been defined in the study. This property is a container for custom review states and cannot be dragged to the Expression workspace. You can drag the named review states that appear below the property.

Rule model properties for items

Rule model properties are different from standard and custom properties, which are properties of the study design and are visible in the Properties Browser.

You add rule model properties to an expression in the Rule Wizard in the Expression tab > Data tab. The properties in this section are available for items of all data types, except where noted.

Note: The types for rule model properties are .NET base types.

Note: All arrays start at 0.

[Alias or code of codelist item]

Characteristic	Description
Icon	
Availability	Items with a codelist.
Return type	One of the following: • String (for Central Designer string types). • Double (for Central Designer float types). • Int32 (for Central Designer Integer and YesNo types).
Description	Codelist item in the codelist that is associated with the item. When you use a codelist item in a rule expression, the value of the codelist item is used.
Purpose	The alias of a codelist item is useful because you can use the alias in place of the code in the codelist. The Rule Wizard allows you to drag the codelist item as a replacement for the code, so you do not have to memorize it or close the Rule Wizard to determine the value.

[Name of codelist]

Characteristic	Description
Icon	
Availability	Appears for items that have a codelist. This property is a container for codelist items and cannot be dragged to the Expression workspace. You can drag the codelists that appear below the property.

AllObjects[]

Characteristic	Description
Icon	
Availability	An item that is a child of one of the following: • Form that is part of a repeating study event. • Repeating form. • Repeating section.
Return type	object[]
Description	Returns an array of values from item instances that are part of a repeating study event, repeating form, or repeating section.
Included in the array	• Values of undeleted items. An item is undeleted if someone has provided a value for it, and the form or section that contains the item has not been deleted. • Values of deleted items. An item is deleted if the form or section that contains it is deleted from a study. • Values of empty items. An item is empty if the form has been created but a value has not been provided for the item.
Example	A study contains the following repeating forms, which contain the AEDate item: • **First instance of the form**—Someone has entered a value for the AEDate item. • **Second instance**—Someone has filled in values for other items but not the AEDate item (the item is empty). • **Third instance**—Someone entered a value for the AEDate item and deleted the form (the item is deleted). • **Fourth instance**—Someone has entered a value for the AEDate item. When you use this property, the following instances are returned: • First instance • Second instance • Third instance • Fourth instance

Characteristic	Description
Notes	**Note:** Values[], AllValues[], Objects[], and AllObjects[] all return an array of data. Consider the following scenarios when deciding upon the property to use: • To include deleted values, use either AllValues[] or AllObjects[]. • To include null values, use Objects[] or AllObjects[]. Additionally, consider that the following CurrentIndex properties are also available: • ***CurrentAllObjectsIndex*** (on page 29). • ***CurrentAllValuesIndex*** (on page 29). • ***CurrentObjectsIndex*** (on page 29). • ***CurrentValuesIndex*** (on page 30).

AllValues[]

Characteristic	Description
Icon	
Availability	An item that is a child of one of the following: • Form that is part of a repeating study event. • Repeating form. • Repeating section.
Return type	• **If an integer type**—Int[]. • **If a float type**—Double[]. • **If a text type**—String[]. • **If a date time type**—PFDateTime[].
Description	Returns an array of values from item instances that are part of a repeating study event, repeating form, or repeating section.
Included in the array	• Values of undeleted items. An item is undeleted if someone has provided a value for it, and the form or section that contains the item has not been deleted. • Values of deleted items. An item is deleted if the form or section that contains it is deleted from a study.
Not included in the array	Values of empty items. An item is empty if the form has been created but a value has not been provided for the item.

Characteristic	Description
Example	A study contains the following repeating forms, which contain the AEDate item: • **First instance of the form**—Someone has entered a value for the AEDate item. • **Second instance**—Someone has filled in values for other items but not the AEDate item (the item is empty). • **Third instance**—Someone entered a value for the AEDate item and deleted the form (the item is deleted). • **Fourth instance**—Someone has entered a value for the AEDate item. When you use this property, the following instances are returned: • First instance • Third instance • Fourth instance
Notes	**Note:** Values[], AllValues[], Objects[], and AllObjects[] all return an array of data. Consider the following scenarios when deciding upon the property to use: • To include deleted values, use either AllValues[] or AllObjects[]. • To include null values, use Objects[] or AllObjects[]. Additionally, consider that the following CurrentIndex properties are also available: • *CurrentAllObjectsIndex* (on page 29). • *CurrentAllValuesIndex* (on page 29). • *CurrentObjectsIndex* (on page 29). • *CurrentValuesIndex* (on page 30).

Completed

Characteristic	Description
Icon	
Availability	Compound items.
Return type	Boolean.
Description	• **True**—All child items of the conditional item are not empty. • **False**—At least one child item of the conditional item is empty. **Note:** Items that are conditional on child items of the compound item are not evaluated.

Characteristic	Description
Purpose	This property allows you to simplify the expressions of data-entry rules that check to see if controls that have been started are completed. Typically these expressions also use the Empty rule model property.

CurrentAllObjectsIndex

Characteristic	Description
Icon	
Availability	An item that is a child of one of the following: • Form that is part of a repeating study event. • Repeating form. • Repeating section.
Return type	Int32 (for Central Designer Integer and YesNo types).
Description	Returns the current position in the array that is returned by the AllObjects[] rule model property. The index is 0 based.

CurrentAllValuesIndex

Characteristic	Description
Icon	
Availability	An item that is a child of one of the following: • Form that is part of a repeating study event. • Repeating form. • Repeating section.
Return type	Int32 (for Central Designer Integer and YesNo types).
Description	Returns the current position in the array that is returned by the AllValues[] rule model property. The index is 0 based.

CurrentObjectsIndex

Characteristic	Description
Icon	
Availability	An item that is a child of one of the following: • Form that is part of a repeating study event. • Repeating form. • Repeating section.
Return type	Int32 (for Central Designer Integer and YesNo types).
Description	Returns the current position in the array that is returned by the Objects[] rule model property. The index is 0 based.

CurrentValuesIndex

Characteristic	Description
Icon	
Availability	An item that is a child of one of the following: • Form that is part of a repeating study event. • Repeating form. • Repeating section.
Return type	Int32 (for Central Designer Integer and YesNo types).
Description	Returns the current position in the array that is returned by the Values[] rule model property. The index is 0 based.

Empty

Characteristic	Description
Icon	
Availability	All items.
Return type	Boolean.

Characteristic	Description
Description	True or False. When True, a value for the item has not been provided. For compound items, returns True only when all child items have no values. **Note:** Items that are conditional on child items of a compound item are not evaluated.

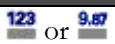
Purpose	Use this property to determine whether a user has entered data in a field and then take another action. For example, if an adverse event is reported as serious, its relation to the study drug cannot be left blank.
	`value = this.itmSerious == 'Y' ? this.itmRelated.Empty : false`
	`when value is true`
	`  issue query on this.itmRelated:`
	`  Relation to study drug cannot be blank for a serious AE`

EnteredUnit

Characteristic	Description
Icon	
Availability	An item with a base unit selected on the Design tab for the item.
Return type	String (for Central Designer string types).
Description	Returns the localized name of the unit, or null if a unit was not selected. For example, you can use this property in query or email text.

Characteristic	Description
Purpose	If you allow InForm users to choose between metric and imperial units but want to require that the user is consistent within a form, use this property to check that the values are from the same measurement system. Consider that values entered in the InForm application are normalized to their base unit. For example, if kg is the base unit, a value of 150 lbs. is stored in the database as 68.0388555 kg. Some unit conversions are not absolute, which can make writing rules difficult. For example, converting mmol/L requires knowledge of a chemical's molecular weight. • To convert mmol/L of glucose to mg/dL, you multiply by 18. • To convert mmol/L of LDL cholesterol to mg/dL, you multiply by 39. You have the following options for recording this information: • If you use a single item, you must create separate unit definitions for each chemical that is measured. • If you use a compound item, with one child to store the value and one child to store the unit, you must perform a conversion to normalize the values for reporting or CDD purposes. • If you use EnteredValue and EnteredUnit, you can store the true normalized value in a hidden field, which you can use for reporting or CDD. The following rule example uses EnteredUnit and EnteredValue to convert glucose from mmol/L to mg/dL, and stores that value in the itmNormalizedGlucose item. Consider that a conversion is not done if another unit is chosen; if the base unit is selected, EnteredValue and Value are the same. `value = this.itmGlucose.EnteredUnit == "mmol/L" ? this.itmGlucose.EnteredValue * 18 : this.itmGlucose.Value always set this.itmNormalizedGlucose = value` You can write a similar rule to convert cholesterol, triglycerides, and other measurements. Additionally, you can replace numerical values, such as 18 and 39, with constants such as Constants.Conversions.Glucose and Constants.Conversions.Cholesterol. You define the constants in the study or a library.

EnteredValue

Characteristic	Description
Icon	 or
Availability	An item with a base unit selected on the Design tab for the item.

Characteristic	Description
Return type	One of the following: • Int32 (for Central Designer Integer and YesNo types). • Double (for Central Designer float types).
Description	Returns the entered value for an item for which a base unit was selected.
Purpose	Some unit conversions are not absolute, which can make writing rules difficult. For example, converting mmol/L requires knowledge of a chemical's molecular weight. • To convert mmol/L of glucose to mg/dL, you multiply by 18. • To convert mmol/L of LDL cholesterol to mg/dL, you multiply by 39. You have the following options for recording this information: • If you use a single item, you must create separate unit definitions for each chemical that is measured. • If you use a compound item, with one child to store the value and one child to store the unit, you must perform a conversion to normalize the values for reporting or CDD purposes. • If you use EnteredValue and EnteredUnit, you can store the true normalized value in a hidden field, which you can use for reporting or CDD. The following rule example uses EnteredUnit and EnteredValue to convert glucose from mmol/L to mg/dL, and stores that value in the itmNormalizedGlucose item. Consider that a conversion is not done if another unit is chosen; if the base unit is selected, EnteredValue and Value are the same. `value = this.itmGlucose.EnteredUnit == "mmol/L" ? this.itmGlucose.EnteredValue * 18 : this.itmGlucose.Value always set this.itmNormalizedGlucose = value` You can write a similar rule to convert cholesterol, triglycerides, and other measurements. Additionally, you can replace numerical values, such as 18 and 39, with constants such as Constants.Conversions.Glucose and Constants.Conversions.Cholesterol. You define the constants in the study or a library.

Objects[]

Characteristic	Description
Icon	

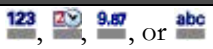
Characteristic	Description
Availability	An item that is a child of one of the following: • Form that is part of a repeating study event. • Repeating form. • Repeating section.
Return type	object[]
Description	Returns an array of values from item instances that are part of a repeating study event, repeating form, or repeating section.
Included in the array	• Values of undeleted items. An item is undeleted if someone has provided a value for it, and the form or section that contains the item has not been deleted. • Values of empty items. An item is empty if the form has been created but a value has not been provided for the item.
Not included in the array	Values of deleted items. An item is deleted if the form or section that contains it is deleted from a study.
Example	A study contains the following repeating forms, which contain the AEDate item: • **First instance of the form**—Someone has entered a value for the AEDate item. • **Second instance**—Someone has filled in values for other items but not the AEDate item (the item is empty). • **Third instance**—Someone entered a value for the AEDate item and deleted the form (the item is deleted). • **Fourth instance**—Someone has entered a value for the AEDate item. When you use this property, the following instances are returned: • First instance • Second instance • Fourth instance

Characteristic	Description
Notes	**Note:** Values[], AllValues[], Objects[], and AllObjects[] all return an array of data. Consider the following scenarios when deciding upon the property to use: • To include deleted values, use either AllValues[] or AllObjects[]. • To include null values, use Objects[] or AllObjects[]. Additionally, consider that the following CurrentIndex properties are also available: • ***CurrentAllObjectsIndex*** (on page 29). • ***CurrentAllValuesIndex*** (on page 29). • ***CurrentObjectsIndex*** (on page 29). • ***CurrentValuesIndex*** (on page 30).

Selected[]

Characteristic	Description
Icon	
Availability	Available for items with multi-select codelists.
Return type	Boolean
Description	For the codelist item that is specified in the brackets, returns a value that indicates if the checkbox for the codelist item is selected. You can also use this property to select or deselect the checkbox for a codelist item.
Purpose	Use this property to determine whether a checkbox has been selected. For example, to disqualify subjects who are using aspirin, you can create a list of common OTC drugs with a checkbox next to each. `value = this.itmOTCMeds.Selected[this.itmOTCMeds.clOTCMeds.Aspirin] when value is true issue query on this.itmOTCMeds: Subjects on an aspirin regimen are not eligible for the study.` You can also use the property to programmatically select items. To select an item, use a SetValue action, setting the value to True for a selected checkbox or False for a deselected checkbox.

Value

Characteristic	Description
Icon	 , or
Availability	All items except compound items.
Return type	One of the following: • String (for Central Designer string types). • DateTime (for Central Designer date time types). • Double (for Central Designer float types). • Int32 (for Central Designer Integer and YesNo types).
Description	Returns the value of the item.
Purpose	Use this property to return the value of an item as it is stored in the database. • If an item has no units, the entered value is returned. • If an item has units, the number that is converted to the base unit is returned. For example, if the base unit for a weight item is kilograms, and the InForm user enters a value of 100 and: • Selects kg, the property returns 100. • Selects lbs, the property returns 45.359237. To determine the value that the InForm user entered into the field, use ***EnteredValue*** (on page 32) and the accompanying ***EnteredUnit*** (on page 31).

Value[]

Characteristic	Description
Icon	
Availability	All items with multi-select codelists.
Return type	• **If an integer type**—Int[]. • **If a float type**—Double[]. • **If a text type**—String[].
Description	Returns an array of values for the item.

Characteristic	Description

Purpose	Use this property to process all of the boxes that are selected in a checkbox group. To use a single checkbox item, use ***Selected[]*** (on page 35).
	
	For example, if you have a codelist with checkboxes and want to make sure that no more than three checkboxes are selected, use this property to return an array of the selected checkboxes, and use the Length property of the array to determine the number of selected checkboxes.
	
	`value = this.itmConcomMeds.Value.Length`
	`when value > 3`
	`  issue query on this.itmConcomMeds:`
	`  Subject should not be on more than three`
	`  concomitant meds at time of enrollment`

ValueLabel

Characteristic	Description
Icon	
Availability	All items with a codelist.
Return type	String (for Central Designer string types).
Description	Returns the text value of the selected radio button or item in a drop-down list. If a value is not selected, the rule stops running.

Purpose	This property prevents you from needing to memorize codes for codelist items and drag codelist item aliases. For example, if you have a Route of Administration drop-down list for medications and you are using the FDA codes, Spinal corresponds to 356. You can write the expression in the following way:
	
	`value = (this.itmRouteOfAdministration.Value == 356)`
	However, this expression requires that you know the value of the code. This property allows you to write the expression in the following way:
	
	`value = (this.itmRouteOfAdministration.ValueLabel == "Spinal")`
	Alternatively, you can perform multiple rule actions:
	
	`value = this.itmRouteOfAdministration.ValueLabel`
	`when value is "Spinal"...`
	`when value is "Oral"...`
	`when value is "Intravenous"...`

Values[]

Characteristic	Description
Icon	

Characteristic	Description
Availability	An item that is a child of one of the following: • Form that is part of a repeating study event. • Repeating form. • Repeating section.
Return type	• **If an integer type**—Int[]. • **If a float type**—Double[]. • **If a text type**—String[]. • **If a date time type**—PFDateTime[].
Description	Returns an array of values from item instances that are part of a repeating study event, repeating form, or repeating section.
Included in the array	Values of undeleted items. An item is undeleted if someone has provided a value for it, and the form or section that contains the item has not been deleted.
Not included in the array	• Values of deleted items. An item is deleted if the form or section that contains it is deleted from a study. • Values of empty items. An item is empty if the form has been created but a value has not been provided for the item.
Example	A study contains the following repeating forms, which contain the AEDate item: • **First instance of the form**—Someone has entered a value for the AEDate item. • **Second instance**—Someone has filled in values for other items but not the AEDate item (the item is empty). • **Third instance**—Someone entered a value for the AEDate item and deleted the form (the item is deleted). • **Fourth instance**—Someone has entered a value for the AEDate item. When you use this property, the following instances are returned: • First instance • Fourth instance

Characteristic	Description
Notes	**Note:** Values[], AllValues[], Objects[], and AllObjects[] all return an array of data. Consider the following scenarios when deciding upon the property to use: • To include deleted values, use either AllValues[] or AllObjects[]. • To include null values, use Objects[] or AllObjects[]. Additionally, consider that the following CurrentIndex properties are also available: • ***CurrentAllObjectsIndex*** (on page 29). • ***CurrentAllValuesIndex*** (on page 29). • ***CurrentObjectsIndex*** (on page 29). • ***CurrentValuesIndex*** (on page 30).

Rule model properties for DateTime items

Year

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime items. This property appears only when **Allow** is selected for the **Year** property of the item.
Return type	Int32 (for Central Designer Integer and YesNo types).
Description	Value of the **Year** component of the DateTime item.

YearEmpty

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime items. This property appears only when **Required** is not selected for the **Year** property of the item.
Return type	Boolean.
Description	True or False. When True, a value for the part of the date time item has not been provided.

YearUnknown

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime. This property appears only when **Allow unknown** is selected for the Year property of the item.
Return type	Boolean.
Description	True or False. When True, the Unknown value was selected for the date time part.

Month

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime items. This property appears only when **Allow** is selected for the **Month** property of the item.
Return type	Int32 (for Central Designer Integer and YesNo types).
Description	Value of the **Month** component of the DateTime item.

MonthEmpty

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime items. This property appears only when **Required** is not selected for the **Month** property of the item.
Return type	Boolean.
Description	True or False. When True, a value for the part of the date time item has not been provided.

MonthUnknown

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime. This property appears only when **Allow unknown** is selected for the **Month** property of the item.
Return type	Boolean.
Description	True or False. When True, the Unknown value was selected for the date time part.

Day

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime items. This property appears only when **Allow** is selected for the **Day** property of the item.
Return type	Int32 (for Central Designer Integer and YesNo types).
Description	Value of the **Day** component of the DateTime item.

DayEmpty

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime items. This property appears only when **Required** is not selected for the **Day** property of the item.
Return type	Boolean.
Description	True or False. When True, a value for the part of the date time item has not been provided.

DayUnknown

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime. This property appears only when **Allow unknown** is selected for the **Day** property of the item.
Return type	Boolean.
Description	True or False. When True, the Unknown value was selected for the date time part.

Hour

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime items. This property appears only when **Allow** is selected for the **Hour** property of the item.
Return type	Int32 (for Central Designer Integer and YesNo types).
Description	Value of the **Hour** component of the DateTime item.

HourEmpty

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime items. This property appears only when **Required** is not selected for the **Hour** property of the item.
Return type	Boolean.
Description	True or False. When True, a value for the part of the date time item has not been provided.

HourUnknown

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime. This property appears only when **Allow unknown** is selected for the **Hour** property of the item.
Return type	Boolean.
Description	True or False. When True, the Unknown value was selected for the date time part.

Minute

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime items. This property appears only when **Allow** is selected for the **Minute** property of the item.
Return type	Int32 (for Central Designer Integer and YesNo types).
Description	Value of the **Minute** component of the DateTime item.

