# Business Rules Engine

Last Updated:2026.02.25

© Copyright 2000–2026 salesforce.com, inc. All rights reserved. Salesforce is a registered trademark of salesforce.com, inc., as are other names and marks. Other marks appearing herein may be trademarks of their respective owners.

# CONTENTS

Business Rules Engine . . . . . 1
Get Started with Business Rules Engine . . . . . 4
Learn and Explore . . . . . 5
Plan and Prepare for Your Business Rules Engine Implementation . . . . . 10
Business Rules Engine Setup . . . . . 25
Decision Matrices . . . . . 29
Decision Tables for Business Rules Engine . . . . . 59
Expression Sets . . . . . 110
Decision Explainer . . . . . 174
Business Rules Engine Integrations . . . . . 189
Calculation Matrix and Procedure Migration to Business Rules Engine . . . . . 222
Business Rules Engine Use Cases . . . . . 226
Business Rules Engine Default Limits . . . . . 250

# Business Rules Engine

Simplify determining eligibility or qualification and automate complex decision-making to find solutions faster using Business Rules Engine. The key components of Business Rules Engine are *expression sets* and *lookup tables*. *Decision matrices* and *decision tables* together constitute *lookup tables*.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Use a Business Rules Engine component or a combination of components to design business rules to provide automated business solutions in a workflow created using any tool. Business Rules Engine is a suite of services, components, and objects that can be used to create business rules that perform complex lookups and calculations.

Business Rules Engine components are available as part of Flows and Omnistudio. Also, you can use Connect APIs to integrate Business Rules Engine with any workflow created using custom or third-party tools. The workflow that you use to gather inputs passes the inputs to Business Rules Engine, which then returns the outcome to the workflow that called the Business Rules Engine component.

## How Business Rules Engine Components Integrate with Workflow Tools

Here's how the components in Business Rules Engine integrate and interact with workflows that are created using various tools and processes.

| Workflow Tool or Process | Business Rules Engine Integration |
|---|---|
| Flow | Expression sets, decision matrices, and decision tables, which are components of Business Rules Engine, are available as actions in Flow. Invoke decision matrices and decision tables as part of an expression set that you invoke in a flow, or independently in flows. |
| Omnistudio | Expression sets and decision matrices are available as actions in Omnistudio. If Business |

| Workflow Tool or Process | Business Rules Engine Integration |
| --- | --- |
| | Rules Engine is available for your cloud and you also have Omnistudio enabled, you can call Expression Set Action and Decision Matrix Action from Integration Procedures and OmniScripts. |
| Custom or third-party tools | Use the Connect APIs in Business Rules Engine to invoke expression sets and lookup tables in workflows that you create in custom or third-party tools. |

### Get Started with Business Rules Engine

Business Rules Engine lets you build powerful, scalable lookups and calculations to automate complex decisions. Rules created with Business Rule Engine are easy to maintain, and integrate directly with Flow, Omnistudio, and third-party workflow tools using connect APIs.

### Learn and Explore

Learn how Business Rules Engine can help your business and customers. Understand its core functionality, create a trial org, and learn how to get started.

### Plan and Prepare for Your Business Rules Engine Implementation

Explore some key concepts and considerations before you implement Business Rules Engine in your org.

### Business Rules Engine Setup

Configure your users to use Business Rules Engine and assign permission sets so that the users can access expression sets and lookup tables.

### Decision Matrices

A decision matrix matches input values to a table row and returns the row's output values. Expression sets and various digital procedures can call decision matrices. Decision matrices accept JSON input from, and return JSON output to the digital processes that call the matrices. Decision matrices are useful for implementing complex rules in a systematic, readable manner.

### Decision Tables for Business Rules Engine

Decision tables are complex lookup tables that read business rules with multiple inputs, and return multiple outputs for records in your Salesforce org, or for the values that you specify. Rules are records in a standard object, custom object, or a custom metadata type.

### Expression Sets

Expression sets–the calculation engine of business rules–consist of a series of steps connected in a logical flow built from variables, constants, conditions, calculations, lookups, and aggregations. Expression sets accept JSON input from and return JSON output to the digital processes that call the sets.

### Decision Explainer

Decision Explainer is a service that provides insights into why a business rule generated a specific result.

### Business Rules Engine Integrations

Expression sets and lookup tables–the Business Rules Engine components–can be called from Flow

and Omnistudio. You can also use Business Rules Engine Business APIs to call these components from workflows that are created by using third-party tools.

### Calculation Matrix and Procedure Migration to Business Rules Engine

To use calculation procedures and calculation matrices from Omnistudio as Business Rules Engine components, migrate these Omnistudio components as expression sets and decision matrices.