MinuteEmpty

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime items. This property appears only when **Required** is not selected for the **Minute** property of the item.
Return type	Boolean.
Description	True or False. When True, a value for the part of the date time item has not been provided.

MinuteUnknown

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime. This property appears only when **Allow unknown** is selected for the **Minute** property of the item.
Return type	Boolean.
Description	True or False. When True, the Unknown value was selected for the date time part.

Second

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime items. This property appears only when **Allow** is selected for the **Second** property of the item.
Return type	Int32 (for Central Designer Integer and YesNo types).
Description	Value of the **Second** component of the DateTime item.

SecondEmpty

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime items. This property appears only when **Required** is not selected for the **Second** property of the item.
Return type	Boolean.
Description	True or False. When True, a value for the part of the date time item has not been provided.

SecondUnknown

Characteristic	Description
Icon	
Availability	Available under the **Value** rule model property for DateTime. This property appears only when **Allow unknown** is selected for the **Second** property of the item.
Return type	Boolean.
Description	True or False. When True, the Unknown value was selected for the date time part.

Methods for repeating study objects

You use methods for rules that refer to repeating study events, repeating forms, and repeating sections. The data collected for repeating study objects comprise an array. To include information about a repeating study object in a rule, you must use a method to specify the instance in the array.

Methods for repeating study objects appear in the Rule Wizard in the Expression tab > Data tab.

To use a method in a rule expression:

1 Double-click the method.

If the method has one or more parameters, the Invoke Function dialog box appears.

2 Drag a study object from the **References** tab to the **Values** field to define a value for each parameter.

[] [Indexer]

Characteristic	Description
Icon	
Availability	Repeating study events, repeating forms, and repeating sections.
Return type and description	Indexer method. Returns an instance of a repeating study object.
Syntax	**NameOfStudyObject[index]**
Parameters	• **Parameter—index**. • **Definition**—An integer from 0 to the length of the array minus 1 (Count - 1). • **Data type**—Integer.
Example	If form RefName is **ConMed**, the following example returns the first instance of the repeating form: **ConMed[0]**

Purpose	This method is useful when you are working with an array of values. For example, use the following expression to determine the date on which the first adverse event was entered:
	`this.frmAE[0].itmDateOfOnset.Value`
	To determine the date on which the last adverse event was entered, use the following expression:
	`this.frmAE[this.frmAE.Count - 1].itmDateOfOnset.Value`

Current()

Characteristic	Description
Icon	
Availability	Repeating study events, repeating forms, and repeating sections.
Return type and description	Returns the current instance of the repeating study object.
Syntax	**NameOfStudyObject.Current( )**
Parameters	No parameters.
Example	If form RefName is **ConMed**: **ConMed.Current( )**
Best practices for workflow rules and global conditions	To create a workflow rule or global condition that uses a repeating study object's current instance to determine if the conditions of the rule are satisfied, Oracle recommends that you use Current() only when a single instance exists. When there are multiple instances of the repeating study object, the method cannot determine which instance is current, so the workflow rule or global condition might not work as expected. If there are multiple instances of the repeating study object, consider using an array of the repeating study object, using the IsValueInArray function. In the expression, Oracle recommends checking whether the array has any values before using it for comparison.

Current(Integer)

Characteristic	Description
Icon	
Availability	Repeating study events, repeating forms, and repeating sections.
Return type and description	Allows you to choose the instance relative to the current instance of the repeating study object. This method skips deleted instances.
Syntax	**NameOfStudyObject.Current(relativeIndex)**
Parameters	• **Parameter—relativeIndex**. • **Definition**—(Optional) An integer with a range of (–CurrentIndex) to (Count – CurrentIndex – 1). • **Data type**—Integer.
Example	If form RefName is **ConMed**: **ConMed.Current(-1)**

Characteristic	Description

Purpose	Use Current with the relative index to compare values across repeating study events and forms. For example, to confirm that the DOV for each unexpected visit is after the DOV for the previous visit, use the following expression:
	`value = _CompareDates(this.Current().frmDOV.itmDOV.Value, this.Current(-1).frmDOV.itmDOV.Value)`
	`when value = -1`
	`issue query on this.Current().frmDOV.itmDOV: DOV for this visit must be later than the previous visit.`

GetValue

Appears as one of the following, depending on the type of the study object:

- GetValue(Integer)
- GetValue(Date)
- GetValue(String)
- GetValue(Float)

Characteristic	Description
Icon	
Availability	All items.
Return type and description	**If the Empty property of the item is True**—Returns the replacement value (a parameter of the method). **If the Empty property of the item is False**—Returns the value of the item.
Syntax	(If rule is created on a study design) **StudyEventRefName.FormRefName.ItemRefName.GetValue(replacementValue)** **Note:** If the item is a child of a repeating study object, method information is included in the rule expression to indicate the item instance to which you are referring.
Parameters	• **Parameter**—**replacementValue**. • **Definition**—The replacement value for the item. • **Data type**—Data type of the item.

Characteristic	Description
Example	If: • Study event RefName is **Death** • Form RefName is **DeathForm** • Item RefName is **RelatedToDevice** • Codelist item RefName that you specify for the parameter is **RelatedCode3** **Death.DeathForm.RelatedToDevice.GetValue(Death.DeathForm.RelatedToDevice.RelatedCodes.RelatedCode3)**

Purpose	This method is useful for replacing a null value with a value that is usable for calculations, or for causing a rule to run even if a value has not been entered. For example, a BMI calculation typically does not run until the InForm user enters both a Height and Weight value for a subject. However, you can use this method to store a value of 0 in the BMI field if either Height or Weight is missing.
	Below is a typical rule expression for calculating BMI:
	`value = this.itmHeight.Value != 0 ?`
	`_CalculateBMI(this.itmWeight.Value, this.itmHeight.Value) : 0`
	This rule expression does not calculate the BMI until both values are present.
	Below is the same expression, except it uses GetValue
	`value = this.itmHeight.GetValue(0) == 0 ?`
	`_CalculateBMI(this.itmWeight.GetValue(0),`
	`this.itmHeight.Value) : 0`
	This expression calculates BMI upon submission of the form, regardless of whether the values for Height and Weight were entered.
	In both cases, the expression checks whether Height is 0. If Height is 0, a divide-by-zero error occurs.

Methods for non-repeating study objects

GetReviewStates

Characteristic	Description
Icon	
Availability	All non-deleted forms.
Return type and description	Integer. Returns the review stage that a form is in within a specified review state.
Syntax	**GetReviewState(Name of review state.Name of review stage)**
Parameters	• **Parameter—state**. • **Definition**—Review state of the form. • **Data type**—Integer.
Example	The following example returns the review stage that the form is in within the QA Review review state: **GetReviewState(this.ReviewStates.QAReview)**
Purpose	Use this method to obtain the review stage that a form is in within a specified review state.

HasReviewStates

Characteristic	Description
Icon	
Availability	All non-deleted forms.
Return type and description	Boolean. Returns true if the form is in the review stage specified in the parameter. Review stages appear in the folder for the form.
Syntax	**HasReviewState(Name of review stage)**
Parameters	• **Parameter—stage**. • **Definition**—Review stage of the form. • **Data type**—Integer.
Example	The following example returns True if the form is in the NeedsReview stage of the QA Review review state: **HasReviewState(this.QAReview.NeedsReview)**
Purpose	Use this method to check whether a form is in a particular review stage.

HasState(Integer)

Characteristic	Description
Icon	
Availability	All non-deleted forms.
Return type and description	Boolean. Returns True if the form is in the state that is specified in the parameter. Specify the state using one of the following constants. The constants appear on the Constants tab and have a data type of integer. • **Deleted**—(Available only for repeating forms) The repeating form instance has been deleted. • **Frozen**—The form is frozen. • **HasComment**—The form has a comment at the form level. • **HasData**—The form has data. • **HasMissingData**—The form has missing data. • **HasQueries**—The form has queries. • **Locked**—The form is locked. • **SdvComplete**—The form has been marked source verified. • **SdvPartial**—The form has been partially source verified. • **SdvReady**—The form has been marked ready for source verification. • **Signed**—The form has been signed. • **Skipped**—The form has been marked not completed with a form-level comment. • **Started**—The form was started with patient data, comments, or queries.
Syntax	**HasState(NameOfConstant)**
Parameters	• **Parameter**—**formState**. • **Definition**—State of the form. • **Data type**—Integer.
Example	The following example returns True if the form is locked: **HasState(Constants.FormStates.Locked)**
Purpose	Use this method to return the status of InForm forms.

CHAPTER 3

Functions

In this chapter

Functions tab of the Rule Wizard

A function is a reusable piece of code that extends the behavior of a rule. A function provides part or all of a rule expression. Without functions, a rule can support simple expressions with no control-flow statements or compound statements. Functions make possible or greatly simplify complex calculations or decision-making functionality in rule expressions. Functions are global. They do not have a scope and can be used in any rule created in a study or library as long as they are part of the study or library.

Functions allow users with different levels of programming experience to participate in rule-building:

- Experienced programmers with the ability to program in a .NET language (for example, C#) and create .NET assemblies build complex function code outside of the Central Designer application and import it into libraries or studies.
- Users who are not familiar with programming languages can create rules or rule templates that use functions.

The Central Designer application includes predefined functions (for example, several functions handle checking the components of date time fields).

You can use functions in a library or study:

- **In a library**—User-defined functions that you imported into a library are available after you publish them. You can use a function in a library when:
  - You are working in the library.
  - You are working in a study for which the library appears in the Library List.

  Note: If you are working in a study, you can use predefined functions that were created in the System Library when the System Library appears in the Library List.

- **In a study**—You can use user-defined functions that you imported into a study.

Functions in rules and rule templates

To add functions to rule expressions, use one of the following:

- **For workflow rules**—Workflow Expression Editor dialog box.
- **For rule templates**—Edit Rule Template dialog box > Definition tab.
- **For data-entry rules**—Rule Wizard Expression tab > Functions tab.
- **For global conditions**—Edit Global Conditions dialog box > Functions tab.

You can use the functions that are registered in a study and in the libraries that appear in the Libraries List in the Study Editor. Both predefined and user-defined functions appear in these locations:

- In the Expression workplace. As you type, a list of the rule model components that you can use in the expression appears dynamically in the Expression workspace. When you select Functions and type a period, an alphabetical list of functions appears and is available for selection.
- In the Functions tab. Predefined functions are listed by category. For more information, see *Predefined functions in the System Library* (on page 61).

You can incorporate functions in an expression using either location or by combining function information from both locations. Both dynamic expression prompts and the Functions tab appear in:

- Expression tab of the Rule Wizard.
- Workflow Expression Editor dialog box.
- Edit Global Conditions dialog box.
- Definition tab of the Edit Rule Template dialog box.

The list of functions is ordered in the following way:

1 Functions in the study.

2 Functions published in the first library on the Library List.

3 Functions published in the second library on the Library List (and so on).

Including a function in a rule expression

Use either or both of the following methods:

- Select the function using the dynamic prompts that appear in the Expression workspace as you type.

  For more information, see *Using dynamic prompts in the Expression workplace* (on page 9).

- Drag the function from the Functions tab to the Expression workspace.

  If parameters are associated with the function, the Invoke Function dialog box appears.

Note: The expression contains the name of the function, followed by parameters, which are enclosed by parentheses. Optionally, you can type the name of the function and its parameters.

Importing a user-defined function

Before you can use a function that is created outside of the Central Designer application, you must import it to a study or library.

1 In a study, select the **Study Information** Explorer bar.

or

In a library, select the **Library Information** Explorer bar.

2 Select **InForm**.

3 Select the **Functions** tab.

4 Click **Import Function**.

The Function File dialog box appears.

5 Locate the .NET assembly (a DLL file) containing the functions to import, and click **Open**.

The functions included in the .NET assembly are stored in the Central Designer database and are added to the list of functions in the grid.

Viewing and editing a function

1 In a study, select the **Study Information** Explorer bar.

or

In a library, select the **Library Information** Explorer bar.

2 Select **InForm**.

3 Select the **Functions** tab.

4 Select or right-click the function, and click **Edit**.

or

Double-click the function.

The Edit Function dialog box appears.

5 Edit the function as necessary, and click **OK** when you are finished.

For more information, see *Edit Function dialog box—Option descriptions* (on page 157).

Deleting a function

1 In a study, select the **Study Information** Explorer bar.

or

In a library, select the **Library Information** Explorer bar.

2 Select **InForm**.

3 Select the **Functions** tab.

4 Select or right-click the function, and click **Delete**.

A confirmation dialog box appears.

5 Click **OK**.

About predefined functions

A function is a reusable piece of code that extends the behavior of a rule. Many predefined functions are available by default in the System Library.

This section describes the predefined functions in the Central Designer application. The description of each function includes:

- Purpose of the function.
- Syntax, including parameters.
- Name, description, and data type of each parameter.
- Description of what the function returns.
- Notes about how the function works.

Date time processing

Functions that process date time items use a customized version of the .NET DateTime data type to enable the handling of incomplete date time fields. In the descriptions of functions, this customized data type is called PFDateTime.

To handle incomplete date time fields (where at least one part of the date time field has the value UNK, for UNKNOWN), date time fields are normalized. Normalization makes sure that UNK date parts have no effect on date computations or comparisons by setting the unknown part of all dates that are involved in the computation to a neutral value. Normalization is performed by all functions that perform date time comparisons or calculations.

During normalization:

- The appropriate date time part value from a template is substituted for each UNK date time part or for a date time field that does not allow entry in all date time parts (for example, a date time field that accepts only dates and no times).

  The default template used in date time functions is 2000-01-01-12:00:00 (Year-Month-Day-Hour-Minute-Second). If you need to perform date time calculations that use a different template for normalization, you can use the _NormalizeDate functions to specify a custom template.
- When using date comparison functions, an additional normalization step occurs to account for the possibility that different date time parts could be marked UNK in the two date time fields. In this case, if a date time part in either date time field has the value UNK, the date time parts in both fields are assigned values from the template.

The following examples use the default template of 2000-01-01-12:00:00.

Date time type	Before normalization	After normalization
One date time field	2007-04-UNK-16:20:UNK	2007-04-01-16:20:00
Two date time fields with same UNK date time part	07:22:UNK	07:22:00
	15:14:UNK	15:14:00

Date time type	Before normalization	After normalization
Two date time fields with different UNK date time parts	2006-12-UNK	2006-01-01-12:00:00
	2007-UNK-15	2007-01-01-12:00:00

Exceptions

When the execution of a function results in an exception:

- Processing of the rule stops.
- An InForm Server Error occurs in the InForm client application.
- An entry is posted in the Event log on the InForm server computer.

Predefined functions in the System Library

Classification	Function	Purpose
Clinical Functions.	*_CalculateBMI* (on page 63)	Calculates the Body-Mass Index, based on height and weight.
	_CalculateBSA (on page 64)	Calculates the body surface area.
	_CalculateWaistHipRatio (on page 66)	Calculates the waist-to-hip ratio.
Clinical Validation Functions.	*_CheckPatientInitials* (on page 67)	Checks the format of the subject initials.
Date Manipulation Functions.	*_CalculateDateTime* (on page 64)	Calculates a new date and time by adding an interval to or subtracting an interval from another date and time. The interval is based on a date part, for example, a number of days.
	_CompareDates (on page 67)	Compares two date times to determine whether the first date time is less than, equal to, or greater than the second date time.
	_CompareDatesWithRange (on page 68)	Compares two date times to determine whether they are within a specified range of each other.
	_GetCurrentDate (on page 72)	Returns the current date and time on the InForm server.
	_GetDateDifference (on page 73)	Returns the number of units between two specified dates, based on the requested date part.
	_NormalizeDate (on page 88)	Normalizes a date based on the specified template.

Classification	Function	Purpose
	_NormalizeDateToMax (on page 90)	Normalizes UNKNOWN or EMPTY date time parts to the maximum values for those parts.
Date/Time, Float, Integer, or Text Array Comparison Functions. These functions take different parameters depending on the data type of the value to which you are comparing the values in the array.	*_Count* (on page 70)	Counts the number of occurrences of a specified value in an array.
	_IsValueGreaterThanArray (on page 77)	Checks whether a specified value is greater than all values in an array.
	_IsValueGreaterThanOrEqualToArray (on page 79)	Checks whether a specified value is greater than or equal to at least one of the values in an array.
	_IsValueInArray (on page 81)	Checks whether a specified value is equal to one of the values in an array.
	_IsValueLessThanArray (on page 83)	Checks whether a specified value is less than all values in an array.
	_IsValueLessThanOrEqualToArray (on page 86)	Checks whether a specified value is less than or equal to at least one of the values in an array.
EDC	*GetScreeningNumber* (on page 74)	Returns the screening number of the subject in the InForm application.
	GetSiteTime (on page 75)	Returns the current time at the data-entry site.
	GetSiteMnemonic (on page 74)	Returns the mnemonic (abbreviated site name, specified in the Admin area of the InForm application) for the site of the current subject.
	GetSiteLocale (on page 72)	Returns the locale of the study site, such as en-US or ja-JP.
	GetTrialName (on page 76)	Returns the name of the trial in the InForm application.
	GetUserName (on page 76)	Returns the name of user who is currently logged in to the InForm application.

Classification	Function	Purpose
	GetSubjectNumber (on page 76)	Returns the number of the subject in the InForm application.
	Randomize (on page 93)	Returns a drug kit number based on the randomization type and the sequence and drug kit numbers stored in the randomization source database on the InForm host computer.

_CalculateBMI

Calculates the Body-Mass Index, based on height and weight.

Syntax

_CalculateBMI(*height,weight*)

Parameters

Parameter	Definition	Data type
height	Height of the subject, in centimeters.	Float
weight	Weight of the subject, in kilograms.	Float

Returns

BMI measurement (Float).

Notes

- The Body-Mass index is calculated using the following formula:

  ```
  weight / (height * height)
  ```
- Height is assumed to be in centimeters, and weight is assumed to be in kilograms.

Exceptions

An exception of type Argument Exception is returned if the height measurement equals zero. This exception results in:

- Failure of the rule execution.
- An InForm Server Error in the InForm client application.
- An entry in the Event log on the InForm server computer.

Example

The following rule is created at the form level on the Vital Signs form, which has items Weight and BMI. The rule also refers to the item Ht in a mapping called RulesLS.

```
evaluate on Form Submission
   value = _CalculateBMI(RulesLS.DSRules.Ht.Value, this.Weight.Value)
always
   set this.BMI.Value = value
```

_CalculateBSA

Calculates the body surface area.

Syntax

_CalculateBSA(*height,weight*)

Parameters

Parameter	Definition	Data type
height	Height of the subject, in centimeters.	Float
weight	Weight of the subject, in kilograms.	Float

Returns

Body surface area (Float).

Notes

- Calculates the BSA using the following formula:

  ```
  (Float)(0.007184 * Math.Pow(height,0.725) * Math.Pow(weight,0.425));
  ```

  (Math.Pow is a method that raises a value to the specified power.)
- Height is assumed to be in centimeters, and weight is assumed to be in kilograms.

_CalculateDateTime

Calculates a new date and time by adding an interval to or subtracting an interval from another date and time. The interval is based on a date part, for example, a number of days.