### Business Rules Engine Use Cases

Explore various use cases that shows how Business Rules Engine is used across different clouds.

### Business Rules Engine Default Limits

Review the default limits for Business Rules Engine components and their usage. To raise any default limits, contact Salesforce Customer Support.

# Get Started with Business Rules Engine

Business Rules Engine lets you build powerful, scalable lookups and calculations to automate complex decisions. Rules created with Business Rule Engine are easy to maintain, and integrate directly with Flow, Omnistudio, and third-party workflow tools using connect APIs.

**Get Oriented**

What Is Business Rules Engine?

Video: Watch Business Rules Engine in Action

Know the Key Terms in Business Rules Engine

Get a Trial Org to Explore Business Rules Engine

**Dive In: Learn About Recommended Core Features**

Look Up Decision Matrices to Get Output for a Set of Inputs

Store and Look Up Business Rules in Decision Tables

Create and Maintain Complex Rules With Expression Sets

Show Decision Explanations to Users

**Go Deeper: Learn About Features for Specific Business Needs**

Use Business Rules Engine Components in Flow

Use Business Rules Engine Components in Flow

Use Business Rules Engine Components in Omnistudio

Migrate Calculation Matrices and Procedures to Business Rules Engine

Invoke Expression Sets that Use Decision Explanations from Flow

**Extend Further: Learn About Additional Capabilities for Developers**

Invoke Expression Set and Decision Matrix With Business and Metadata APIs

Invoke Decision Tables With Business and Metadata APIs

Explain Business Rule Results

**Get Ready for Your Implementation**

Migrate Business Rules

Set Up Business Rules Engine Permissions

Business Rules Engine Default Limits

**Know Your Resources & Get Help**

Trail: Explore Business Rules Engine

Trailhead Community: Business Rules Engine

Slack Channel: #bre-partners

| | | |
|---|---|---|
| With Decision Explainer Business and Metadata APIs | | |

# Learn and Explore

Learn how Business Rules Engine can help your business and customers. Understand its core functionality, create a trial org, and learn how to get started.

**Business Rules Engine Key Terms**
Let's look at some key terms that you come across when you use Business Rules Engine.

**Business Rules Engine in Workflows (Example)**
Use Business Rules Engine components and features in your workflows to simplify complex lookups and to automate business decisions. Here's an example of how Business Rules Engine components work with a workflow.

**Lookup Tables: When to Use Which**
Business Rules Engine has two types of lookup tables–decision matrices and decision tables. To help you decide which type to use for your business requirements, let's look at the similarities and differences in them.

**Get a Trial Org to Explore Business Rules Engine**
Get hands-on experience with Business Rules Engine in a full-featured trial experience.

## Business Rules Engine Key Terms

Let's look at some key terms that you come across when you use Business Rules Engine.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Key Terms

| Term | Description |
|---|---|
| Expression Sets | Set of rules that are created in Expression Set Builder in Business Rules Engine. Expression sets are a series of rules created by using expression set elements. Each element added to an expression set forms a logical step that's runs sequentially. Expression sets evaluate conditions, |

| Term | Description |
| --- | --- |
| | perform mathematical operations, look up decision tables and matrices, and perform multiple transformations simultaneously. Add expression sets to workflows that are created in Flow, Omnistudio, or third-party tools. |
| Expression Set Elements | The building blocks of expression sets. You can add each available element as a step in an expression set. Use step elements to perform calculations and aggregations, build conditions, and look up decision tables and decision matrices to implement complex rules in a systematic and readable way. |
| Decision Matrices | Lookup tables that match your input values to a row in the matrix. When the matrix finds an exact match, it returns the row's output value. |
| Decision Tables | Advanced lookup tables that can provide multiple matching outcomes for the same set of inputs. Decision tables can read rules from Salesforce objects and can support different operators for each field in the object. |
| Advanced decision tables | A type of decision table that supports a large volume of rows and lesser complexity. |
| Standard decision tables | A type of decision table that supports complex conditions. |
| Expression Set Templates | Prebuilt, read-only rules that you can't modify. Save expression set templates as expression sets to modify them for your business needs. Templates provide an outline for building your expression set and save you the trouble of creating an expression set from scratch. |
| Workflows | Digital processes that can call Business Rules Engine components: expression sets, decision tables, and decision matrices. Create workflows with Flow, Omnistudio, or custom or third-party tools. |

# Business Rules Engine in Workflows (Example)

Use Business Rules Engine components and features in your workflows to simplify complex lookups and to automate business decisions. Here's an example of how Business Rules Engine components work with a workflow.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Example** Let's look at a workflow that provides a user with a quote for health insurance. Here's how the workflow is set up to gather inputs from a user and to return the insurance premium quote to the user:

1. A form in the workflow gathers the user's age, ZIP code, and smoking status.
2. After the user enters the requested details and clicks **Get Quote** on the input form, the workflow passes the user details to the Calculation Service expression set.
3. The expression set calls the Rate Factor decision matrix. The decision matrix looks for and matches the user's age, ZIP code, and smoking status to a row in the matrix, and returns the row's rate factor to the expression set.
4. The next steps in the expression set use the rate factor that the matrix returns and perform further calculations configured in the expression set to derive the quote.
5. The expression set returns the final quote to the workflow, which is then shown to the user.

# Lookup Tables: When to Use Which

Business Rules Engine has two types of lookup tables–decision matrices and decision tables. To help you decide which type to use for your business requirements, let’s look at the similarities and differences in them.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Decision matrices are exact-match based lookup tables that match your inputs to values in a row, and return the row’s output. Decision tables are complex lookup tables that match your inputs to values in multiple rows, if applicable, and return multiple matching outputs.

Lookup Table Features

| Functionality | Decision Matrix | Decision Table |
|---|---|---|
| Use sObjects and custom objects to create a lookup table | Not supported | Supported |
| Read rules from sObjects | Not supported | Supported |
| Use in flows | Supported | Supported |
| Have multiple rows with same input values, but different output values | Not supported | Supported |
| Update table data using a CSV file | Supported | Supported |
| Generate CSV file from table | Supported | Supported only for CSV-based Decision Tables |
| Edit table data directly | Supported | Supported only for CSV-based Decision Tables |
| Create, manage, and activate versions | Supported | Not supported |
| Look up a column using a single operator (equals to) | Supported | Supported |
| Look up a column using multiple operators (less than, more than, contains) | Not supported | Supported |
| Use row-level conditions. Example: If a table has three inputs columns, include a row-level condition that returns an output if row 1 and either row 2 or row 3 have matching values. | Not supported | Supported |

Let's look at some use cases where you'd have to look up a table for values. Let's also look at which lookup table type you can use for the specific use case.

| Use Case | Decision Matrix | Decision Table |
|---|---|---|
| Look up a table for an exact match of values or a range of values | Supported | Supported |
| Look up a table that has versions | Supported | Not supported |

| Use Case | Decision Matrix | Decision Table |
|---|---|---|
| grouped by a key. For example, a table with insurance premium rates, where each version has rates for a different state. | | |
| Look for an exact match of your input values in a table that has over 300 columns and over a million rows. | Supported | Supported |
| Call a lookup table in an Omniscript. | Supported | Not supported |
| Look up exact values or ranges in table with less than 10 input columns and less than five output columns | Supported | Supported |
| Look up a table using different operators for each input column. For example: less than, greater than, equals, not equals, or matches. | Not supported | Supported |
| Use the data in sObjects or custom objects to perform a lookup and get the output. | Not supported | Supported |
| Look up a column that supports picklist values. | Not supported | Supported |
| Call a lookup table in a flow. | Supported | Supported |

See Also

*Salesforce Help*: Create Standard Decision Matrices
*Salesforce Help*: Create a Decision Table

## Get a Trial Org to Explore Business Rules Engine

Get hands-on experience with Business Rules Engine in a full-featured trial experience.

REQUIRED EDITIONS

Available in: Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled

**Note** Trial orgs are intended for proof of concept and guided self-exploration. Trial orgs expire in 30 days.

1. Go to Developer Free Trials sign-up page and look for your cloud in the Industry-Based Trials tab.
2. Sign up for a trial org.
   If your cloud has more than one trial org options, click **Compare 2 Trial Options** and select an org type.

| | |
|---|---|
| **Learning Trial Org** | This fully configured org includes rich sample data to see a cloud's features in action. |
| **Base Trial Org** | This unconfigured org is a blank slate with only the necessary licenses and permissions. Use these orgs for quick proof of concept projects or general testing. |

3. Enter your contact details.
4. Read and agree to the service agreement.
5. Click **Start Your Free Trial**.

After your org is created, look for an email with the login details.

# Plan and Prepare for Your Business Rules Engine Implementation

Explore some key concepts and considerations before you implement Business Rules Engine in your org.

**How to Choose Between Standard and Advanced Decision Table Types**
Optimize your rule management based on the size and complexity of data by selecting the right decision table type. This categorization improves system performance and storage efficiency, ensuring fast and scalable decision-making processes. Customize your tables to improve flexibility and resource allocation.

**How to Bring External Data into Business Rules Engine**
Review the methods and processes to bring data from your external systems into Business Rules Engine, and choose the method that best suits your business requirements.

**How Business Rules Engine and Salesforce Flow Enhance Digital Process Automation**
Discover how Business Rules Engine and Salesforce Flow work together to enhance decision-making and reduce the complexity of your automation workflow. Salesforce Flow is a powerful tool for automating processes and defining business logic. Business Rules Engine offers distinct advantages for handling complex, dynamic rule-based decision-making at scale. Business Rules Engine complements Salesforce Flow for managing advanced rule-based logic.

**Best Practices for Executing Expression Set Rules on Large Datasets by using Salesforce Flow**
Business scenarios can involve running rules on thousands or even millions of input records to get the results you need. Business Rules Engine gives you multiple options for bulk processing rules on large datasets, according to your specific requirements. You can run business rules on large datasets in a single call or in batches.

# How to Choose Between Standard and Advanced Decision Table Types

Optimize your rule management based on the size and complexity of data by selecting the right decision table type. This categorization improves system performance and storage efficiency, ensuring fast and scalable decision-making processes. Customize your tables to improve flexibility and resource allocation.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, edit, and activate a decision table: | Rule Engine Designer |
| To run decision tables: | Rule Engine Designer<br>OR<br>Rule Engine Runtime |

## Complex Rule in Decision Tables

A rule is complex if it uses only OR conditions. This is because the decision table looks through all the fields in a database to find the match, increasing latency and affecting performance. AND conditions, in contrast, specify which fields to check, making the lookup faster.

## Differences Between Standard and Advanced Tables

These are the key differences between standard and advanced decision table types.

| Support | Standard | Advanced |
|---|---|---|
| Number of Rows | A maximum of 100,000 rows can be processed. It can be | A maximum of 2 million rows can be processed. It can be |

| Support | Standard | Advanced |
|---|---|---|
| | extended if required. | extended if required. |
| Number of Active Decision Tables | 20 decision tables can be simultaneously active in an org. It can be extended if required. | 40 decision tables can be simultaneously active in an org. It can be extended if required. |
| Rule Complexity | Complex rules that use only OR conditions.<br><br>Simple rules that use only AND conditions or a combination of AND and OR conditions. | Simple rules that use only AND conditions or a combination of AND and OR conditions. The rule must meet these criteria:<br><br>• At least one AND condition,<br>• At least one Required input condition column,<br>• At least one input condition column that uses the Equals operator.<br><br>Example: (1 OR 2) AND (3 OR 4) this is not supported in Advanced decision tables as there is no Required input condition.<br><br>Example: 1 AND (2 OR (3 OR 4)) is supported in Advanced decision tables where at least one input column is Required and uses the Equals operator. |
| Number of AND and OR Conditions | The maximum number of logical conditions, including both AND and OR, is 30. | The maximum number of OR conditions is 10 and the maximum number of AND conditions is 31. |
| CSV File | You can't upload data as CSV files. | You can upload data as CSV files. |
| Null Values | You get the option to include Null Values when you create a decision table. | Null Values are considered by default in the decision table. |
| Standard Objects | Not Available | Standard objects like Account, Lead, Contact, Opportunity, and Case are available. |

## When to Use Standard and Advanced Decision Tables

Here are some scenarios to help you choose between standard and advanced decision tables.

| Scenario | Description | Decision Table Type |
|---|---|---|
| Less volume and high complexity | If you have 100,000 or less rows with complex rules. | Standard |
| Less volume and less complexity | If you have 100,000 or less rows with simple rules. | Standard, Advanced |
| Large Volume and less complexity | If you have more than 100,000 rows with simple rules. | Advanced |
| Large Volume and high complexity | If you have more than 100,000 rows and complex rules. | • Advanced - Simplify your rules that use at least one 'AND' condition and then use advanced decision tables.<br>• Standard - If you can't simplify the rules, contact Salesforce admin to extend the row limits for standard decision tables. |

These additional features are available only in advanced decision table type. If you want to use any of these features, you must use the advanced decision table type.

| Features | Description |
|---|---|
| CSV Data Upload | If you want to upload data through a CSV file. |
| Standard Objects | If you want to use standard objects like Account, Lead, Contact, Opportunity, and Case |

When you need complex rules and want to access features that are available only in advanced decision tables, simplify your rules that use at least one AND condition. Then, use the advanced decision tables.

**Example** Standard Decision Table A bank wants to offer a special loan to customers who meet at least one of these criteria:

- Condition 1: The customer has a credit score greater than 700.
- Condition 2: The customer has been an active user of the bank for more than 5 years till 2024.
- Condition 3: The customer has a monthly income greater than $5,000.
- Condition 4: The customer holds investments worth more than $100,000 in the bank.