Syntax

_CalculateDateTime(*date,interval,datePart*)

Parameters

Parameter	Definition	Data type
date	Original date and time.	PFDateTime
interval	Time interval to add to or subtract from the original date time.	Integer
datePart	Date part to which the value specified in the interval parameter is added.	Integer

Returns

New date time (PFDateTime).

Notes

- Date intervals are calculated in the following way:
  - Years are measured in 365- or 366-day increments, depending on the number of calendar days in the year.

    Note: If the time period includes a February month with 29 days, then 366 days equal 1 year. Otherwise, 365 days equal 1 year.

    - Fewer than 365 (or, as mentioned above, 366 days)—0 years.
    - 365 (or 366) to 729 (or 730) days—1 year.
    - 729 (or 730) to 1094 (or 1095) days—2 years.
  - Months are measured in 28-, 29-, 30-, or 31-day increments, depending on the number of days in the month.

    For example, if a start date is in a month with 31 days, then 31 days from the start date is one month. If the date range goes from February until September, then each month's number of days is used for the calculation. For example, 28 (or 29, if February has 29 days) days from the start date in February is one month; 31 days from the date in March is another month; 30 days from the date in April is another month; and so on.

    For increments that are fewer than 28, 29, 30, or 31 days, depending on the month, the range is 0 months.
  - Days are measured in 24-hour increments. For example:
    - Fewer than 24 hours—0 days.
    - 24 to 47 hours—1 day.
    - 48 to 71 hours—2 days.
- The new date time is based on the sum of the original date time and the interval.
- Before the interval is added, the date is normalized. For more information, see *About date time processing* (on page 60).
- Valid dateParts are taken from the DateTimeParts enumerator, which returns an integer constant. (An enumerator is a variable type that represents a restricted list of values.) To reference a date part, type **DateTimeParts.***datePart,* where *datePart* is one of the following:
  - Years
  - Months
  - Days
  - Hours
  - Minutes
  - Seconds

Example

This example sets a date for follow-up that is four weeks after the initial date entered. Both items are on the same form, so the rule is created at the form level. There is no enumeration for weeks in DateTimeParts, so the rule uses twenty-eight days instead of four weeks.

```
evaluate on Form Submission
   value = _CalculateDateTime(this.InitialExamDate.Value, 28,
   DateTimeParts.Days)
always
   set this.FollowupExamDate.Value = value
```

_CalculateWaistHipRatio

Calculates the waist-to-hip ratio.

Syntax

_CalculateWaistHipRatio(*waist,hip*)

Parameters

Parameter	Definition	Data type
waist	Waist measurement of the subject.	Float
hip	Hip measurement of the subject.	Float

Returns

Waist-to-hip ratio (Float).

Notes

- Waist-to-hip ratio is calculated as *waist/hip*.
- Both measurements must be in the same units. For example, if one measurement is in centimeters, the other must be, as well.

Exceptions

An exception of type Argument Exception is returned if the hip measurement equals zero. This exception results in:

- Failure of the rule execution.
- An InForm Server Error in the InForm client application.
- An entry in the Event log on the InForm server computer.

Example

The following rule is attached to the Vital Signs form and does not need any objects from outside the form. The form includes a waistCircumference item to record the measurement of waist circumference and a hipCircumference item to record the measurement of hip circumference. Both allow the value to be recorded in centimeters or inches, but both values are normalized to centimeters.

```
evaluate on Form Submission
   value = _CalculateWaistHipRatio(this.waistCircumference.Value,
   this.hipCircumference.Value)
```

```
always
   set this.whRatio.Value = value
```

_CheckPatientInitials

Checks the format of the subject initials.

Syntax

_CheckPatientInitials(*initials*)

Parameters

Parameter	Definition	Data type
initials	Subject initials.	Text

Returns

True or False (Boolean), indicating whether the subject initials are in the correct format.

Notes

The entered string is validated to make sure that it consists of three letters or two letters with a dash in the middle to indicate that no middle name was provided.

_CompareDates

Compares two date times to determine whether the first date time is less than, equal to, or greater than the second date time.

Syntax

_CompareDates(*date1,date2*)

Parameters

Parameter	Definition	Data type
date1	First date to use.	PFDateTime
date2	Second date to use.	PFDateTime

Returns

One of the following (Integer):

- **-1**—If date1 is less than date2.
- **0**—If date1 equals date2.
- **1**—If date1 is greater than date2.

Notes

Both dates are normalized before comparison. For more information, see *About date time processing* (on page 60).

Example

This example of a constraint rule uses both the GetSiteTime and _CompareDates functions to check whether the date entered is in the future. It is attached at the item level to a DOV item.

```
evaluate on Form Submission
   value = _CompareDates(this.Value, GetSiteTime())
when value == 1
   issue query: Date cannot be in the future compared to System Date
```

You could also write the rule to evaluate to true or false instead of to a numeric value:

```
evaluate on Form Submission
   value = _CompareDates(this.Value, GetSiteTime()) == 1
when value is true
   issue query: Date cannot be in the future compared to System Date
```

_CompareDatesWithRange

Compares two date times to determine whether they are within a specified range of each other.

Syntax

_CompareDatesWithRange(*date1*,*date2*,*datePart*,*rangeMin*,*rangeMax*)

Parameters

Parameter	Definition	Data type
date1	First date to use.	PFDateTime
date2	Second date to use.	PFDateTime
datePart	Date part used to calculate the interval between date1 and date2.	Integer
rangeMin	Minimum number in the range to compare with the interval between date1 and date2.	Integer
rangeMax	Maximum number in the range to compare with the interval between date1 and date2.	Integer

Returns

True or False (Boolean), indicating whether the first date time is within the specified range of the

second date time.

Notes

- Date intervals are calculated in the following way:
  - Years are measured in 365- or 366-day increments, depending on the number of calendar days in the year.

    Note: If the time period includes a February month with 29 days, then 366 days equal 1 year. Otherwise, 365 days equal 1 year.

    - Fewer than 365 (or, as mentioned above, 366 days)—0 years.
    - 365 (or 366) to 729 (or 730) days—1 year.
    - 729 (or 730) to 1094 (or 1095) days—2 years.
  - Months are measured in 28-, 29-, 30-, or 31-day increments, depending on the number of days in the month.

    For example, if a start date is in a month with 31 days, then 31 days from the start date is one month. If the date range goes from February until September, then each month's number of days is used for the calculation. For example, 28 (or 29, if February has 29 days) days from the start date in February is one month; 31 days from the date in March is another month; 30 days from the date in April is another month; and so on.

    For increments that are fewer than 28, 29, 30, or 31 days, depending on the month, the range is 0 months.
  - Days are measured in 24-hour increments. For example:
    - Fewer than 24 hours—0 days.
    - 24 to 47 hours—1 day.
    - 48 to 71 hours—2 days.
- The method compares the interval between the specified datePart in each date time and determines whether the interval is within the range indicated by the rangeMin and rangeMax parameters. This method is useful when determining whether the age of a subject is within a specific age range. The minimum and maximum ranges are inclusive.
- Both dates are normalized before comparison. For more information, see *About date time processing* (on page 60).
- Valid dateParts are taken from the DateTimeParts enumerator, which returns an integer constant. (An enumerator is a variable type that represents a restricted list of values.) To reference a date part, type **DateTimeParts.***datePart,* where *datePart* is one of the following:
  - Years
  - Months
  - Days
  - Hours
  - Minutes
  - Seconds

Example

This example checks whether a reported adverse event lasts for more than six months. The rule is defined at the form level and checks the StartDate and EndDate items in that form. This rule creates a query when an adverse event lasts more than six months.

```
evaluate on Form Submission
   value = _CompareDatesWithRange(this.StartDate.Value, this.EndDate.Value,
   DateTimeParts.Months, 0, 6)
when value is false
   issue query on this.EndDate: Adverse Event lasted more than six months.
```

_Count

Counts the number of occurrences of a specified value in an array. This function takes different parameters depending on the data type of the value you are counting.

Data type of value to count	Function to use
String	***_Count(string, array, boolean)*** (on page 70)
Date	***_Count(date,array)*** (on page 71)
Integer	***_Count(integer,array)*** (on page 71)
Float	***_Count(float,array)*** (on page 71)

_Count(String, Array, Boolean)

Counts the number of occurrences of a string value in an array.

Syntax

_Count(valueToCount,valueList,caseSensitive)

Parameters

Parameter	Definition	Data type
valueToCount	Value to count	String
valueList	Array of values to search	Array of String values
caseSensitive	Case sensitive search flag, True or False. If the value is True, the function takes case into consideration when searching for strings that match the value to count. For example, if caseSensitive is True, ABC is not counted as an occurrence of the string abc.	Boolean

Returns

The number of occurrences of a string value in an array.

_Count(Date, Array)

Counts the number of occurrences of a date time value in an array.

Syntax

_Count(valueToCount,valueList)

Parameters

Parameter	Definition	Data type
valueToCount	Value to count	PFDateTime
valueList	Array of values to search	Array of PFDateTime values

Returns

The number of occurrences of a given date time value in an array.

_Count(Integer, Array)

Counts the number of occurrences of an integer value in an array.

Syntax

_Count(valueToCount,valueList)

Parameters

Parameter	Definition	Data type
valueToCount	Value to count	Integer
valueList	Array of values to search	Array of Integer values

Returns

The number of occurrences of a given integer value in an array.

_Count(Float, Array)

Counts the number of occurrences of a float value in an array.

Syntax

_Count(valueToCount,valueList)

Parameters

Parameter	Definition	Data type
valueToCount	Value to count	Float
valueList	Array of values to search	Array of Float values

Returns

The number of occurrences of a given float value in an array.

_GetCurrentDate

Returns the current date and time on the InForm server.

Syntax

_GetCurrentDate()

Returns

Current date and time at the location of the InForm server computer.

Notes

To get the current time in the time zone of the data-entry site, use ***_GetSiteTime*** (on page 75).

GetSiteLocale

Returns the locale of the study site, such as en-US or ja-JP. You select locales in the Study Editor.

Syntax

GetSiteLocale()

Returns

Locale of the study site.

Notes

You can use the site locale to format a date time or floating point number. For example, you can write a user-defined function that uses the locale that is returned by this function and the date time value that a user enters in the InForm application to format the date time value according to the standard formatting of the locale.

_GetDateDifference

Returns the number of units between two specified dates, based on the requested date part.

Syntax

_GetDateDifference(*date1***,***date2***,***units***)**

Parameters

Parameter	Definition	Data type
date1	First date to use. date1 is subtracted from date2.	PFDateTime
date2	Second date to use.	PFDateTime
units	Unit to use when computing the difference. Units are taken from DateTimeParts: Years, Months, Days, Hours, Minutes, Seconds.	Integer

Returns

A positive, zero, or negative interval length depending on whether the first date is earlier than, equal to, or later than the second date. Unknown date parts are normalized. Results are rounded down.

Notes

- GetDateDifference returns positive value if date1 is before date2 and returns a negative value when date 2 is before date 1.
- Date intervals are calculated in the following way:
  - Years are measured in 365- or 366-day increments, depending on the number of calendar days in the year.

    Note: If the time period includes a February month with 29 days, then 366 days equal 1 year. Otherwise, 365 days equal 1 year.

    - Fewer than 365 (or, as mentioned above, 366 days)—0 years.
    - 365 (or 366) to 729 (or 730) days—1 year.
    - 729 (or 730) to 1094 (or 1095) days—2 years.
  - Months are measured in 28-, 29-, 30-, or 31-day increments, depending on the number of days in the month.

    For example, if a start date is in a month with 31 days, then 31 days from the start date is one month. If the date range goes from February until September, then each month's number of days is used for the calculation. For example, 28 (or 29, if February has 29 days) days from the start date in February is one month; 31 days from the date in March is another month; 30 days from the date in April is another month; and so on.

    For increments that are fewer than 28, 29, 30, or 31 days, depending on the month, the range is 0 months.
  - Days are measured in 24-hour increments. For example:

  - Fewer than 24 hours—0 days.
  - 24 to 47 hours—1 day.
  - 48 to 71 hours—2 days.
- Valid dateParts are taken from the DateTimeParts enumerator, which returns an integer constant. (An enumerator is a variable type that represents a restricted list of values.) To reference a date part, type **DateTimeParts.***datePart,* where *datePart* is one of the following:
  - Years
  - Months
  - Days
  - Hours
  - Minutes
  - Seconds
- Dates are normalized before the number of units is calculated. For more information, see ***About date time processing*** (on page 60).
- The entire date is used in the calculation, not only the specified date part. Results are rounded down to the nearest integer. For example, if _GetDateDifference runs with the following data, it returns 0, not 1:

  _GetDateDifference (new DateTime(2007,1,10,0,0,0), new DateTime (2007,2,1,0,0,0), DateTimeParts.Months)

Example

The _GetDateDifference function is used to determine how long an adverse event lasts and to store the result in an item. This rule is attached to the AE form, which contains the OnsetDate and EndDate items, as well as an AEDuration item to store the calculated value.

```
evaluate on Form Submission
   value = _GetDateDifference(this.OnsetDate.Value, this.EndDate.Value,
   DateTimeParts.Days)
always
   set this.AEDuration.Value = value
```

If OnsetDate is January 1, 2008, and EndDate is January 6, 2008, the function returns 5.

GetScreeningNumber

Returns the screening number of the subject in the InForm application.

Syntax

GetScreeningNumber()

Returns

Screening number of the subject in the InForm application.

GetSiteMnemonic

Returns the mnemonic (abbreviated site name, specified in the Admin area of the InForm application) for the site of the current subject.

Syntax

GetSiteMnemonic()

Returns

Mnemonic of the site of the current patient.

GetSiteTime

Returns the current time at the data-entry site.

Syntax

GetSiteTime()

Returns

Current time at the data-entry site.

Notes

The function returns the current time in the time zone of the data-entry site, not the time zone of the location of the InForm server. To get the current time at the InForm server computer, use *_GetCurrentDate* (on page 72).

Example

This example of a constraint rule uses both the GetSiteTime and _CompareDates functions to check whether the date entered is in the future. It is attached at the item level to a DOV item.

```
evaluate on Form Submission
   value = _CompareDates(this.Value, GetSiteTime())
when value == 1
   issue query: Date cannot be in the future compared to System Date
```

You could also write the rule to evaluate to true or false instead of to a numeric value:

```
evaluate on Form Submission
   value = _CompareDates(this.Value, GetSiteTime()) == 1
when value is true
   issue query: Date cannot be in the future compared to System Date
```

GetTrialName

Returns the name of the trial in the InForm application.

Syntax

GetTrialName()

Returns

Name of the trial in the InForm application.

Notes

This function is particularly useful for populating email parameters, writing query text, and writing user-defined functions. For example, you can display the study name in the query text of a rule or in the email subject and body.

GetUserName

Returns the name of user who is currently logged in to the InForm application.

Syntax

GetUserName()

Returns

Name of the currently logged-on user in the InForm application.

Notes

The **GetUserName()** function is supported in the following InForm releases:

- Release 4.6 SP2 and above, except release 4.7.
- Release 5.0 and above.

Note: If a rule containing the function runs in an unsupported release, the rule returns an empty string.

GetSubjectNumber

Returns the number of the subject in the InForm application.

Syntax

GetSubjectNumber()

Returns

Number of the subject in the InForm application.

Notes

- This function calls the GetPatientNumber() method in the InForm application. The number is

assigned to the subject during enrollment.

- In the Rule Test Cases dialog box, rules that use this function return TestSubjectNumber.

_IsValueGreaterThanArray

Checks whether a specified value is greater than all values in an array. This function takes different parameters depending on the data type of the value to which you are comparing the values in the array.

Data type to compare	Function to use
Date time	*_IsValueGreaterThanArray (PFDateTime, Array)* (on page 77).
Float	*_IsValueGreaterThanArray (Float, Array)* (on page 78).
Integer	*_IsValueGreaterThanArray (Integer, Array)* (on page 78).

_IsValueGreaterThanArray (PFDateTime, Array)

Checks if a date time value is greater than all of the date times in a list.

Syntax

_IsValueGreaterThanArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Date time value with which to compare the values in the list.	PFDateTime
valueList	Array of date time values against which to compare the value specified in the valueToTest parameter.	Array of PFDateTime values

Returns

True or False (Boolean), indicating whether the date time to test is greater than all of the date times in the array.

Notes

- The valueToTest value is compared to each element of the array. If the value is greater than every element of the array, the method returns True. If at least one element is less than or equal to the value, the method returns False.
- All comparisons are done using the _CompareDates method and are normalized. For more information, see *About date time processing* (on page 60).

_IsValueGreaterThanArray (Float, Array)

Checks if a float value is greater than all of the date times in an array.

Syntax

_IsValueGreaterThanArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Float value with which to compare the values in the array.	Float
valueList	Array of float values against which to compare the value specified in the valueToTest parameter.	Array of Float values

Returns

True or False (Boolean), indicating whether the float value to test is greater than all of the elements in the array.

Notes

The valueToTest value is compared to each element of the array. If the value is greater than every element of the array, the method returns True. If at least one element is less than or equal to the value, the method returns False.

_IsValueGreaterThanArray (Integer, Array)

Checks if an integer value is greater than all of the date times in an array.

Syntax

_IsValueGreaterThanArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Integer value with which to compare the values in the array.	Integer
valueList	Array of integer values against which to compare the value specified in the valueToTest parameter.	Array of Integer values

Returns

True or False (Boolean), indicating whether the integer value to test is greater than all of the elements in the list.

Notes

The valueToTest value is compared to each element of the array. If the value is greater than every element of the array, the method returns True. If at least one element is less than or equal to the

value, the method returns False.

_IsValueGreaterThanOrEqualToArray

Checks whether a specified value is greater than or equal to at least one of the values in an array. This function takes different parameters depending on the data type of the value to which you are comparing the values in the array.

Data type to compare	Function to use
Date time	*_IsValueGreaterThanOrEqualToArray (PFDateTime, Array)* (on page 79).
Float	*_IsValueGreaterThanOrEqualToArray (Float, Array)* (on page 80).
Integer	*_IsValueGreaterThanOrEqualToArray (Integer, Array)* (on page 80).

_IsValueGreaterThanOrEqualToArray (PFDateTime, Array)

Checks if a date time value is greater than or equal to all of the date times in an array.

Syntax

_IsValueGreaterThanOrEqualToArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Date time value with which to compare the values in the array.	PFDateTime
valueList	Array of date time values against which to compare the value specified in the valueToTest parameter.	Array of PFDateTime values

Returns

True or False (Boolean), indicating whether the date time to test is greater than or equal to all of the date times in the array.

Notes

- The valueToTest value is compared to each element of the array. If the value is greater than or equal to every element of the array, the method returns True. If at least one element is less than the value, the method returns False.
- All comparisons are done using the _CompareDates method and are normalized. For more information, see *About date time processing* (on page 60).

Example

This example checks whether the date of termination is greater than or equal to all visit dates for the patient and issues a query if the date of termination is less than any of the patient visit dates. The rule

is created at the item level and uses the mapping "RulesLS" and the data set and series within that mapping. All of the visit dates in a data series called VisitDates within the RulesLS mapping.

```
evaluate on Form Submission
   value = _IsValueGreaterThanOrEqualToArray(this.Value,
   RulesLS.DateCollection.VisitDates.Values)
when value is false
   issue query: Termination date must be later than any visit date.
```

_IsValueGreaterThanOrEqualToArray (Float, Array)

Checks if a float value is greater than or equal to at least one of the values in an array.