The decision to offer the loan is based on OR conditions across these criteria. The decision table

condition logic for this scenario is 1 OR 2 OR 3 OR 4.To determine loan eligibility, review the sample customer details. Each criteria is captured as an input condition field and the loan eligibility output is captured as a result field in the decision table. Customers are eligible if any one of the conditions is met.

| **Input** | | | | | **Output** |
|---|---|---|---|---|---|
| **Customer ID** | **Credit Score (Condition 1)** | **Joining Date (Condition 2)** | **Monthly Income (Condition 3)** | **Investments (Condition 4)** | **Loan Eligibility** |
| C001 | 720 | 11.11.2023 | $4,800 | $90,000 | Yes |
| C002 | 680 | 03.03.2015 | $5,200 | $80,000 | Yes |
| C003 | 680 | 07.10.2021 | $4,800 | $90,000 | No |
| C004 | 690 | 01.05.2022 | $4,500 | $150,000 | Yes |

As the rule in this scenario uses only OR conditions, it's considered complex, and a standard decision table type must be used.

**Example** Advanced Decision TableA utility company wants to reward residential customers who cut energy use. They are offering financial incentives to customers who use less energy, use energy-efficient technology, or join energy-saving programs. A customer's eligibility for the energy-saving incentive is determined by these criteria:

- Condition 1: Energy Consumption Reduction: The customer reduces their energy consumption by at least 15% compared to the previous year.
- Condition 2: Energy-Efficient Technology: The customer has installed an energy-efficient heat pump.
- Condition 3: Participation in Utility Programs: The customer participates in at least one utility energy-saving program, such as a demand response program or time-of-use pricing.
- Condition 4: Annual Energy Usage: The customer uses less than 20,000 kWh annually.

The decision to award the incentive is based on a combination of AND and OR conditions. The decision table condition rule logic for this scenario is 1 AND (2 OR 3) AND 4. To get the energy-saving incentive, a customer must satisfy these conditions:

- 1 and 4
- Either 2 or 3

Review some of the sample customer details based on which the customer's eligibility is determined. Each criteria is captured as an input condition field and the incentive eligibility output is captured as a result field in the decision table.

| Input | | | | | Output |
|---|---|---|---|---|---|
| **Customer ID** | **Energy Reduction <15% (Condition 1)** | **Installed Heat Pump (Condition 2)** | **Participated in Utility Program (Condition 3)** | **Annual Energy Usage < 20,000 kWh (Condition 4)** | **Incentive Eligibility** |
| C001 | Yes | Yes | Yes | Yes | Yes |
| C002 | Yes | No | Yes | Yes | Yes |
| C003 | Yes | Yes | No | Yes | Yes |
| C004 | No | Yes | Yes | Yes | No |
| C005 | Yes | No | Yes | No | No |

As the rule in this scenario uses a combination of AND and OR conditions, it's considered simple, and an advanced decision table type must be used.

**Example** Advanced Decision TableA health insurance company wants to offer a wellness incentive to customers who meet specific health criteria. The eligibility for the wellness incentive is determined by these criteria:

- Condition 1: The customer has completed an annual health check-up.
- Condition 2: The customer has a BMI (Body Mass Index) within the healthy range (18.5 - 24.9).
- Condition 3: The customer participates in a fitness program.
- Condition 4: The customer has a non-smoker status.

The customer must have completed an annual health check-up, and must either have a BMI within the healthy range, or participate in a fitness program, or be a non-smoker. The decision to award the wellness incentive is based on a combination of AND and OR conditions. The decision table condition rule logic for this scenario is 1 AND (2 OR (3 OR 4)). Review some of the sample customer details based on which the customer's eligibility is determined. Each criterion is captured as an input condition field and the incentive eligibility output is captured as a result field in the decision table.

| Input | | | | | Output |
|---|---|---|---|---|---|
| **Customer ID** | **Annual Health Check-up (Condition 1)** | **BMI within Healthy Range (Condition 2)** | **Participates in Fitness Program (Condition 3)** | **Non-Smoker Status (Condition 4)** | **Wellness Incentive Eligibility** |
| C001 | Yes | Yes | No | Yes | Yes |
| C002 | Yes | No | Yes | No | Yes |
| C003 | Yes | No | No | Yes | Yes |

| Input | | | | | Output |
|---|---|---|---|---|---|
| **Customer ID** | **Annual Health Check-up (Condition 1)** | **BMI within Healthy Range (Condition 2)** | **Participates in Fitness Program (Condition 3)** | **Non-Smoker Status (Condition 4)** | **Wellness Incentive Eligibility** |
| C004 | Yes | Yes | Yes | No | Yes |
| C005 | No | Yes | Yes | Yes | No |

As the rule in this scenario uses a combination of AND and OR conditions, and at least one input column is required and uses the Equals operator, it's considered simple, and an advanced decision table type must be used.

# How to Bring External Data into Business Rules Engine

Review the methods and processes to bring data from your external systems into Business Rules Engine, and choose the method that best suits your business requirements.

## Role of Business Rules Engine

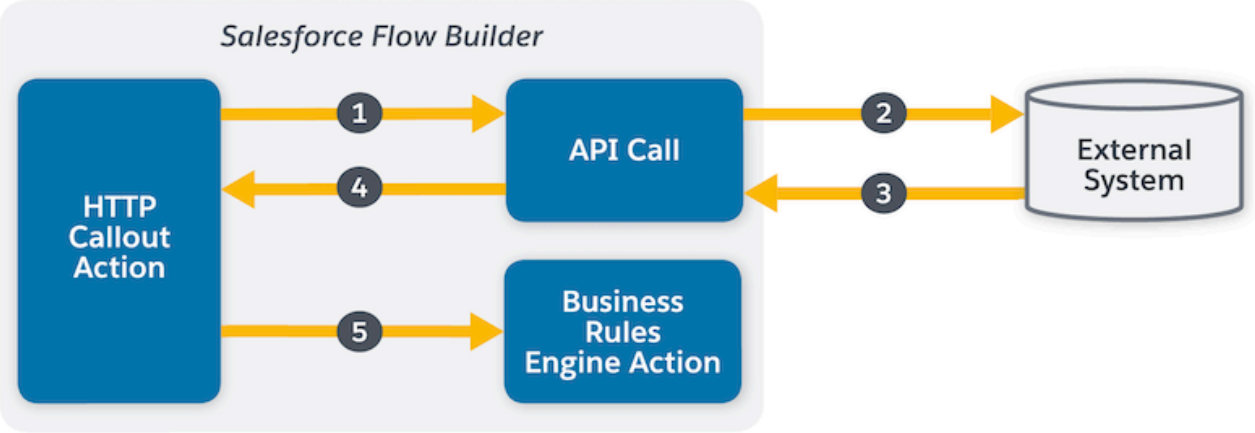
Business Rules Engine doesn't inherently provide processes or methods to fetch external data. Instead, it offers components such as lookup tables and expression sets that you can use to perform complex lookups and calculations, and create business rules. Before you invoke Business Rules Engine components, use other Salesforce products and services to extract data from external sources.

**Note** All customer data coming from external sources into Salesforce adheres to the Salesforce standard data compliance and security protocols. See Salesforce Security Guide.

Based on the architecture that you've designed for your digital processes, choose one of these methods to bring external data into Salesforce and run rules by using Business Rules Engine.

## Method 1: Use Salesforce Flow

If you want to use Salesforce Flow to automate your business processes and need external data to run rules by using Business Rule Engine, create an HTTP callout action that calls a web-based service API. You can do this in Flow Builder without code. After you add the API details, Flow Builder generates a reusable callout action that you can use right away in Flow Builder. You can also edit an HTTP callout action declaratively and configure the inferred data structure. The API call fetches the data from your external system and makes it available in flow. Then, use an action element in flow to invoke Business Rules Engine components such as expression sets and lookup tables, and pass the data to these components. See HTTP Callout.



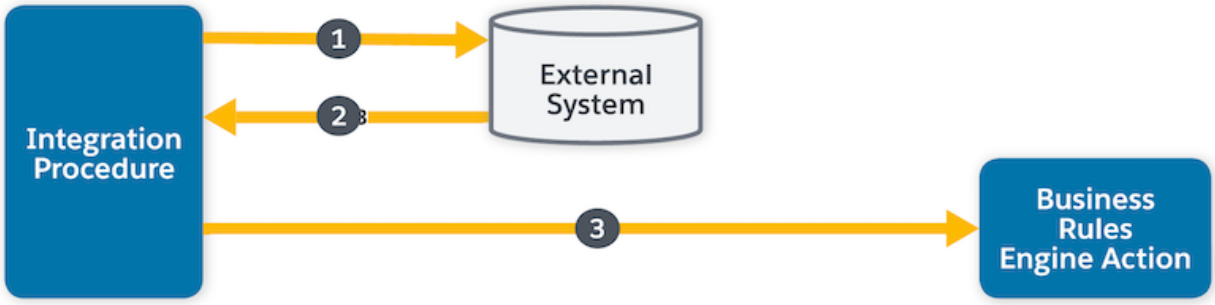

## Method 2: Use Salesforce Custom Object

Create a Salesforce custom object and fields, and upload your external data as CSV files. You can also use any of the other data import methods available to sync data between your external systems and Salesforce custom object. Then, use the data from the custom object to run rules in Business Rules Engine. See Salesforce Data Import.