Syntax

_IsValueGreaterThanOrEqualToArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Float value with which to compare the values in the array.	Float
valueList	Array of float values against which to compare the value specified in the valueToTest parameter.	Array of Float values

Returns

True or False (Boolean), indicating whether the float value to test is greater than all of the elements in the array.

Notes

The valueToTest value is compared to each element of the array. If the value is greater than or equal to every element of the array, the method returns True. If at least one element is less than the value, the method returns False.

_IsValueGreaterThanOrEqualToArray (Integer, Array)

Checks if a specified integer value is greater than or equal to at least one of the integer values in an array.

Syntax

_IsValueGreaterThanOrEqualToArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Integer value with which to compare the values in the array.	Integer
valueList	Array of integer values against which to compare the value specified in the valueToTest parameter.	Array of Integer values

Returns

True or False (Boolean), indicating whether the integer value to test is greater than all of the elements in the array.

Notes

The valueToTest value is compared to each element of the array. If the value is greater than or equal to every element of the array, the method returns True. If at least one element is less than the value, the method returns False.

_IsValueInArray

Checks whether a specified value is equal to one of the values in an array. This function takes different parameters depending on the data type of the value to which you are comparing the values in the array.

Data type to compare	Function to use
Date time	*_IsValueInArray (PFDateTime, Array)* (on page 81).
Float	*_IsValueInArray (Float, Array)* (on page 82).
Integer	*_IsValueInArray (Integer, Array)* (on page 82).
Text	*_IsValueInArray (Text, Array)* (on page 83).

_IsValueInArray (PFDateTime, Array)

Checks if a specified date time value is equal to one of the date time values in an array.

Syntax

_IsValueInArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Date time value with which to compare the values in the array.	PFDateTime
valueList	Array of date time values against which to compare the value specified in the valueToTest parameter.	Array of PFDateTime values

Returns

True or False (Boolean), indicating whether the date time value to test is in the array of date times.

Notes

- The valueToTest value is compared to each element of the array. If the value is equal to at least one element of the array, the method returns True. Otherwise, the method returns False.
- All comparisons are done using the _CompareDates method and are normalized. For more information, see *About date time processing* (on page 60).

_IsValueInArray (Float, Array)

Checks if a specified float value is equal to one of the float values in an array.

Syntax

_IsValueInArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Float value with which to compare the values in the array.	Float
valueList	Array of float values against which to compare the value specified in the valueToTest parameter.	Array of Float values

Returns

True or False (Boolean), indicating whether the float value to test is in the array of float values.

Notes

The valueToTest value is compared to each element of the array. If the value is equal to at least one element of the array, the method returns True. Otherwise, the method returns False.

_IsValueInArray (Integer, Array)

Checks if a specified integer value is equal to one of the integer values in an array.

Syntax

_IsValueInArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Integer value with which to compare the values in the array.	Integer
valueList	Array of integer values against which to compare the value specified in the valueToTest parameter.	Array of Integer values

Returns

True or False (Boolean), indicating whether the integer value to test is in the array of integer values.

Notes

The valueToTest value is compared to each element of the array. If the value is equal to at least one element of the array, the method returns True. Otherwise, the method returns False.

Example

In this example of a global condition, when a user indicates in the AE form that a patient died as the result of an adverse event, a form called Death is enabled. Because the AE form is a common form that does not have reference to specific visits, the Outcome field from the AE form is collected in a mapping.

```
_IsValueInArray("Death", RulesLS.DSRules.Outcomes.Values)
```

This condition returns true or false. When the condition is applied to the Death form, the Death form appears when the expression is true.

_IsValueInArray (Text, Array)

Checks if a specified text value is equal to one of the text values in an array.

Syntax

_IsValueInArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Text value with which to compare the values in the array.	Text
valueList	Array of text values against which to compare the value specified in the valueToTest parameter.	Array of Text values

Returns

True or False (Boolean), indicating whether the text value to test is in the array of text values.

Notes

The valueToTest value is compared to each element of the array. If the value is equal to at least one element of the array, the method returns True. Otherwise, the method returns False.

_IsValueLessThanArray

Checks whether a specified value is less than all values in an array. This function takes different parameters depending on the data type of the value to which you are comparing the values in the array.

Data type to compare	Function to use
Date time	*_IsValueLessThanArray (PFDateTime, Array)* (on page 84).
Float	*_IsValueLessThanArray (Float, Array)* (on page 85).
Integer	*_IsValueLessThanArray (Integer, Array)* (on page 85).

_IsValueLessThanArray (PFDateTime, Array)

Checks if a specified date time value is less than all date time values in an array.

Syntax

_IsValueLessThanArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Date time value with which to compare the values in the array.	PFDateTime
valueList	Array of date time values against which to compare the value specified in the valueToTest parameter.	Array of PFDateTime values

Returns

True or False (Boolean), indicating whether the specified date time value is less than all date time values in an array.

Notes

- The valueToTest value is compared to each element of the array. If the value is less than every element of the array, the method returns True. If at least one element is greater than or equal to the value, the method returns False.
- All comparisons are done using the _CompareDates method and are normalized. For more information, see *About date time processing* (on page 60).

_IsValueLessThanArray (Float, Array)

Checks if a specified float value is less than all float values in an array.

Syntax

_IsValueLessThanArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Float value with which to compare the values in the array.	Float
valueList	Array of float values against which to compare the value specified in the valueToTest parameter.	Array of Float values

Returns

True or False (Boolean), indicating whether the specified float value is less than all float values in an array.

Notes

The valueToTest value is compared to each element of the array. If the value is less than every element of the array, the method returns True. If at least one element is greater than or equal to the value, the method returns False.

_IsValueLessThanArray (Integer, Array)

Checks if a specified integer value is less than all integer values in an array.

Syntax

_IsValueLessThanArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Integer value with which to compare the values in the array.	Integer
valueList	Array of integer values against which to compare the value specified in the valueToTest parameter.	Array of Integer values

Returns

True or False (Boolean), indicating whether the specified integer value is less than all integer values in an array.

Notes

The valueToTest value is compared to each element of the array. If the value is less than every element of the array, the method returns True. If at least one element is greater than or equal to the

value, the method returns False.

_IsValueLessThanOrEqualToArray

Checks whether a specified value is less than or equal to at least one of the values in an array. This function takes different parameters depending on the data type of the value to which you are comparing the values in the array.

Data type to compare	Function to use
Date time	*_IsValueLessThanOrEqualToArray (PFDateTime, Array)* (on page 86).
Float	*_IsValueLessThanOrEqualToArray (Float, Array)* (on page 87).
Integer	*_IsValueLessThanOrEqualToArray (Integer, Array)* (on page 87).

_IsValueLessThanOrEqualToArray (PFDateTime, Array)

Checks if a specified date time value is less than or equal to at least one of the date time values in an array.

Syntax

_IsValueLessThanOrEqualToArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Date time value with which to compare the values in the array.	PFDateTime
valueList	Array of date time values against which to compare the value specified in the valueToTest parameter.	Array of PFDateTime values

Returns

True or False (Boolean), indicating whether the specified date time value is less than or equal to at least one of the date time values in an array.

Notes

- The valueToTest value is compared to each element of the array. If the value is less than or equal to every element of the array, the method returns True. If at least one element is greater than the value, the method returns False.
- All comparisons are done using the _CompareDates method and are normalized. For more information, see *About date time processing* (on page 60).

_IsValueLessThanOrEqualToArray (Float, Array)

Checks if a specified float value is less than or equal to at least one of the float values in an array.

Syntax

_IsValueLessThanOrEqualToArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Float value with which to compare the values in the array.	Float
valueList	Array of Float values against which to compare the value specified in the valueToTest parameter.	Array of Float values

Returns

True or False (Boolean), indicating whether the specified float value is less than or equal to at least one of the float values in an array.

Notes

The valueToTest value is compared to each element of the array. If the value is less than or equal to every element of the array, the method returns True. If at least one element is greater than the value, the method returns False.

_IsValueLessThanOrEqualToArray (Integer, Array)

Checks if a specified integer value is less than or equal to at least one of the integer values in an array.

Syntax

_IsValueLessThanOrEqualToArray*(valueToTest,valueList)*

Parameters

Parameter	Definition	Data type
valueToTest	Integer value with which to compare the values in the array.	Integer
valueList	Array of integer values against which to compare the value specified in the valueToTest parameter.	Array of Integer values

Returns

True or False (Boolean), indicating whether the specified integer value is less than or equal to at least one of the integer values in an array.

Notes

The valueToTest value is compared to each element of the array. If the value is less than or equal to every element of the array, the method returns True. If at least one element is greater than the value,

the method returns False.

_NormalizeDate

Normalizes a date based on the specified template. For more information, see *About date time processing* (on page 60).

What to normalize	Function to use
One date time.	*_NormalizeDate(PFDateTime, PFDateTime)* (on page 88).
One date time and a reference date time.	*_NormalizeDate(PFDateTime, PFDateTime, PFDateTime)* (on page 89).

_NormalizeDate (PFDateTime, PFDateTime)

Normalizes a single date by replacing the parts that are entered as UNK with the date part values in the specified template.

Syntax

_NormalizeDate(*dateToNormalize,template***)**

Parameters

Parameter	Definition	Data type
dateToNormalize	Date time value to normalize by replacing UNK date time parts from a template.	PFDateTime
template	Date time value containing the date and time part values with which to replace UNK date and time parts. This template contains no unknown parts.	PFDateTime

Returns

A normalized date time in which all UNK parts are replaced with the corresponding part from the template.

Notes

During normalization, the appropriate date time part value from the specified template, in Year-Month-Day-Hour-Minute-Second format, is substituted for each UNK date time part. Any date time parts that are not allowed in the definition of the date time field are replaced with values from the template. For example, if the value of the dateToNormalize is 1987-04-UNK and the value of the template is 1980-01-01, the normalized value is 1987-04-01.

Example

```
In this example, the Start Date and End Date of an adverse event are compared
to make sure that the End Date is after the Start Date. However, the patient
may not know the exact start and end dates of the adverse event, so the Day
part of the date time fields is set to Allow Unknown. A valid comparison
requires that the dates first be normalized. Then the rule uses the
_CompareDates function to compare the dates and creates a query if StartDate is
```

```
later than EndDate. When normalizing, the function substitutes 1 when the day
is omitted.
evaluate on Form Submission
   value = _CompareDates(
      _NormalizeDate(this.EndDate.Value, new DateTime(2007, 1, 1, 0, 0, 0)),
      _NormalizeDate(this.StartDate.Value, new DateTime(2007, 1, 1, 0, 0, 0))
   )
when value == -1
   issue query on this.EndDate: End Date cannot be prior to Start Date
```

_NormalizeDate (PFDateTime, PFDateTime, PFDateTime)

Normalizes a specified date and a reference date by replacing the parts that are entered as UNK with the date part values in the specified template. This function is used to normalize two dates that are being compared. It makes sure that the unknown parts in both dates are considered when determining which parts to normalize.

Syntax

_NormalizeDate(*dateToNormalize,referenceDate,template***)**

Parameters

Parameter	Definition	Data type
dateToNormalize	Date time value to normalize by replacing UNK date time parts from a template.	PFDateTime
referenceDate	Second date time value to normalize by replacing UNK date time parts from a template. This date can be compared to the dateToNormalize after both dates are normalized.	PFDateTime
template	Date time value containing the date and time part values with which to replace UNK date and time parts.	PFDateTime

Returns

A new date with all UNK parts replaced with the corresponding part from the template.

Notes

This function uses a second date as a reference. If either the date to normalize or the reference date has any UNK parts, those parts will be replaced with the corresponding part from the template. For example:

- **dateToNormalize**—2007/2/5.
- **referenceDate**—2006/5/UNK.
- **template**—2005/1/1.
- **Normalized result for dateToNormalize**—2007/2/1, even though its Day part is not unknown.

Since the reference date has an unknown Day part, both dates are normalized. For more information, see *About date time processing* (on page 60).

Example

In the following example, the _NormalizeDate function is used within the _CompareDates function to normalize two dates that have unknown parts:

- **date1**—1987-0-UNK-14:45
- **date2**—1987-2-7-10:UNK
- **template**—1981-10-1-0:0:0

_CompareDates ((_NormalizeDate (*date1*, *date2,* new DateTime (1981,10,1,0,0,0)),(_NormalizeDate (*date2*, *date1,* new DateTime (1981,10,1,0,0,0)))

Step	Result
The first date, *date1*, is normalized against the template 1981,10,1,0,0,0 with *date 2* as a reference.	normalized *date1* = 1987-0-1-14:0:0.
The second date, *date2*, is normalized against the template 1981,10,1,0,0,0 with *date 1* as a reference.	normalized *date2* = 1987-2-1-10:0:0.
The two normalized results are compared.	1 (*date2* is greater than *date1*).

_NormalizeDateToMax

Normalizes UNKNOWN or EMPTY date time parts to the maximum values for those parts. This function takes different parameters for an array of date times and a single date time.

What to normalize	Function to use
One date time	*_NormalizeDateToMax(Date)* (on page 91)
Array of date times	*_NormalizeDateToMax(array)* (on page 92)

_NormalizeDateToMax(Date)

Normalizes a single date time value by replacing any parts that are UNKNOWN or EMPTY with the maximum value for that part.

Syntax

_NormalizeDateToMax(dateToNormalize)

Parameters

Parameter	Definition	Data type
dateToNormalize	The date time to normalize.	PFDateTime

Returns

A new date time with all UNKNOWN or EMPTY parts replaced with their respective maximum values:

- **Year**—9999
- **Month**—12
- **Day**—(Last day of the month that year)
- **Hour**—23
- **Minute**—59
- **Second**—59

Notes

The _NormalizeDateToMax function accounts for leap years and returns the correct number of days in February of any year.

Example

The requirement in this example is to find the duration of a previous disease by subtracting the disease start date from the disease end date in the medical history. An unknown month and day are allowed for both start date and end date. An additional requirement is to substitute unknown date parts with the earliest number for start date and with the latest number for end date so that the longest possible duration of disease is assumed.

To accomplish this:

- Normalize the start date to January 1 of any year.
- Normalize the end date to one of the following:
  - December 31 of any year.
  - The last day of the given month if the month is known.

```
_GetDateDifference(
   _NormalizeDateToMax(this.EVTEND.Value),
   _NormalizeDate(this.EVTSTART.Value, new DateTime(1,1,1)),
   DateTimeParts.Days)
```

Sample returned values		
Start date	**End date**	**Returned value**
Feb 02, 2008	Oct 13, 2008	254 days
Feb UNK, 2008	Oct UNK, 2008	273 days. The start date is converted to Feb 1, 2008, and the end date is converted to Oct 31, 2008.
UNK UNK, 2008	UNK UNK, 2008	365 days. The start date is converted to Jan 1, 2008, and the end date is converted to Dec 31, 2008

_NormalizeDateToMax(Array)

Normalizes an array of date times by replacing any parts that are UNKNOWN or EMPTY with the maximum value for that part.

Syntax

_NormalizeDateToMax(datesToNormalize)

Parameters

Parameter	Definition	Data type
datesToNormalize	The array of date times to normalize.	Array of PFDateTime values

Returns

A new array of date times, with all UNKNOWN or EMPTY parts replaced with their respective maximum values:

- **Year**—9999
- **Month**—12
- **Day**—(Last day of the month that year)
- **Hour**—23
- **Minute**—59
- **Second**—59

Notes

The _NormalizeDateToMax function accounts for leap years and returns the correct number of days in February of any year.

You can use arrays called by either the Values() or Objects() property.

Example

The requirement in this example is to find the latest date among all dates in a repeating form, using the MaxValueInArrayDate function. If the date on the repeating form allows users to enter UNKNOWN for days or months, comparing dates without normalizing them can be unpredictable

and depends on the sequence of dates in the array. Therefore, use the _NormalizeDateToMax(Array) function to return normalized dates with predictable UNKNOWN parts.

Consider the following arrays of dates, in which the only difference in content is the value of the Day part, which causes the order of the dates to be different in each array:

ArrayOfDates1			
	Day	Month	Year
Date1	18	10	2008
Date2	UNK	10	2008

ArrayOfDates2			
	Day	Month	Year
Date1	UNK	10	2008
Date2	18	10	2008

Sample function calls	
Function call	Returned value
MaxValueInArrayDate(ArrayOfDates1)	Date1
MaxValueInArrayDate(ArrayOfDates2)	Date2
MaxValueInArrayDate(_NormalizeDateToMax (ArrayOfDates1))	Date2
MaxValueInArrayDate(_NormalizeDateToMax (ArrayOfDates2))	Date2
MaxValueInArrayDate(_NormalizeDate(ArrayOfDates1, new DateTime(1,1,1)))	Date1
MaxValueInArrayDate(_NormalizeDate(ArrayOfDates2, new DateTime(1,1,1)))	Date1

Randomize

Returns a drug kit number based on the randomization type and the sequence and drug kit numbers stored in the randomization source database on the InForm host computer.

Syntax

Randomize(*source,type*)

Parameters

Parameter	Definition	Data type
source	Randomization source list name, or stratification code.	String

Parameter	Definition	Data type
type	Randomization type: • **Simple Central**—The study uses one list of drug kits. Each new patient is assigned the next sequential drug kit number on the list. • **Central Stratified**—The study has multiple lists of drug kits. Each new subject is assigned to a drug kit list based on entered subject data. Then, the subject is assigned the next sequential drug kit number on that list. • **Simple Site**—Each site has a different drug kit list. Each new subject is assigned the next sequential drug kit number on the list for the site of the subject. • **Stratified by Site**—Each site has multiple lists of drug kits. Each new subject is first assigned to the set of lists for the site of the subject. Then, the subject is assigned to one of the site drug kit lists based on entered subject data. Finally, the subject is assigned the next sequential drug kit number on that list.	String

Returns

Drug kit number (String), in the format *sequence/drug_kit*

Notes

Using the Randomize function requires special configuration in the study design and on the computer hosting the InForm study:

- The rule using the Randomization function must calculate the value of an item with the Special Fields custom property value of **Randomization field (Randomization)**.
- The following configuration is required on the computer hosting the InForm study:
  - Define a randomization sequence for each different list of drug kits used in the study and install it in the study database using the MedML Installer tool.
  - Configure the randomization data source manager.
  - Configure the format of each randomization sequence.
  - Set up a randomization source database.
  - Create an ODBC connection for the randomization source database.

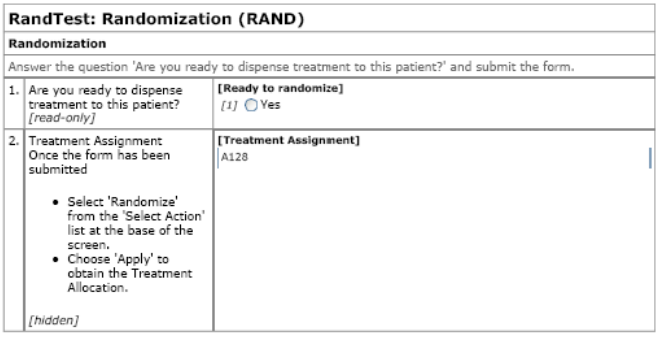
For more information, see the InForm documentation.

Example—Using the Randomize function

The randomization feature in the InForm application allows you to assign a drug kit to a subject based on a randomization scheme that has been chosen for a study. After randomization configuration is complete, one of the forms in the study contains a Drug Kit section. When a user clicks the Randomize button, the InForm application returns a drug kit number, along with associated information about the drug kit, in the Drug Kit section of the form.

For the randomization feature to work, you must create a randomization item. The randomization

item must be on a form with at least one other item that will be completed before the treatment information can be populated. The following example illustrates creating study objects for randomization and using the Randomize function.

Step	Action	Description
1	Create the following form on the Baseline study event.	• **Title**—Randomization • **RefName**—frmRandomization • **Short Title**—RAND
2	Create the following items on the frmRandomization form.	• **Title**—Ready to randomize • **RefName**—itmRandReady • **Type**—Integer • **Question**—Are you ready to dispense treatment to this patient? • **Codelist**—clYes • **Codelist item**—citmYes • **Display override**—Read Only • **Title**—Treatment Assignment • **RefName**—itmRandTreatment • **Question**—Treatment Assignment • **Type**—Text • **Display override**—Hidden • **Special Field Property**—Randomization Field (Randomization)
3	Generate form layout. For example:	

Step	Action	Description
4	Create a rule on itmRandTreatment.	The rule expression for the Randomization function uses the following format: **Randomize([listname], [randomization type])** **Note:** The value of [randomization type] can be one of the following (use the spacing that is specified): • **"Simple Central"** • **"Central Stratified"** • **"Simple Site"** • **"Stratified by Site"** See the following example for more details.

A Simple Central randomization with one list called "SimpleList" uses the following expression.

Description	Rule expression
	evaluate on Form Submission **value = Randomize("SimpleList", "Simple Central")**
The following action sets the value of the itmRandTreatment item to the Description and Sequence Number that is returned by the randomization process.	**always** **set this.Value = value.Description + " " + value.SequenceNumber**
The following action sets the value of the itmRandTreatment item to the Description that is returned by the randomization process.	**always** **set this.Value = value.Description**

About user-defined functions

Central Designer rules support simple expressions with no compound or flow-of-control statements. For complex calculations or decision making, rule expressions can reference functions. The Central Designer application provides predefined functions that are available in all studies and libraries.

Additionally, using a programming language such as C#, experienced programmers can create user-defined functions and make them available to Central Designer users for use in rules. The Central Designer application supports user-defined functions defined as public static methods in a .NET class, compiled into a .NET assembly.

Function overloading is supported in accordance with C# overloading rules. The C# compiler determines the function to call based on the best match of actual parameters with the formal parameters of the overloaded function definitions. Automatic type promotion and conversion is used when resolving a function call to the appropriate definition.

Rule designers can associate user-defined functions with libraries or studies by importing them in the Functions tab of a study or library. After a user-defined function has been imported into the study or library in which a user is working, or into a library in the Libraries List for a study, the function appears in the Functions tab in the Study Editor or Library Editor and in the References window of the Rule Wizard and related dialog boxes. From there, users can include the function in rule expression definitions.

When a deployment package is created, all assemblies containing user-defined functions used by the study are included in the deployment package.

In rule processing, a user-defined function is treated as a local method on the study object class.

Tips

- Try to make user-defined functions as generic as possible to increase the likelihood that the function will be reused. For example, if you must create a very complex function, consider splitting it up into two or three small functions that are more likely to be reused.
- Do not create user-defined functions in a production study or library. Instead, create them in a test study or development library.
- You can pass only base data types (Int, Double, String, PFDateTime, and Array) into a function.
- The full name of the external functions assembly is:

  Oracle.Designer.ExternalFunctions.dll.
- If you create a user-defined function, you must add the assembly as a reference in your function project and add a namespace for the assembly. The namespace is:

  Designer.UserDefinedFunctions
- Use C# attributes to categorize user-defined functions. Attributes are already defined in predefined functions.
- Oracle recommends testing the following parts of a user-defined function:
  - The user-defined function must encapsulate the logic that is required.
  - The correct data points must have been used as parameters.
  - The function must be deployed to the InForm application and tested with real values.
- Consider creating a repository of user-defined functions that all users can access. A central

repository is especially useful if users are working in different locales or if any part of the study is being outsourced.

Function definition requirements

All standard .NET classes and facilities are available to a function definition. The following requirements apply:

- The function must be defined as a public static method in a .NET class, compiled into a .NET assembly.
- The assembly containing user-defined function definitions must be self-contained. It cannot reference any other assemblies other than standard .NET framework assemblies.
- The method must return a value of one of the following types:
  - Boolean.
  - Double.
  - Int32.
  - String.
  - PFDateTime. PFDateTime is a customized version of the .NET DateTime data type that allows date time functions to handle incomplete date time fields. For more information, see *Date time processing* (on page 60). Note that user-defined functions cannot use System.DateTime as a parameter.
  - An array of any of the listed types.
- Each parameter for the method must be of one of the listed types.
- The method cannot directly address any study objects or any other global value. It can operate only on its parameters.
- The name of the method must be the same as the name of the user-defined function.
- The function must return a value. Functions returning void are not permitted.
- The function signature must be unique for each function. (More than one function can have the same name as long as the signature is unique.) The signature of a function consists of the:
  - Function name.
  - Type and order of the function parameters.
- Two functions that differ only in return type are not permitted because the function signature does not include the return type.
- If a study contains a user-defined function that performs a task such as reading from or writing to a file, accessing the database or the registry, making web service calls, running an external application, sending an email, or using the event log directly, if the assembly for the user-defined function is not signed with a strong named signature that is valid and trusted, the function does not work in the InForm application. For more information, see *Securing user-defined functions* (on page 99).

Securing user-defined functions

If a study contains a user-defined function that performs a task such as reading from or writing to a file, accessing the database or the registry, making web service calls, running an external application, sending an email, or using the event log directly, if the assembly for the user-defined function is not signed with a strong named signature that is valid and trusted, the function does not work in the InForm application.

To ensure that the user-defined functions and assemblies in your study projects and library projects are secure, Oracle recommends that you sign user-defined function assemblies using a strong named, valid and trusted signature. For more information, see *Building user-defined function assemblies* (on page 101).

In addition, user-defined functions that use the Log4Net application must use the latest version of Log4Net or the Central Designer Log4Net wrapper. The Log4Net wrapper allows untrusted custom functions to use Log4Net logging, and shields users from future Log4Net upgrades. In addition, the wrapper elevates permissions so that the following loggers can run properly:

- Console
- ADO.NET
- EventLog

The wrapper is available in the Oracle.Designer.ExternalFunctions.dll assembly, and contains the following classes and interface:

- PhaseForward.Designer.Shared.Functions.Log4Net.LogManager
- PhaseForward.Designer.Shared.Functions.Log4Net.Log
- PhaseForward.Designer.Shared.Functions.Log4Net.ILog

The methods provided in the classes and interface match those in the Log4Net documentation.

Attributes of user-defined functions

The Central Designer application provides the following required and optional attributes that you can use in function definitions:

- *DesignerFunctionClassification* (on page 99).
- *DesignerFunction* (on page 100).
- *DesignerParameter* (on page 101).

For an example of function definition code that uses these attributes, see *Sample function definition code* (on page 102).

Central Designer function attributes are defined in the Central Designer assembly Oracle.Designer.ExternalFunctions.dll. Include this assembly in the project references when building function definitions.

DesignerFunctionClassification

Required: No.

Valid on: The static method of the function on the class containing a collection of function definitions.

Purpose: Defines the classification of the function, which is used to group functions in the:

- Functions tab of the Study Editor.
- Function tab of the References section of the Rule Wizard and related dialog boxes.

If you do not include a DesignerFunctionClassification attribute, the classification of a function is either:

- The classification defined for the class that contains the function.
- The name of the class, if no classification is defined for the class that contains the function,

Parameter:

Name	Type	Description
Description	String	Classification name

DesignerFunction

Required: Yes.

Valid on: The static method that defines a Central Designer function.

Purpose: Identifies the static method as a user-defined Central Designer function and supplies the descriptive text that appears with the function in the Central Designer application. Only public static functions with this attribute are treated as user-defined function definitions.

Parameter:

Name	Type	Description
Description	String	• Description of the function. This text appears in the: ▪ Functions tab of the Study Editor. ▪ Rule Wizard and related dialog boxes.

DesignerParameter

Required: No.

Valid on: Function parameters.

Purpose: Provides the parameter description that is displayed in the Invoke Function dialog box.

Parameter:

Name	Type	Description
Description	String	Description of the parameter. This text appears in the Invoke Function dialog box.

User-defined function compilation

Compile user-defined function code as a DLL, not as an executable.

The compiler settings must include a reference to the full path of the ExternalFunctions.dll file in the Central Designer client installation folder (for example, C:\Program Files\Central Designer Client\1.0.0.859\Oracle.Designer.ExternalFunctions.dll).

Building user-defined function assemblies

If a study contains a user-defined function that performs a task such as reading from or writing to a file, accessing the database or the registry, making web service calls, running an external application, sending an email, or using the event log directly, if the assembly for the user-defined function is not signed with a strong named signature that is valid and trusted, the function does not work in the InForm application.

When you create a user-defined function with an assembly that you want to secure and sign, on each machine in your environment that builds assemblies, use the following procedure to install a PFX signing file to a crypto service (CSP) certificate container. The PFX file is then used to secure and sign the assembly for the user-defined function.

1 Add the following to the AssemblyInfo.cs file for your project:

```
// Tell compiler to use whatever key pair is stored in <container name> CSP
container. Ignore
// warning that this can be done via command-line switch
#pragma warning disable 1699
[assembly: AssemblyKeyName("<container name>")]
#pragma warning restore 1699
```

where:

- **container name**—CSP container name.

2 Open a Command Prompt window.

3 Navigate to the directory where your PFX file is located.

4 Run the following command:

sn -i <*PFX file name*> <*container name*>.

where:

- **PFX file name**—PFX signing file name.
- **container name**—CSP container name.

5 Build the user-defined function assembly.

Sample function definition code

The following code examples illustrate the definition of user-defined functions.

Body-Mass Index (BMI) function

The following code sample illustrates the definition of a function that returns the body-mass index (BMI) based on the values of height and weight. The function has a classification of Clinical. It takes two parameters, height and weight. Descriptive text for the function and its parameters appears in the Rule Wizard and the Invoke Function dialog box.

Note: Central Designer attributes are shown in bold.

```
using System;
using Oracle.Designer.ExternalFunctions;
namespace Customer.Designer.ClinicalData.Functions
{
   /// <summary>
   /// Summary description of Clinical
   /// </summary>
   [DesignerFunctionClassification("Clinical")]
   public class Clinical
   {
      private Clinical()
      {
      }

      /// <summary>
      /// Returns the body-mass index based on height and weight
      /// </summary>
      /// <param name="height">Height of the patient</param>
      /// <param name="weight">Weight of the patient</param>
      /// <returns></returns>
      [DesignerFunction ("Returns the body-mass index based on height and weight")]
      public static float BMI (
         [DesignerParameter ("Patient height")]
         float height,
         [DesignerParameter ("Patient weight")]
         float weight)
      {
         return weight / (height * height);
      }
   }
}
```

Date time difference function

The following code sample illustrates the definition of a function that returns the length of the interval between any two date time parts. If there are any unknown date time parts, the dates are normalized, and the results are rounded down. This function has a classification of Date Manipulation Function. It takes two dates and a date part as parameters. Descriptive text for the

function and its parameters appears in the Rule Wizard and the Invoke Function dialog box.

Note: Central Designer attributes are shown in bold.

```
using System;
using Oracle.Designer.ExternalFunctions;
namespace Customer.Designer.DateTime.Functions
{

   /// <summary>
   /// Returns the number of units between two dates, based on the requested
   interval.
   /// The units sign will be negative if date1 is before date2
   /// </summary>
   /// <param name="date1">The earlier date</param>
   /// <param name="date2">The later date</param>
   /// <param name="units">The interval to calculate</param>
   /// <returns>The integer number of interval units. The value will be negative
   if date1 is before date2.</returns>
   /// <remarks>Valid units are taken from the DateTimeParts enumerator, and
   include:
   /// <list type="bullet">
   /// <item>Years</item>
   /// <item>Months</item>
   /// <item>Days</item>
   /// <item>Hours</item>
   /// <item>Minutes</item>
   /// <item>Seconds</item>
   /// </list>
   /// Dates are normalized if either date has any UNKNOWN parts.  Results are
   rounded down
   /// to the nearest integer.  For example, <code>_GetDateDifference (new
   DateTime(2007,1,10,0,0,0),
   /// new DateTime (2007,2,1,0,0,0), DateTimeParts.Months)</code> will return
   0, not one</remarks>
   [DesignerFunctionClassification("Date Manipulation Functions")]

      [DesignerFunction("Returns the length of the interval between two dates.  The dates are normalized if
      there are any unknown parts.  Results are rounded down.")]
      public static int _GetDateDifference (
         [DesignerParameter("Date 1 for compare")]
         PFDateTime date1,
         [DesignerParameter("Date 2 for compare")]
         PFDateTime date2,
         [DesignerParameter(". Valid intervals are taken from DateTimeParts enum: Years, Months, Days,
         Hours, Minutes, Seconds")]
         int units)
      {
         if (date1 == null || date2 == null)
         {
            throw new ArgumentException("null parameter to GetDateDifference");
         }

         double difference = 0.0;

         DateTime normalizedEarlierDate = (DateTime)_NormalizeDate(date1, date2,
         new DateTime(2000,1,1,12,0,0));
         DateTime normalizedLaterDate = (DateTime)_NormalizeDate(date2, date1,
         new DateTime(2000,1,1,12,0,0));

         if (normalizedEarlierDate > normalizedLaterDate)
         {
            DateTime temp = normalizedEarlierDate;
            normalizedEarlierDate = normalizedLaterDate;
            normalizedLaterDate = temp;
```

```
      sign = -1;
   }
   TimeSpan ts = (DateTime) normalizedLaterDate - (DateTime)
   normalizedEarlierDate;
   switch (units)
   {
      case DateTimeParts.Hours:
         difference = ts.TotalHours;
         break;
      case DateTimeParts.Minutes:
         difference = ts.TotalMinutes;
         break;
      case DateTimeParts.Seconds:
         difference = ts.TotalSeconds;
         break;
      case DateTimeParts.Days:
         difference = ts.TotalDays;
         break;
         // Rounds down to nearest month
      case DateTimeParts.Months:
      case DateTimeParts.Years:
         difference = (normalizedLaterDate.Year -
         normalizedEarlierDate.Year)*12 + (normalizedLaterDate.Month -
         normalizedEarlierDate.Month);
         if (normalizedLaterDate.Day < normalizedEarlierDate.Day)
         {
            difference -= 1;
         }
         break;
         }
         if (units == DateTimeParts.Years)
         {
      difference /= 12;
         }
         int ret = ( (int)difference * sign);
         return ret;
}
```

CHAPTER 4

Constants

In this chapter

Constants tab of the Rule Wizard

A constant is a value that is defined in a library or study and that can be referenced by any rule.

You can define a constant in a library and change the value as needed for each study where you use it. For example, you might have a red blood cell count rule on a Hematology form. Because the value for the red blood cell count would be different for a Phase I arthritis trial and a Phase II leukemia trial, you could create a constant (for example, rbcMinCount) and change the value of the constant for each study.

User-defined constants are global. They do not have a scope and can be used in any rule created in a study or library.

You add constants to the Central Designer application:

- **In a library project**—In the Constants tab of the Library Editor.

  Constants created in a library are available after they have been published. You can use constants from the library in a study if the library appears on the Library List for the study.

- **In a study project**—In the Constants tab of the Study Editor.

  Constants created in a study are available only in that study.

Using constants in rules and rule templates

To add constants to rule expressions, use one of the following:

- **For workflow rules**—Workflow Expression Editor dialog box.
- **For rule templates**—Edit Rule Template dialog box > Definition tab.
- **For data-entry rules**—Rule Wizard Expression tab > Constants tab.
- **For global conditions**—Edit Global Conditions dialog box > Constants tab.

Note: RefNames for constants used in both a study and a library are not updated in both places if you update the RefName of the constant. If you update a RefName in one location, you must manually update it in the other.

The Constants tab (on the Expression tab of the Rule Wizard) lists all constants created in the study and in the libraries that appear in the Libraries List in the Study Editor. Any rule on any study object can reference any constant that appears.

Constants are organized in folders according to their classification. A classification is a user-defined value provided when a constant is created.

The list of constants is ordered in the following way:

1 Constants defined in the study.

2 Constants published in the first library on the Library List.

3 Constants published in the second library on the Library List (and so on).

You can publish special versions of constants from different libraries and allow the order of libraries in the Library List to determine which constant appears first.

For example, if you have two constants with matching names and data types, you see only the constant from the higher library in the Library List (or, if you have a local copy, in the study).

To include a constant in a rule expression:

- Drag the constant from the Constants tab to the Expression workspace.

Predefined constants in the System Library

Predefined constants appear in:

- The Constants tab of the System Library.
- The Rule Wizard > Expression tab > Constants tab in the System Library or in a study. Predefined constants do not appear in this location for user-created libraries.

When your Libraries Browser searches include the System Library, the predefined constants are included in the search.

In the Rule Wizard, when you drag a function to the Expression workspace, the Invoke Function dialog box appears. In the dialog box, you can expand the Constants tab, and drag the appropriate constant to the Value field for the parameter.

DateTimeParts constants—Use the following constants to specify a parameter in a predefined function that requires a date part. For example, the following functions include date part parameters:

- _CalculateDateTime
- _CompareDatesWithRange
- _GetDateDifference

Name	Description	Data type
Years, Months, Days, Hours, Minutes, Seconds	Date time part for years, months, days, hours, minutes, or seconds.	Integer

FormStates constants—Use the following constants to specify a form state using *HasState(Integer)* (on page 51).

Name	Description	Data type
Deleted	(Available only for repeating forms) The repeating form instance has been deleted.	Integer
Frozen	The form is frozen.	Integer
HasComment	The form has a comment at the form level.	Integer
HasData	The form has data.	Integer
HasMissingData	The form has missing data.	Integer
HasQueries	The form has queries.	Integer
Locked	The form is locked.	Integer
SdvComplete	The form has been marked source verified.	Integer
SdvPartial	The form has been partially source verified.	Integer
SdvReady	The form has been marked ready for source verification.	Integer
Signed	The form has been signed.	Integer

Name	Description	Data type
Skipped	The form has been marked not completed with a form-level comment.	Integer
Started	The form was started with subject data, comments, or queries.	Integer

Creating a constant

1 In the upper-left corner of the **Project Explorer**, click the **View Project/View Groups** icon so that you see the Project view. (When the Project view appears, the tooltip for the icon is **View Groups**.)

2 Select a study or library.

The Study or Library Editor appears.

3 Select the **Constants** tab.

4 Click **New Constant**.

The New Constant dialog box appears.

5 Fill in the fields of the dialog box, and click **OK**.

For more information, see:

Constants tab - Option descriptions (on page 158).

New Constant dialog box - Option descriptions (on page 159).

Deleting a constant

1. In the upper-left corner of the **Project Explorer**, click the **View Project/View Groups** icon so that you see the Project view. (When the Project view appears, the tooltip for the icon is **View Groups**.)
2. Select a study or library.