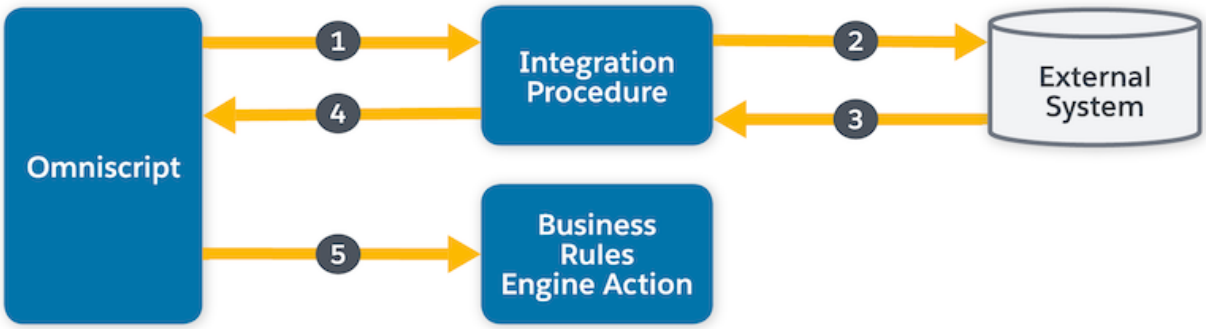
## Method 3: Use Omnistudio Integration Procedures

Omnistudio Integration Procedures can read and write data from external systems via REST calls. Both Omniscripts and Integration Procedures can invoke Business Rules Engine components and pass external data to them, but the process differs slightly for each.

- Integration Procedures: Fetch external data via REST calls, then invoke Business Rules Engine components and pass the data to these components.



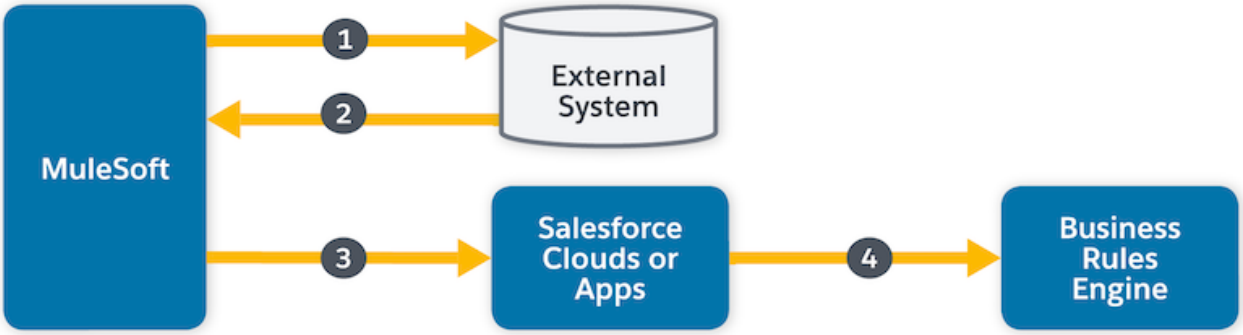

- Omniscripts: Call an Integration Procedure to fetch external data, then invoke the Business Rules Engine components and pass the data to these components.




There are prebuilt integrations with Business Rules Engine components that you can call from an Omniscript or Integration Procedure with invocable actions. See Omnistudio Integration Procedures.

## Method 4: Use MuleSoft Integration

You can use MuleSoft Integration to bring data from outside Salesforce into the Business Rules Engine. Create a flow in your MuleSoft application that connects to the external system and retrieves the necessary data. Then, use any of the Salesforce Cloud processes or apps to call the external data from MuleSoft and pass it to Business Rules Engine. With MuleSoft, you can automate data flow from external systems into Business Rules Engine and create business rules to process the data according to your business logic. See MuleSoft Documentation.