   The Study or Library Editor appears.
3. Select the **Constants** tab.
4. Select the constant to delete.
5. Click **Delete**.

   or

   Right-click, and select **Delete**.

CHAPTER 5

Data mappings

In this chapter

Data Mappings tab

Study objects and properties that are related to mappings are called data mappings. You add data mappings to rule expressions in the Rule Wizard using the dynamic expression prompts in the Expression tab or by dragging them from the Expression tab > Data Mappings tab. The Data Mappings tab lists:

- RefNames of the data mappings, data sets, and data series in the study or library.
- Rule model properties for data series.

  A data series has the properties of the item that is mapped to it. If a data series contains an item that collects more than one value, the rule model properties for repeating study objects appear so you can access an array of all of the values of the item.
- Methods for data sets.

  A method appears if you select the corresponding standard data dimension of the data set. You can use data set methods to return a subset of the data in the data set.
- Study events, forms, sections, and items that are mapped to each data set.
- Study objects appear if you select the corresponding standard data dimension of the data set. The properties of the study objects are used as parameters of data set methods.

You manage data mappings in the Project Explorer, where you create mappings, data sets, and data series and add items to data series.

Using data mappings in rules and rule templates

To add data mapping study objects to rule expressions, use one of the following:

- **For workflow rules**—Workflow Expression Editor dialog box.
- **For rule templates**—Edit Rule Template dialog box > Definition tab.
- **For data-entry rules**—Rule Wizard Expression tab > Data Mappings tab.
- **For global conditions**—Edit Global Conditions dialog box > Data Mappings tab.

Note: Rule expressions use RefNames, not titles. Therefore, RefNames appear in the Rule Wizard and in the rule expression.

The Data Mappings tab (on the Expression tab of the Rule Wizard) lists:

- RefNames of the data mappings, data sets, and data series in the study or library.
- Rule model properties for data series.

  A data series has the properties of the item that is mapped to it. If a data series contains an item that collects more than one value, the rule model properties for repeating study objects appear so you can access an array of all of the values of the item.
- Methods for data sets.

  A method appears if you select the corresponding standard data dimension of the data set. You can use data set methods to return a subset of the data in the data set.
- Study events, forms, sections, and items that are mapped to each data set.
- Study objects appear if you select the corresponding standard data dimension of the data set. The properties of the study objects are used as parameters of data set methods.

To include a data mapping study object or property in a rule expression:

- Drag the data mapping study object or study object property from the Data Mappings tab to the Expression workspace.

Icons used on the Data Mappings tab

Icon	Description
	Mapping.
	Data set.
	Data series. **Note:** If only one item is mapped to a data series, and that item occurs in the study in only one form and study event, the icon that appears is the rule model properties icon for the return type that matches the item's data type.
	Methods.
	Rule model properties with a DateTime return type.
	Rule model properties with a Float return type.
	Rule model properties with an Integer return type.
	Rule model properties with a Text return type.
	Rule model properties with a Boolean return type.
	Values[ ] and Variables[ ] rule model property.
	Groups of study events, forms, or items.
	Individual study events, forms, or items in groups.

Rule model properties for data series

Rule model properties are available for a data series in the Expression tab > Data Mappings tab in the Rule Wizard.

Count

Characteristic	Description
Icon	
Availability	A data series that contains: • Two or more items. or • An item that is used more than once in a study.
Return type	Int32 (for Central Designer Integer and YesNo types).
Description	The number of values that have been collected for the data series.

Values[]

Characteristic	Description
Icon	
Availability	A data series that contains multiple items or an item used on a repeating section, form, or study event.
Return type	Array.
Description	Array of values that have been collected for a data series. **Note:** To indicate the first value in the array, use 0.

Variables[]

Characteristic	Description
Icon	
Availability	A data series that contains: • Two or more items. or • An item that is used more than once in a study.
Return type	Array.

Characteristic	Description
Description	Array of items that are mapped to the data series. **Note:** To indicate the first value in the array, use 0.

Empty

Characteristic	Description
Icon	
Availability	A data series that contains a single item that is used only once in a study. Available below the Variables[ ] property.
Return type	Boolean.
Description	True or False. When True, a value for the item in the data series has not been provided.

Value

Characteristic	Description
Icon	One of the following: 
Availability	A data series that contains a single item that is used only once in a study.
Return type	One of the following: • String (for Central Designer string types). • DateTime (for Central Designer date time types). • Double (for Central Designer float types). • Int32 (for Central Designer Integer and YesNo types).
Description	Value of the item in the data series.

Methods for data sets

Data sets can contain multiple data series, which can contain multiple items. You use data set methods to return a data set that is a subset of the original data set. Data set methods always return a data set, not a single value.

Methods for data sets appear in the Rule Wizard in the Expression tab > Data Mappings tab. The list of methods that appear is based on the selection of standard data dimensions and the creation of custom data dimensions in the definition of the data set. In addition to the data set methods listed in the following table, one method appears for each custom dimension that is created for the data set.

Standard data dimensions selected	Data set method in Data Mappings tab
Event	*StudyEvent(StudyEvents)* (on page 119)
Event and Event Index	*StudyEvent(StudyEvents,Integer)* (on page 120)
Form	*Form(Forms)* (on page 120)
Form and Form Index	*Form(Forms,Integer)* (on page 120)
Section	*Section(Sections)* (on page 121)
Section and Section Index	*Section(Sections,Integer)* (on page 121)
Item	*Item(Items)* (on page 122)

To use a method in a rule expression:

1 Double-click the method.

If the method has one or more parameters, the Invoke Function dialog box appears.

2 Drag a study object from the **References** tab to the **Values** field to define a value for each parameter.

StudyEvent(StudyEvents)

Characteristic	Description
Icon	
Availability	When the **Event** standard data dimension is selected for the data set.
Returns	A data set subset with data from only the study event.
Syntax	**NameOfDataSet.StudyEvent(StudyEvents.studyEvent)**
Parameters	• **Parameter—studyEvent.** • **Definition**—Study event on which to filter. • **Data type**—Integer.

StudyEvent(StudyEvents,Integer)

Characteristic	Description
Icon	
Availability	When the **Event** and **Event Index** standard data dimensions are selected for the data set. **Note:** The Event Index standard data dimension is necessary for data sets that contain repeating study events.
Returns	A data set subset with data from only the study event.
Syntax	**NameOfDataSet.StudyEvent(StudyEvents.studyEvent, studyEventIndex)**
Parameters	• **Parameter—studyEvent.** • **Definition**—Study event on which to filter. • **Data type**—Integer. • **Parameter—studyEventIndex.** • **Definition**—Index of the repeating study event, from 0 to a maximum value, which equals the count of the repeating study event instances. • **Data type**—Integer.

Form(Forms)

Characteristic	Description
Icon	
Availability	When the **Form** standard data dimension is selected for the data set.
Returns	A data set subset with data from only the form.
Syntax	**NameOfDataSet.Form(Forms.form)**
Parameters	• **Parameter—form**. • **Definition**—Form on which to filter. • **Data type**—Integer.

Form(Forms,Integer)

Characteristic	Description
Icon	

Characteristic	Description		
Availability	When the **Form** and **Form Index** standard data dimensions are selected for the data set. **Note:** The Form Index standard data dimension is necessary for data sets that contain repeating study events.		
Returns	A data set subset with data from only the form.		
Syntax	**NameOfDataSet.Form(Forms.form, formIndex)**		
Parameters	**Parameter**	**Definition**	**Data type**
	form	Form on which to filter.	Integer
	formIndex	Index of the repeating form, from 0 to a maximum value, which equals the count of the repeating form instances.	Integer

Section(Sections)

Characteristic	Description
Icon	
Availability	When the **Section** standard data dimension is selected for the data set.
Returns	A data set subset with data from only the section.
Syntax	**NameOfDataSet.Section(Sections.section)**
Parameters	• **Parameter—section.** • **Definition**—Section on which to filter. • **Data type**—Integer.

Section(Sections,Integer)

Characteristic	Description
Icon	
Availability	When the **Section** and **Section Index** standard data dimensions are selected for the data set. **Note:** The Section Index standard data dimension is required for data sets that contain repeating sections.
Returns	A data set subset with data from only the section.
Syntax	**NameOfDataSet.Section(Sections.section, sectionIndex)**

Characteristic	Description
Parameters	• **Parameter—section** • **Definition**—Section on which to filter. • **Data type**—Integer • **Parameter—sectionIndex** • **Definition**—Index of the repeating section, from 0 to a maximum value, which equals the count of the repeating section instances. • **Data type**—Integer

Item(Items)

Characteristic	Description
Icon	
Availability	When the **Item** standard data dimension is selected for the data set.
Returns	A data set subset with data from only the item.
Syntax	**NameOfDataSet.Item(Items.item)**
Parameters	• **Parameter—item**. • **Definition**—Item on which to filter. • **Data type**—Integer.

[NameOfCustomDataDimension]

Note: A method is generated for every custom data dimension that is created for a data set.

Characteristic	Description
Icon	
Availability	When a custom data dimension is created for a data set.
Returns	A data set subset with data from only the custom data dimension.
Syntax	**NameOfDataSet.dimension(ValueOfDimension)**
Parameters	• **Parameter—dimension**. • **Definition**—Name of the custom dimension on which to filter. • **Data type**—String or Integer, depending on the data type of the custom data dimension.

Examples—Data set methods in rule expressions

The examples use the following study object names:

- **Study event**—Visit1
- **Form**—Vitals
- **Item**—Pulse
- **Data set**—VitalSigns

Structure

You must use the following structure when you use a method in a rule expressions.

Structure for using a method in a rule expression

1—Name of the data set.

2—Name of the method.

3—Filtering information created when you provide a value for the parameter or parameters of the method.

To use a method in a rule expression:

1 Double-click the method.

If the method has one or more parameters, the Invoke Function dialog box appears.

2 Drag a study object from the **References** tab to the **Values** field to define a value for each parameter.

Examples

The following table provides examples of rule expressions that you can use to obtain a subset of a data set. The examples use the study object names defined for the examples.

Rule expression	Returns a data set subset with data from..
VitalSigns.StudyEvent(StudyEvents.Visit1)	Visit1 only.
VitalSigns.StudyEvent(StudyEvents.Visit1).Form(Forms.Vitals)	The Vitals form in Visit1 only.
VitalSigns.StudyEvent(StudyEvents.Visit1).Form(Forms.Vitals).Item(Items.Pulse)	The Pulse item on the Vitals form in the Visit1 study event.
VitalSigns.Form(Forms.Vitals)	The Vitals form in any study event.

Rule expression	Returns a data set subset with data from..
VitalSigns.StudyEvent(StudyEvents.Visit1).Item(Items.Pulse)	The Pulse item on any form in the Visit1 study event.

Additional study objects in the Data Mappings tab

In addition to rule model properties and methods, the Data Mappings tab displays study events, forms, sections, and items that are part of the data set that is expanded in the References section. Study objects appear if they are selected as standard data dimensions of a data set. For example, if you select the Form standard dimension, the Forms node appears under the data set and under the Form(Forms) method.

You can use the study events, forms, sections, and items to define values of parameters for methods. When you double-click a method to use it in the rule expression, the Invoke Function dialog box appears, prompting you to define values for the parameters of the method. You can drag a study object to the Value field for each parameter.

Study events

Characteristic	Description
Icons	 —For a collection of study events.  —For individual study events. **Note:** The collective lists of study objects cannot be used in a rule expression. You can use only individual study objects in a rule expression.
Availability	All study events that are part of the expanded data set appear in the list. Available as: • Properties of data series. • Properties of data set methods.
Description	Use individual study events to define the values of parameters for methods.

Forms

Characteristic	Description
Icons	 —For a collection of forms.  —For individual forms. **Note:** The collective lists of study objects cannot be used in a rule expression. You can use only individual study objects in a rule expression.
Availability	All forms that are part of the expanded data set appear in the list. Available as: • Properties of data series. • Properties of data set methods.
Description	Use individual forms to define the values of parameters for methods.

Sections

Characteristic	Description
Icons	 —For a collection of sections.  —For individual sections. **Note:** The collective lists of study objects cannot be used in a rule expression. You can use only individual study objects in a rule expression.
Availability	All sections that are part of the expanded data set appear in the list. Available as: • Properties of data series. • Properties of data set methods.
Description	Use individual sections to define the values of parameters for methods.

Items

Characteristic	Description
Icons	 —For a collection of items.  —For individual items. **Note:** The collective lists of study objects cannot be used in a rule expression. You can use only individual study objects in a rule expression.
Availability	All items that are part of the expanded data set appear in the list. Available as: • Properties of data series. • Properties of data set methods.
Description	Use individual items to define the values of parameters for methods.

CHAPTER 6

Methods, operators, and literals

In this chapter

Methods

A method is a block of code that is called by a rule and that is used to manipulate data.

You can use standard C# methods, including math methods, as well as methods that are automatically generated for certain study objects, to create rule expressions. For example, you can use a count method to return the number of instances created for a repeating form.

You can use any method in a rule expression.

Math methods

A math method performs a mathematical operation on a value or set of values and can be used in a rule expression. Math methods are provided by the Microsoft .NET framework.

Note: When you use a math method in a rule expression, you must precede it with **Math.** (Math followed by a period). For example, the Pow method, which is used to return a value raised to a specific power, becomes **Math.Pow**.

For a comprehensive list of math methods, see a C# programming guide.

Abs

Characteristic	Description
Returns	Absolute value of a specified number.
Return type	Data type of the parameter that you enter.
Syntax	**Math.Abs(a)**
Parameters	• **Parameter**—a • **Definition**—A number for which an absolute value is to be found. • **Data type**—Any numeric data type.

Ceiling

Characteristic	Description
Returns	Smallest integer greater than or equal to the specified number.
Return type	Int32 (for Central Designer Integer and YesNo types).
Syntax	**Math.Ceiling(a)**
Parameters	• **Parameter**—a • **Definition**—A number for which the smallest integer greater than or equal to it is to be found. • **Data type**—Any numeric data type.

DivRem

Characteristic	Description
Returns	Quotient of two numbers and also the remainder in an output parameter.
Return type	• **Quotient**—Int32 (for Central Designer Integer and YesNo types). • **Remainder**—Int32 (for Central Designer Integer and YesNo types).
Syntax	**Math.DivRem(a, b, c)**
Parameters	• **Parameter**—a • **Definition**—Dividend • **Data type**—Int32 • **Parameter**—b • **Definition**—Divisor • **Data type**—Int32 • **Parameter**—c • **Definition**—Remainder • **Data type**—Int32

Exp

Characteristic	Description
Returns	e raised to the specified power.
Return type	Double (for Central Designer float types).
Syntax	**Math.Exp(a)**
Parameters	• **Parameter**—a • **Definition**—Number specifying a power. • **Data type**—Any numeric data type.

Floor

Characteristic	Description
Returns	Largest integer less than or equal to the specified number.
Return type	Int32 (for Central Designer Integer and YesNo types).
Syntax	**Math.Floor(a)**

Characteristic	Description
Parameters	• **Parameter**—a • **Definition**—A number for which the largest integer less than or equal to it is to be found. • **Data type**—Any numeric data type.

IEEERemainder

Characteristic	Description
Returns	Remainder resulting from the division of a specified number by another specified number.
Return type	Double (for Central Designer float types).
Syntax	**Math.IEEERemainder(a, b)**
Parameters	• **Parameter**—a • **Definition**—Dividend. • **Data type**—Any numeric data type. • **Parameter**—b • **Definition**—Divisor. • **Data type**—Any numeric data type.

Log

Characteristic	Description
Returns	Logarithm of a specified number.
Return type	Double (for Central Designer float types).
Syntax	**Math.Log(a)**
Parameters	• **Parameter**—a • **Definition**—A number for which a logarithm is to be found. • **Data type**—Any numeric data type.

Log10

Characteristic	Description
Returns	Base 10 logarithm of a specified number.
Return type	Double (for Central Designer float types).

Characteristic	Description
Syntax	**Math.Log10(a)**
Parameters	• **Parameter**—a • **Definition**—A number for which a logarithm is to be found. • **Data type**—Any numeric data type.

Max

Characteristic	Description
Returns	Larger of two specified numbers.
Return type	Data type of the parameter you enter.
Syntax	**Math.Max(a, b)**
Parameters	• **Parameter**—a • **Definition**—The value of the first number to compare. • **Data type**—Any numeric data type. • **Parameter**—b • **Definition**—The value of the second number to compare. • **Data type**—Any numeric data type.

Min

Characteristic	Description
Returns	Smaller of two specified numbers.
Return type	Data type of the parameter you enter.
Syntax	**Math.Min(a, b)**
Parameters	• **Parameter**—a • **Definition**—The value of the first number to compare. • **Data type**—Any numeric data type. • **Parameter**—b • **Definition**—The value of the second number to compare. • **Data type**—Any numeric data type.

Pow

Characteristic	Description
Returns	Specified number raised to the specified power.
Return type	Double (for Central Designer float types).
Syntax	**Math.Pow(a, b)**
Parameters	• **Parameter**—a • **Definition**—The number to be raised to a power. • **Data type**—Any numeric data type.
	• **Parameter**—b • **Definition**—The number that specifies a power. • **Data type**—Any numeric data type.

Round

Characteristic	Description
Returns	Value rounded to the nearest integer or specified number of decimal places.
Return type	Int32 (for Central Designer Integer and YesNo types). or Double (for Central Designer float types).
Syntax	**Math.Round(a, b)**
Parameters	• **Parameter**—a • **Definition**—A number to be rounded. • **Data type**—Any numeric data type.
	• **Parameter**—b • **Definition**—The number of decimal places (precision) in the return value. • **Data type**—Integer

Sqrt

Characteristic	Description
Returns	Square root of a specified number.
Return type	Double (for Central Designer float types).
Syntax	**Math.Sqrt(a)**

Characteristic	Description
Parameters	• **Parameter**—a • **Definition**—A number for which a square root is to be found. • **Data type**—Any numeric data type.

Truncate

Characteristic	Description
Returns	Integral part of a number.
Return type	Int32 (for Central Designer Integer and YesNo types).
Syntax	**Math.Truncate(a)**
Parameters	• **Parameter**—a • **Definition**—A number for which the integral part is to be found. • **Data type**—Any numeric data type.

Examples—Math methods in rule expressions

For example, you might want to raise a query when the value of an item is a specific distance from a known mean value. You can create constants called Mean and Range for the mean and the range from the mean value. The rule might look like this:

evaluate on Form Submission

value = (Mean + Range) >= item.Value && item.Value >= (Mean – Range)

when value is false

query "Value is out of range"

However, because the difference between the mean value and the range could be positive or negative, you must check the absolute value against the range. In that case, you can use the math method that returns an absolute value. Your rule might look like this:

evaluate on Form Submission

value = Math.Abs(Mean – item.Value) <= Range

when value is false

query "Value is out of range"

Methods for study objects

For a list of methods for repeating and non-repeating study objects, see:

- *Methods for repeating study objects* (on page 46).
- *Methods for non-repeating study objects* (on page 50).

Data set methods

Data sets can contain multiple data series, which can contain multiple items. You use data set methods to return a data set that is a subset of the original data set. Data set methods always return a data set, not a single value.

Methods for data sets appear in the Rule Wizard in the Expression tab > Data Mappings tab. The list of methods that appear is based on the selection of standard data dimensions and the creation of custom data dimensions in the definition of the data set. In addition to the data set methods listed in the following table, one method appears for each custom dimension that is created for the data set.

Standard data dimensions selected	Data set method in Data Mappings tab
Event	*StudyEvent(StudyEvents)* (on page 119)
Event and Event Index	*StudyEvent(StudyEvents,Integer)* (on page 120)
Form	*Form(Forms)* (on page 120)
Form and Form Index	*Form(Forms,Integer)* (on page 120)
Section	*Section(Sections)* (on page 121)
Section and Section Index	*Section(Sections,Integer)* (on page 121)
Item	*Item(Items)* (on page 122)

Note: You cannot use a data set method more than once in a rule expression.

Operators and literals

You can use operators:

- To create rule expressions.
- To include an expression in the parameters of a function.
- To include an expression in the "when" part of a rule.

Frequently used operators

For more information, see a C# or Java programming reference.

Operator category	Operator	Use	Description
Arithmetic	+	x + y	Adds x and y.
	-	x - y	Subtracts y from x.
	*	x * y	Multiplies x by y.
	/	x / y	Divides x by y.
	%	x % y	Returns the remainder of integer division of x and y.
Logical (Boolean and bitwise)	&	x & y	Evaluates both x and y and returns the logical conjunction ("AND") of their results. If x and y are integers, the logical conjunction is performed bitwise.
	|	x | y	Evaluates x and y and returns the logical disjunction ("OR") of their results. If x and y are integers, the logical disjunction is performed bitwise.
	^	x ^ y	Returns the exclusive or ("XOR") of their results. If x and y are integers, the exclusive or is performed bitwise.
	!	!x	Evaluates x and returns the negation ("NOT") of the result. • If x evaluates to false, it returns true. • If x evaluates to true, it returns false.
	~	~x	Evaluates x and returns the bitwise negation of the result. ~x returns a value where each bit is the negation of the corresponding bit in the result of evaluating x.
	&&	x && y	Evaluates x. • If the result is false, it returns false. • Otherwise, it evaluates and returns the results of y. Note that if evaluating y would hypothetically have no side effects, the results are identical to the logical conjunction performed by the & operator.

Operator category	Operator	Use	Description
	||	x || y	Evaluates x. • If the result is true, it returns true. • Otherwise, it evaluates and returns the results of y. Note that if evaluating y would hypothetically have no side effects, the results are identical to the logical disjunction performed by the | operator.
	()		Used to group information.
String concatenation	+	x + y	Concatenates strings.
Relational	==	x == y	• If x and y have the same value, it returns true. • Otherwise, it returns false.
	!=	x != y	Returns the logical negation of the operator ==. • If x is not equal to y, it returns true. • If x is equal to y, it returns false.
	<	x < y	• If x is less than y, it returns true. • Otherwise, it returns false.
	>	x > y	• If x is greater than y, it returns true. • Otherwise, it returns false.
	<=	x <= y	• If x is less than or equal to y, it returns true. • Otherwise, it returns false.
	>=	x >= y	• If x is greater than or equal to y, it returns true. • Otherwise, it returns false.
Member access	.	this.Value	Used to form long names.
Indexing	[ ]		Used to access array elements.
Conditional	?:	x ? y : z	• If x is true, it returns y. • Otherwise, it returns z.
Assignment	=	x = y	Assigns the value of y to x.

Frequently used literals

Literals are like constants that you can include in expressions.

Literal type	Literal	Description
Boolean	true false	A literal for true and false.
String	“xxx” “xx\”x”	A group of characters. Strings are delimited with double quotes ("). To quote within a string, use a backslash (\) to precede the double quotes.
Integer	Example: 13	A number without a decimal point or exponent.
Float	Example: 13.2	A number with a decimal point or exponent.

CHAPTER 7

Sample expressions for data-entry rules

In this chapter

Sample expressions that use operators

The following simple rule expressions show how to use operators in rule expressions.

Example:	Expression:
x is greater than y	x > y
x is less than or equal to z	x <= z
x is greater than y and x is less than or equal to z	(x > y) && (x <= z)
x is greater than y or x is less than or equal to z	(x > y) || (x <= z)
x is equal to 3 or x is equal to 6.4	(x = =3) || (x = =6.4)
x is not more than ten percent greater than y	x <= y*1.1 !(x>y*1.1)
If x is greater than y, then value is 100, else value is 200.	x>y?100:200
If x is not greater than y, then check if y is greater than z. Otherwise, return false.	!(x>y)?y>z:false x>y?false:y>z x<=y?y>z:false

Sample data-entry rule that uses the Data tab

Example 1

Characteristic	Description
Description	Create a rule named **rulCompareQTcQRS** that checks that the QTc interval (on the ECG form) is greater than (exclusive) the QRS interval. If false, issue a query on the QTc item that says **QTc must be greater than QRS. Please verify.**
Scope	ECG form

Study structure	• ECG (form)
	  • QTcInt (item)
	  • QRSDur (item)

Rule summary	`evaluate on Form Submission`
	
	`value = this.itmQTcInt.Value > this.itmQRSInt.Value`
	
	`when value is false`
	`  issue query on this.QTcInt: QTc interval must be greater than QRS interval. Please verify.`

Example 2

Description	Create a rule named **rulExclusionCheck1** that checks for a response other than **No** for a question on the Exclusion form. If the response is not **No**, fire a query. (Use the codelist item from the Data tab.)
Scope	Exc1 item

Study structure	Excl (form)
	• Exc1 (item)
	  • YesNoCodes (codelist)
	  • Yes (codelist item)
	  • No (codelist item)

Rule summary	`evaluate on Form Submission`
	
	`value = this.Value == this.clYesNoCodelist.citmclNo`
	
	`when value is false`
	`  issue query: The answer to this question indicates that the patient is not eligible for the study. Please clarify or correct.`

Sample data-entry rule that uses rule model properties

Characteristic	Description
Description	Create a rule named **rulCheckCompletionDate** that checks that a study completion date has been entered if **Yes** is selected for **Study complete?**. Issue a query when the rule evaluates to true.
Scope	Study Completion form

Study structure	CompletionStatus (form)
	• CompDate (item)
	• CompletionStatus (item)
	  • YesNoCodes (codelist)
	  • Yes (codelist item)
	  • No (codelist item)

Rule summary	`evaluate on Form Submission`
	
	`value = this.itmCompletionStatus.Value == this.itmCompletionStatus.clYesNoCodelist.citmYes ? this.itmCompDate.Empty : false`
	
	`when value is true`
	`issue query on this.itmCompletionStatus: Study completetion date cannot be empty, if study completion status is complete. Please verify.`

Sample data-entry rules that use methods

Example 1

Characteristic	Description
Description	Use a math method to round the result of the existing rule, **rulCalcBMI**, to two decimal places.
Scope	Baseline event

Study structure	• Baseline (study event)
	  • Demog (form)
	  • HT (item)
	  • Vitals (form)
	  • Weight (item)
	  • BMI (item)

Rule summary	`evaluate on Form Submission`
	
	`value =`
	`Math.Round(_CalculateBMI(this.frmDemographics.itmHeight.Val`
	`ue,this.frmVitals.itmWeight.Value),2)`
	
	`always`
	`  set this.frmVitals.itmBMI.Value = value`
	
	Note: The solution above is the simplest solution, but not the best, as it does not clear the BMI field if Height or Weight are subsequently cleared, and it will cause an ISE if Height is 0.0. A better solution follows.
	
	`evaluate on Form Submission`
	
	`value = !this.frmDemographics.itmHeight.Empty &&`
	`!this.frmVitals.itmWeight.Empty`
	`?`
	`(this.frmDemographics.itmHeight.Value != 0 ?`
	`1:`
	`(!this.frmVitals.itmBMI.Empty ?`
	`2 : 3)) :`
	`(!this.frmVitals.itmBMI.Empty ?`
	`2 : 3)`
	
	`when value == 1`
	`  set this.frmVitals.itmBMI.Value =`
	`Math.Round(_CalculateBMI(this.frmDemographics.itmHeight.Val`
	`ue,this.frmVitals.itmWeight.Value),2)`
	
	`  when value == 2`
	`set this.frmVitals.itmBMI.Empty = true`

Example 2

Characteristic	Description
Description	Create a rule named **rulTabsDispensed** that checks that the sum of tablets dispensed is greater than zero and less than 500. The sum of tablets should use 0 as a replacement value. If false, fire a query.
Scope	Dispensing Record section

Study structure	• DISPREC (section)
	  • Tabs (compound item)
	  • onemg (item)
	  • twomg (item)
	  • threemg (item)

Rule summary	`evaluate on Form Submission`
	
	`value = (this.itmTabNum.itmonemg.GetValue(0) +`
	`this.itmTabNum.itmthreemg.GetValue(0) +`
	`this.itmTabNum.itmtwomg.GetValue(0)) > 0 &&`
	`(this.itmTabNum.itmonemg.GetValue(0) +`
	`this.itmTabNum.itmthreemg.GetValue(0) +`
	`this.itmTabNum.itmtwomg.GetValue(0)) < 500`
	
	`when value is false`
	`  issue query on this.itmTabNum: The total number of tablets dispensed {TabsDisp} is not within the expected range. Please clarify or correct.`

Sample data-entry rule that uses constants

Characteristic	Description
Description	Create a rule named **rulTempRangeCheckInclusive** that checks that the Temperature item is >=36.1 and <=37.8. If not create a query: The entered temperature {EnteredTemp} {EnteredTempUnit} is outside the expected range. Use the TempMin and TempMax constants that you created in the expression. Use the EnteredValue and EnteredUnit rule model properties to construct the query text.
Scope	Temperature item on the Vitals form
Study structure	• Vitals (form) • Pulse (item)

Rule summary	`evaluate on Form Submission`
	
	`value = Value must be greater than or equal to {MinValue:Constants.TemperatureRange.TempMin}, and less than or equal to {MaxValue:Constants.TemperatureRange.TempMax}`
	
	`when value is false`
	`issue query: The entered temperature {EnteredTemp} {EnteredTempUnit} is outside the expected range of {TempMin} to {TempMax}`
	
	`OR`
	`evaluate on Form Submission`
	
	`value = this.Value >= (Constants.TemperatureRange.TempMin) && this.Value <=(Constants.TemperatureRange.TempMax)`
	
	`when value is false`
	`  issue query: The entered temperature {EnteredTemp} {EnteredTempUnit} is outside the expected range.`
	
	Note: In the first solution, the rule is defined as an intrinsic rule that uses the range check rule template. In the second solution, the rule is defined as a constraint rule that does not use a template.

Sample data-entry rules that use functions

The _GetDateDifference function uses twenty-four hour periods, not calendar days, to compute the differences in days. If the DOV item contained date and time information, there are two possible solutions. The first solution uses the complete DOV date/time and the _GetDateDifference function and therefore evaluates days as twenty-four hour periods. The alternate solution evaluates the differences between DOVs using calendar days by creating new DateTime objects that contain no Time information from the DOVs and passing those new DateTime objects to the _GetDateDifference function. Each of the solutions uses the predefined constant DateTimeParts.Days.

Example 1

Characteristic	Description
Description	Create a rule named **rulWeekSeqCk** that checks that there are 7 days between Week 1B and Week 2B visits. Issue a query on the Week 2B DOV item (in Treatment Arm B), if the DOV on the Week 2B study event is less than 7 days after the DOV on the Week 1B study event.
Scope	Treatment Arm B study element

Study structure	• Week1
	  • frmDOV (form)
	  • DOV (item)
	• Week2
	  • frmDOV (form)
	  • DOV (item)

Rule summary	`evaluate on Form Submission`
	
	`value = _GetDateDifference`
	`(this.evtWeek1B.frmDOV.sctDOV.itmDOV.Value,`
	`this.evtWeek2B.frmDOV.sctDOV.itmDOV.Value,`
	`Constants.DateTimeParts.Days)>=7`
	
	`when value is false`
	`  issue query on this.evtWeek2B.frmDOV.sctDOV.itmDOV: Date of Visit is less than a week after the Date of Visit given at Week 1. Please clarify or correct.`

Characteristic	Description
Alternate rule summary	`evaluate on Form Submission` `value = _GetDateDifference (new DateTime(this.evtWeek1B.frmDOV.sctDOV.itmDOV.Value.Year, this.evtWeek1B.frmDOV.sctDOV.itmDOV.Value.Month, this.evtWeek1B.frmDOV.sctDOV.itmDOV.Value.Day, 00, 00, 00), new DateTime(this.evtWeek2B.frmDOV.sctDOV.itmDOV.Value.Year, this.evtWeek2B.frmDOV.sctDOV.itmDOV.Value.Month, this.evtWeek2B.frmDOV.sctDOV.itmDOV.Value.Day, 00, 00, 00), DateTimeParts.Days) >= 7`  `when value is false` `issue query on this.evtWeek2B.frmDOV.sctDOV.itmDOV: Date of Visit is less than a week after the Date of Visit given at Week 1. Please clarify or correct.`

Example 2

Characteristic	Description
Description	Create a rule named **rulDateCompareWithRange** that checks that the DOV in Week 2A is within 6-8 days after the DOV in Week 1A (in Treatment Arm A).
Scope	Treatment Arm A study element

Structure	• Week1A
	  • frmDOV (form)
	  • DOV (item)
	• Week2A
	  • frmDOV (form)
	  • DOV (item)

Rule summary	`evaluate on Form Submission`
	
	`value = _CompareDatesWithRange(this.evtWeek1A.frmDOV.sctDOV.itmDOV.Value,this.evtWeek2A.frmDOV.sctDOV.itmDOV.Value,Constants.DateTimeParts.Days,6,8)`
	
	`when value is false`
	`issue query on this.Week2A.frmDOV.sctDOV.itmDOV: The date of the Week 1A visit and the date of the Week2A visit are not within 6-8 days of each other. Please clarify or correct.`

Characteristic	Description
Alternate rule summary	evaluate on Form Submission value = _CompareDatesWithRange(this.evtWeek1A.frmDOV.sctDOV.itmDOV.Value,this.evtWeek2A.frmDOV.sctDOV.itmDOV.Value,Constants.DateTimeParts.Days,6,8) when value is false issue query on this.Week2A.frmDOV.sctDOV.itmDOV: The date of the Week 1A visit and the date of the Week2A visit are not within 6-8 days of each other. Please clarify or correct.

Sample calculation rules

Example 1

Characteristic	Description
Description	Using the CalcAge function, create a rule named **rulCalcAge** that calculates a subject's age and populates the age field. Use the DOV from the visit containing the DEM form.
Scope	Baseline study event

Study structure	• Baseline (study event)
	  • frmDOV (form)
	  • DOV (item)
	  • Demog (form)
	  • DOB (item)
	  • AGE (item)

Rule summary	`evaluate on Form Submission`
	
	`value = !this.frmDOV.sctDOV.itmDOV.Empty &&`
	`!this.frmDemographics.itmDOB.Empty ? 1`
	`:(!this.frmDemographics.itmAGE.Empty ? 2:3)`
	
	`when value == 1`
	`  set this.frmDemographics.itmAGE.Value =`
	`  _CalcAge(this.frmDemographics.itmDOB.Value,`
	`  this.frmDOV.sctDOV.itmDOV.Value)`
	
	`when value == 2`
	`  set this.frmDemographics.itmAGE.Empty = true`

Example 2

Characteristic	Description
Description	Create a rule named **rulBMIRangeCheck** that checks that the BMI is between 18.5 and 30. Specify an action to issue a query if the BMI is out of range.
Scope	Baseline event

Study structure	• Baseline (study event)
	  • Vitals (form)
	  • Weight (item)
	  • BMI (item)
	  • Demog (form)
	  • HT (item)

Characteristic	Description

Rule summary	`evaluate on Form Submission`
	
	`value = (this.Value >= Constants.BMIRange.BMILow)&&`
	`(this.Value <= Constants.BMIRange.BMIHigh)`
	
	`when value is false`
	`issue query on this: BMI {CalculatedBMI} is out of`
	`range. Please verify.`

Sample data-entry rules that use mappings

Example 1

Characteristic	Description
Description	Create a rule called **rulCompletionLaterThanDOV** that checks whether the completion date is later than any of the visit dates entered. If not, issue a query.
Scope	Global; rule is on the Study Completion form, using data from every instance of the DOV form.

Study structure	• fmStudyCompletion (form)
	  • itmCompletionDate (item)
	• fmDOV (form in each study event)
	  • itmDOV (Item)
	• VisitDates (mapping)
	  • VisitDatesDataSet (data set)
	  • DOVs (data series with itmDOV item mapped using the Always mapping type so the item is available in every form and every study event where it occurs)

Rule summary	`evaluate on Form Submission`
	
	`value = _IsValueGreaterThanOrEqualToArray (this.Value,this.VisitDatesDataSet.DOVs.Values)`
	
	`when value is false`
	`  issue query: Termination Date must be later than any visit date.`

Example 2

Characteristic	Description
Description	Create a rule called **rulHemoglobinRange** that checks whether the entered hemoglobin range is between 140 and 180 for males or between 120 and 160 for females. If not, issue a query.
Scope	Global; rule is on the Hematology form, using data from the Demographics form.

Characteristic	Description

Study structure	• fmHematology (form)
	  • itmHgb (item)
	• fmDemog (form)
	  • itmGender (item)
	• Demog (mapping)
	  • DemogDataSet (data set)
	  • GenderDataSeries (data series with itmGender item mapped)

Rule summary	`evaluate on Form Submission` `value = (this.DemogDataSet.GenderDataSeries.Value == this.DemogDataSet.GenderDataSeries.GenderCodes.Female && this.Value >= Constants.HgbRanges.HgbLowF && this.Value <=Constants.HgbRanges.HgbHighF) || (this.DemogDataSet.GenderDataSeries.Value == this.DemogDataSet.GenderDataSeries.GenderCodes.Male && this.Value >=Constants.HgbRanges.HgbLowM && this.Value <= Constants.HgbRanges.HgbHighM)`  `when value is false` ` issue query: This {EnteredValue} is outside valid hemoglobin range for {EnteredGender}s.`

APPENDIX A

Option descriptions

In this appendix

Rule expressions

New Rule Template dialog box—Option descriptions

Option	Description
Properties tab	
Name	Name of the rule template.
Classification	User-defined term used to organize rule templates.
Description	Description of the rule template.
Display Text	Text that appears in the Rule Summary section of the Rule wizard after the **When Value Is** information. If this field is blank, the contents of the Expression workspace are used. If the expression contains parameters, the name of the parameter and the value of the parameter appear. For example, if the expression is **value must be between {a} and {b}**, and the value of **a** is 10 and the value of **b** is 100, the parameters appear as **a:10** and **b:100**.
Definition tab	
Return Type drop-down list	Return type of the rule template; one of the following: Integer, Float, String, Boolean, Date/Time, or Array (A list of values, all of the same type).
Expression	Expression of the rule. For more information, see *About the rule expression language* (on page 2).
Parameters (Optional)	
Parameter	Name of the parameter.
Data Type	Return type of the parameter; one of the following: Integer, Float, String, Boolean, Date/Time, or Array (A list of values, all of the same type).
Default Value	Specified value of the parameter.
References	
Data tab	Lists study objects in the scope of the rule. Optionally, to view the rule model properties of all of the study objects, select **Show all**.
Functions tab	Lists functions registered in a study and the libraries that appear in the Libraries List in the Study Editor. Any rule in the study can reference a function.
Constants tab	Lists constants created in the study and the libraries that appear in the Libraries List in the Study Editor. Any rule in the study can reference a constant.

Option	Description
Data Mappings tab	Lists: • RefNames of the data mappings, data sets, and data series in the study or library. • Rule model properties for data series. A data series has the properties of the item that is mapped to it. If a data series contains an item that collects more than one value, the rule model properties for repeating study objects appear so you can access an array of all of the values of the item. • Methods for data sets. A method appears if you select the corresponding standard data dimension of the data set. You can use data set methods to return a subset of the data in the data set. • Study events, forms, sections, and items that are mapped to each data set. • Study objects appear if you select the corresponding standard data dimension of the data set. The properties of the study objects are used as parameters of data set methods.

Rule Templates tab—Option descriptions

The grid displays rule templates created on the study object selected in the Project Explorer.

Option	Description
Rule templates grid	
Data Type	Return type of the rule template.
Description	Description of the rule template.
Display Text	Text that appears in the Rule Summary section of the Rule wizard after the **When Value Is** information.
Id	Identification information for the rule.
Expression	Rule expression of the rule template.
Parameters	Specified parameters of the rule template, separated by semicolons.
Template Name	Name of the rule template.
Type	Indicates that the rule was created using the Central Designer rule expression language.

Functions

Functions tab on the Study and Library Information Explorer bars

The Functions tab displays imported, user-defined functions that you can use with rules and rule templates in a study or library. Functions defined in the Functions tab are available study- or library-wide, regardless of the scope of any rules.

A function is a reusable piece of code that extends the behavior of a rule. Many predefined functions are available by default in the System Library.

The Functions tab is available in the following locations:

- Study Information Explorer bar > study node.
- Library Information Explorer bar > library node.

For more information, see **Functions tab - Option descriptions** (on page 156).

Functions tab—Option descriptions

Option	Description
Fields	**Note:** Not all fields appear in the default view. Optionally, you can add the other fields to the browser view, and you can rearrange the order of fields. For more information, see ***Showing and hiding a field*** in the *User Guide*.
Class	The class in which the function is included.
Classification	User-defined text used to group, sort, and filter functions.
Data Type	Return type of the function, either Integer, Float, String, Boolean, Date/Time, or Array (A list of values, all of the same type).
Description	Description of the function.
Filename	Name of the assembly file that contains the function.
ID	Unique identifier for the function.
Language	Language in which the function was written.
Name	Name of the function.
Parameters	Parameters for the function.
Published (only in libraries)	Indicates that the study object has been published.
Title	Title of the function.
Type	Function type. All functions are .NET class functions.
Toolbar buttons	
Import function	Import an existing function.
Edit	Open the selected function to view or edit it.

Option	Description
Delete	Delete the selected function.
Publish (only in libraries)	Publish the selected function. This button is enabled for library objects that have been saved.

Edit Function dialog box—Option descriptions

Option	Description
Properties tab	
Name	Name of the function.
Classification	User-defined text used to group, sort, and filter functions.
Description	Description of the function.
Definition tab	
Return Type	Data type for the returned value for the function. One of the following: Integer, Float, String, Boolean, Date/Time, or Array (A list of values, all of the same type).
Assembly	Name of the assembly file that contains the function.
Browse	Locate the assembly for the function.
Class	The class in which the function is included.
Parameters section	In the parameters section, you can provide one or more parameters for the function. **Note:** Not all fields appear in the default view. Optionally, you can add the other fields to the browser view, and you can rearrange the order of fields. For more information, see ***Showing and hiding a field*** in the *User Guide*.
Data Type, Default Value, Description, Parameter	Data type, default value, description, and name of the parameter.

Constants

Constants tab on the Study and Library Information Explorer bars

In the Constants tab of the Study and Library Editors, you can define constants to be used in rules in the study or library. Constants defined in the Constants tab are available study- or library-wide, regardless of the scope of any rules.

A constant is a value that is defined in a library or study and that can be referenced by any rule.

For more information, see **Constants tab - Option descriptions** (on page 158).

Constants tab—Option descriptions

Option	Description
Fields	**Note:** Not all fields appear in the default view. Optionally, you can add the other fields to the browser view, and you can rearrange the order of fields. For more information, see ***Showing and hiding a field*** in the *User Guide*.
Classification	User-defined text used to group, sort, and filter constants.
Data Type	Return type of the constant: • **Integer**—Contains only numbers. • **Float**—Contains numbers and a decimal point. • **String**—Contains alphanumeric characters. • **Boolean**—True or False. • **Date/Time**—Contains date and time information.
Description	Description of the constant.
Name	Name of the constant. The name must: • Begin with a letter. • Contain only letters and digits. • Be unique.
Published (only in libraries)	Indicates that the study object has been published.
Value	Value of the constant.
Toolbar buttons	
New Constant Delete	Create and delete a constant.
Publish (only in libraries)	Publish the selected constant. This button is enabled for library objects that have been saved.

New Constant dialog box—Option descriptions

Option	Description
Name	Name of the constant.
Data Type	Data type of the constant, such as Integer, Float, String, Boolean, or Date/Time.
Value	Value of the constant. The value must: • Follow C# variable standards. • Start with a letter or an underscore. • Contain only letters, numbers, and underscores.
Classification	(Optional) User-defined classification of the constant.
Description	(Optional) Description of the constant.

Glossary

A

annotated study book

A form-by-form summary of the design of a study. Optionally, it includes a time and events schedule, a preview of each form, and selected annotations that list design details.

See also *study book* (on page 167).

application role

A role associated with administrative activities.

arm

See *study arm* (on page 167).

authentication

The method of ensuring that you are using the correct user name and password to log on.

authorization

The method of giving users access to information or functionality. Access is controlled using rights, roles, and teams.

B

baseline

A snapshot of all components in a study. Validation creates a baseline.

Baselines Browser

A browser in which you view the results of validation and make temporary baselines public so that other users can work with them.

branch

See *study branch* (on page 167).

C

calculation rule

A rule that sets the value of an item based on a calculation.

catalog

A collection of categories and keywords that can be attached to users and study objects to facilitate searching in the Libraries Browser and Users Browser.

catalog administration

The process of creating keywords and categories and assigning them to users and study objects for faster and more sophisticated searching.

category

A hierarchical grouping of keywords. You can create categories only from existing keywords.

CDISC

Clinical Data Interchange Standards Consortium. CDISC is an open, multidisciplinary, non-profit organization committed to the development of industry standards to support the electronic acquisition, exchange, submission and archiving of clinical trials data and metadata for medical and biopharmaceutical product development.

checkbox

A type of data entry control in which you can select one or more options by selecting the box that represents each option.

clinical project

See *study project* (on page 168).

clinical protocol

See ***protocol*** (on page 166).

clinical study

See ***study*** (on page 167).

codelist

A collection of code-label pairs that gather together the entry choices for an item. A code-label pair consists of a single code (the value that is used for analysis) and a label (the value that is visible to users).

See also ***codelist item*** (on page 162).

codelist item

A code-label pair consisting of a single code (the value that is used for analysis) and a label (the value that is visible to users). Multiple codelist items make up a codelist.

See also ***codelist*** (on page 162).

coding

The process of selecting terms and codes from a dictionary for a verbatim

coding dictionary

A standardized collection of terms and the codes that correspond to those terms.

coding map

A study object that contains the necessary information to code an item.

coding target

See ***target item*** (on page 168).

collaboration

The process by which users with different roles and specialties can work together to create, validate, and deploy a study.

collaboration note

A note that you attach to any study object.

collaboration note type

A classification used to identify the type and purpose of a collaboration note.

Collaboration Notes Browser

A browser in which you work with collaboration notes.

common form

A form that is designed for use with multiple study events. The same data appears in the form in all study events in which the form is used.

component

Any design building block that is configured in a study or library. Design components include study objects (such as a project, study, study element, study event, form, or item) as well as rules, individual items selected from drop-down lists, and controls (such as checkboxes and radio buttons).

compound item

An item that has one or more child items that can have different data types.

constant

A value that is defined in a library or study and that can be referenced by any rule.

constraint rule

A rule that checks whether data is valid. Constraint rules are used to confirm that clinical data meets the requirements of the clinical protocol.

container

A node in the Project Explorer that contains zero or more study objects or components.

context item

An item that provides additional coding information, such as the indication and route of administration for drugs, that can be displayed with an item coded using the WHO-DD dictionary.

CSML

Clinical Study Markup Language. CSML is an XML-based markup language developed by Oracle for representing and exchanging clinical data definitions created in the Central Designer application.

See also *MedML* (on page 165).

custom data dimension

See *data dimension* (on page 163).

custom property

A user-defined or default characteristic of a study object.

D

data dimension

A key item for a data set. A data dimension specifies the additional information that will be saved when study data is collected. You can specify standard data dimensions (Study, Subject, Event and Event Index, Form and Form Index, and Item) and custom data dimensions.

data series

A grouping of one or more items with the same clinical meaning, such as one or more items that measure weight.

See also *data set* (on page 163), *mapping* (on page 165).

data set

A grouping of one or more related data series.

See also *data series* (on page 163), *mapping* (on page 165).

data type

An attribute for items and data series. For an item, the data type determines the type of entry an item will accept. For a data series, the data type determines which items can be added to it. Data types include date time, integer, float, and text.

data-entry rule

A rule that checks whether data is valid or that sets the value of an item based on a calculation.

See also *workflow rule* (on page 169).

date time item

An item used to collect date and time information on a form.

deployment

The process of sending a study to a target application. To collect data, a study must be deployed into a target application as a complete deployment package.

dictionary metadata item

An identifier that describes administrative data about a dictionary and that you can use to create a coding map.

dictionary type

A name or identifier for the metadata for a dictionary.

drop-down list

A data entry format in which you select an option from a list.

dynamic form

A form that is automatically generated in the InForm application when subject data satisfies certain criteria tested in another form.

dynamic visit

A visit that is automatically generated in the InForm application when patient data satisfies certain criteria tested in another visit.

E

edit check

A data-entry rule that checks whether entered data is valid.

See *data-entry rule* (on page 163).

element

See *study element* (on page 167).

event

See *study event* (on page 167).

explicit lock

A lock that you request and that does not expire.

See also *implicit lock* (on page 164).

expression

The part of a rule that specifies what to evaluate.

F

field

The area in a data-entry window where the value for an item is entered or displayed.

float item

An item used to collect numerical values with decimal points.

form

A container for one or more items. A form can contain one or more sections and supports multiple locales and layouts. A form is deployed to a target application as a data-entry form used to collect subject information and other clinical data.

full installation deployment package

A deployment package that contains everything needed to deploy a complete study.

function

A reusable piece of code that extends the behavior of a rule. A function can be predefined or user-defined.

G

global condition

A logical construct that, when applied to a study object, determines whether the study object will appear for a particular subject. A global condition does not affect other study objects in the workflow.

See also *workflow rule* (on page 169).

globals

Study objects and properties that are related to mappings.

grouping

A default or user-provided value used to organize custom properties of a study object.

I

implicit lock

A type of lock used when you edit a study object. An implicit lock is automatically applied when you select or open a study object and is automatically released when you close or save a study object.

See also *explicit lock* (on page 164).

incremental deployment package

A deployment package that contains a complete study based on a previously created deployment package, plus any additions or changes.

integer item

An item used to collect a numerical value without a decimal point.

internationalization

The process of configuring a study for translation into different languages or for different regional requirements.

intrinsic rule

A constraint rule or calculation rule based on a predefined rule template.

item

A study object used as a container for the collection of clinical data.

J

Job Log Browser

A browser in which you view the results of asynchronous jobs, such as validation or import.

K

keyword

An identifier that is associated with users and study objects to facilitate more powerful and efficient searches.

L

Libraries Browser

A browser in which you search the repository for study objects and then add them to studies or libraries.

library

A container used to store related study objects and templates to be published for reuse in studies or other libraries. A library provides a view of the study objects in the repository.

See also *repository* (on page 166).

Library List

A hierarchical list of libraries from which resources can be used. The hierarchy determines the order in which libraries are searched. The Library List is defined for each study in the Study Editor.

library project

A project containing a library.

library role

A role associated with library activities.

library team

A group of users who have rights granted by a certain role to perform tasks in a particular library.

locale

A supported language or language variation.

localization

The process of designing a study for a specific locale.

locked

A state in which only the user who created the lock can modify a study object. Locks can be implicit or explicit.

See also *implicit lock* (on page 164), *explicit lock* (on page 164).

M

mapping

A data grouping that provides an alternate data view of a study. Mappings were previously called logical schemas.

See also *data series* (on page 163), *data set* (on page 163).

MedML

An XML-based markup language developed by Oracle for representing and exchanging clinical data definitions created in the InForm application..

See also *CSML* (on page 162).

method

A block of code that is called by a rule and that is used to manipulate data.

N

normalization

The process of converting data to a required format.

O

object

See *study object* (on page 167).

ODM

Operational Data Model. ODM is an XML-based standard developed by the Clinical Data Interchange Standards Consortium (CDISC) for representing and exchanging clinical data.

P

precondition

The part of a rule that specifies when to evaluate the rule expression.

project

See ***library project*** (on page 165) and ***study project*** (on page 168).

Project Explorer

A browser that displays a view of the open project and the study objects it contains.

Properties Browser

A browser in which you can view and modify the properties of the study object selected in the Project Explorer.

property

A defining characteristic of a study object.

protocol

A detailed plan that describes how investigators conduct a study. The clinical protocol sets the guidelines for the study, describes the conditions of the study, and contains a set of forms on which clinical data is collected.

publish

The action that makes a study object created in a library available to other users.

See also ***unpublish*** (on page 169).

Q

query

A text string that appears on a CRF item in the InForm application when a rule on that item fails. When designing a rule in the Central Designer application, you can specify the query text and the circumstances under which a rule results in a query.

R

radio button

A type of data entry format in which you must select a single item from a list of choices.

reference

A text note, a link to a Web page or file (URL), a document, or a combination of all three, that is attached to a study project for users to consult during the development of a study.

RefName

A unique identifier for a study object.

repeating form

A type of form for which the *Repeating* property is set to true. You use a repeating form to collect multiple instances of the same data at different dates and times.

repository

A single database instance that contains all Central Designer study objects, components, and users.

revision

An audit history record that is created automatically when a user edits a study object and saves the changes.

right

A predefined permission that controls access to a specific feature or activity in the Central Designer client or Central Designer Administrator client and that can be assigned to one or more roles.

See also ***role*** (on page 166).

role

A collection of rights. When a user is assigned to a role, the rights associated with the role are granted to the user.

See also ***library role*** (on page 165), ***study role*** (on page 168), and ***user role*** (on page 161).

role administration

The process of managing tasks that users perform in the Central Designer and Central Designer Administrator applications, assigning rights to roles, and assigning roles to users.

rule

See *data-entry rule* (on page 163), *workflow rule* (on page 169).

rule action

The action, or actions, that takes place as a result of the evaluation of a rule expression.

rule scope

The set of study objects that a rule can reference. The scope of a rule is determined by the study object on which the rule is defined.

rule template

A function that is defined on a study object, study object template, or study object type and can be used as the expression clause of a rule.

rule type

See *calculation rule* (on page 161), *constraint rule* (on page 162), and *intrinsic rule* (on page 164).

S

SDTM

Study Data Tabulation Model. SDTM is a CDISC model used to standardize data structures in data extracts.

shared form

See *common form* (on page 162).

site

A location that participates in a study.

standard data dimension

See *data dimension* (on page 163).

standard task type

A task type that is typically used for all non-translation tasks assigned to an individual or team.

study

The definition of the workflow, data-entry, and data-management system for a clinical study.

study arm

The CDISC term for a study branch consisting of a planned sequence of study elements. A study arm is typically equivalent to a treatment group.

study book

The set of forms used to collect clinical data.

See also *annotated study book* (on page 161).

study branch

A path for which data is collected for certain subjects. A study can contain multiple branches as different conditions are assessed, and a branch is followed depending on the subject and other circumstances.

study design

A container for the structure of a study.

study element

The CDISC term for a basic building block of a study. A study element represents a segment of a study and can consist of one or more study events. Study elements are optional.

study event

A subject evaluation checkpoint when data is collected. Study events usually correspond to visits, but one visit can span multiple study events.

study object

A study building block that appears in the Project Explorer. Study objects include study projects, library projects, studies, libraries, study elements, study events, forms, sections, items, codelists, and codelist items.

See also *component* (on page 162).

study object editor

An editor for each study object, such as a project, study, study element, study event, form, or item. A study object editor appears in the workspace when you select a study object.

study project

A project containing one or more studies that are related to each other.

study role

A role associated with study activities.

study team

A group of users who perform tasks granted by a certain role for a particular study.

study workflow

See *workflow* (on page 169).

subject

An individual who participates in a clinical study.

system

An application to which you deploy a study from the Central Designer application.

system configuration administration

The process of creating, configuring, and managing internationalization, collaboration, and customization information using the Central Designer Administrator application.

T

target item

An item that holds a term, code, or additional information after a verbatim is coded.

task

A request that you attach to a study object and assign to an individual or a study team.

task classification

The classification of a task. You can choose either standard (used for all non-translation tasks assigned to an individual or team) or translation (used for tasks that request translation of a study object into one or more languages). You define the classification for task types in the Central Designer Administrator application.

task type

A classification used to identify the type of the task and the way the task is used.

Tasks Browser

A browser in which you work with tasks.

team

See *study team* (on page 168), *library team* (on page 165).

template

A study object that is either partially or fully defined and that can be used to create other study objects. You can create templates for study projects, studies, study elements, study events, forms, items, codelists, and mappings.

text box

A data-entry format in which you type data.

text item

An item used to collect alphanumeric information.

translation task type

A task type that is used for tasks that request

translation of a study object into one or more languages.

type

A study object that is either partially or fully defined and can be used to create other study objects. Types are like templates except that types appear as options in the Actions menu and in the Project Explorer menu when you create a new study object.

U

unpublish

The action that makes a study object in a library no longer available to other users.

See also *publish* (on page 166).

user

A person who works in the Central Designer or Central Designer Administrator application.

user administration

The process of managing users.

Users Browser

A browser in which you search the repository for users and then add them to study teams or library teams.

V

validation

The process of checking the status of a study to indicate if the study is ready for deployment. The study validation process determines whether all essential components exist and are consistent.

verbatim

The original reported text that describes the adverse event, disease, drug, or other item to be coded in the Central Coding application.

verbatim type

A classification of a verbatim as defined in a coding dictionary.

version

An explicitly requested audit history record for a study object.

visit

See *study event* (on page 167).

W

workflow

The progression of work for a study, as determined by the study designers.

workflow rule

A logical construct that tests data values to determine the study element, study event, or form to which a subject progresses next. A workflow rule prevents study objects in the workflow from appearing until the rule is evaluated.

See also *data-entry rule* (on page 163), *global condition* (on page 164).

workspace

The work area of the Central Designer and Central Designer Administrator applications. The contents of the workspace depend on the type of activity you are performing and the rights that you have been granted. The workspace displays the editor for the study object that is selected in the Project Explorer.

Y

yes no item

An item used to collect yes or no answers to questions. A yes no item contains a predefined codelist with Yes and No options.

Index

S

T

U

V

Y