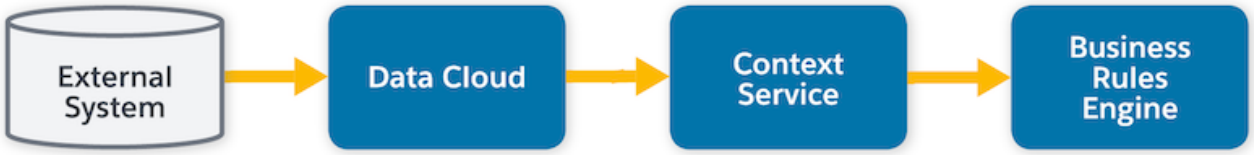



## Method 5: Use Apex

If you have a custom integration scenario that requires precise control over how you fetch, transform, and insert data into Salesforce, use Apex to create custom integrations with external systems. You can use Apex to call external APIs, retrieve data, and insert or update records in Salesforce. In Apex, you can use the Invocable namespace feature to invoke expression set and lookup table actions. Invocable Namespace contains the required classes and methods to invoke actions from Apex. To know more about invoking Business Rules Engine components in Apex, see Invocable Actions.

### Method 6: Use Data Cloud and Context Service

Data Cloud is an integrated platform to bring together data from disparate sources. Use connectors to ingest external data into Data Cloud. Then, use Context Service to map the xml data from Data Cloud to Business Rules Engine at run time and run business rules. To explore how external data is ingested into Data Cloud, see Connect an External Data Source to Salesforce Data Cloud.




## How Business Rules Engine and Salesforce Flow Enhance Digital Process Automation

Discover how Business Rules Engine and Salesforce Flow work together to enhance decision-making and reduce the complexity of your automation workflow. Salesforce Flow is a powerful tool for automating processes and defining business logic. Business Rules Engine offers distinct advantages for handling complex, dynamic rule-based decision-making at scale. Business Rules Engine complements Salesforce Flow for managing advanced rule-based logic.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Business Rules Engine offers specialized and scalable solution for managing complex business rules. It helps increase performance, flexibility, and governance of your workflow process, which is difficult to achieve through Salesforce Flow alone. Learn more about how Business Rules Engine features work with Salesforce Flow to increase efficiency of your digital workflows.

### Simplistic and Declarative Approach

Business Rules Engine is specifically designed to simplify the complex decisioning process and create business rules declaratively. Business Rules Engine is a prebuilt extension to Salesforce Flow for creating and maintaining business rules. Move all your decisioning steps to Business Rules Engine and call them in Salesforce Flow by using an action element. This integration reduces the steps in Salesforce Flow, thereby simplifying your workflow. Business users can use Business Rules Engine to easily define complex rules by using the no-code interface.

Business Rules Engine provides a simple and intuitive visual builder tool called expression sets to set up

rules. It also includes compact and user-friendly lookup tables, for example, a decision matrix and decision table that you can use to create conditions and actions in a tabular format. These lookup tables are easy to maintain and update. A decision table can contain thousands of rules in a single object. In Salesforce Flow, to achieve the same result, you need to create several decision nodes. You can also simulate your rules in the visual builder tool to test your decisions and verify the results.

## Reusability Across Multiple Processes

Create reusable rules in Business Rules Engine that you can apply to different objects and applications. Instead of creating the same rule multiple times for all the different processes, define it once in Business Rules Engine and then use it wherever you need it.

Business Rules Engine enables you to create and maintain rules in a dedicated workspace without interfering with the Salesforce Flow logic.

Example: An energy company uses a Business Rules Engine rule to manage clean energy discount calculations that apply to multiple processes. When they modify a rule, the changes are automatically updated in all the processes where the rule is used.

## Design Agility

Business Rules Engine can easily adapt to changes. To give you more flexibility, it separates the change management process of business rules from that of your workflow. With Business Rules Engine, you can quickly create, modify, and deploy rules to automate decision-making without the need for coding or lengthy development cycles. Your workflow design becomes more challenging and time-consuming when the complexity of the rules grows if you’re creating it entirely in Salesforce Flow.

With Business Rules Engine, you can quickly test and deploy any changes to rules. You can roll out new rules or updates based on shifting market conditions, competition, or customer feedback in a matter of hours or even minutes. This flexibility means businesses can reduce longer implementation timelines, and stay agile and competitive in dynamic environments.

## Centralized Management of Business Rules

Business Rules Engine centralizes rule management so that you can create, update, or delete rules from a single location instead of doing it individually for each process. You can also govern the rules centrally with specific user access control, which enhances security and consistency. This centralized rule management helps you to use the same rules across different departments or processes. To call the rules in a flow, use an invocable action.

In Salesforce Flow, business rules are often embedded within individual flow frameworks, which makes it time-consuming to keep workflows consistent when you use the same rule across multiple flows. The rule data and framework are separate in Business Rules Engine. With Business Rules Engine, you can refer to and maintain rules more easily from a centralized repository.

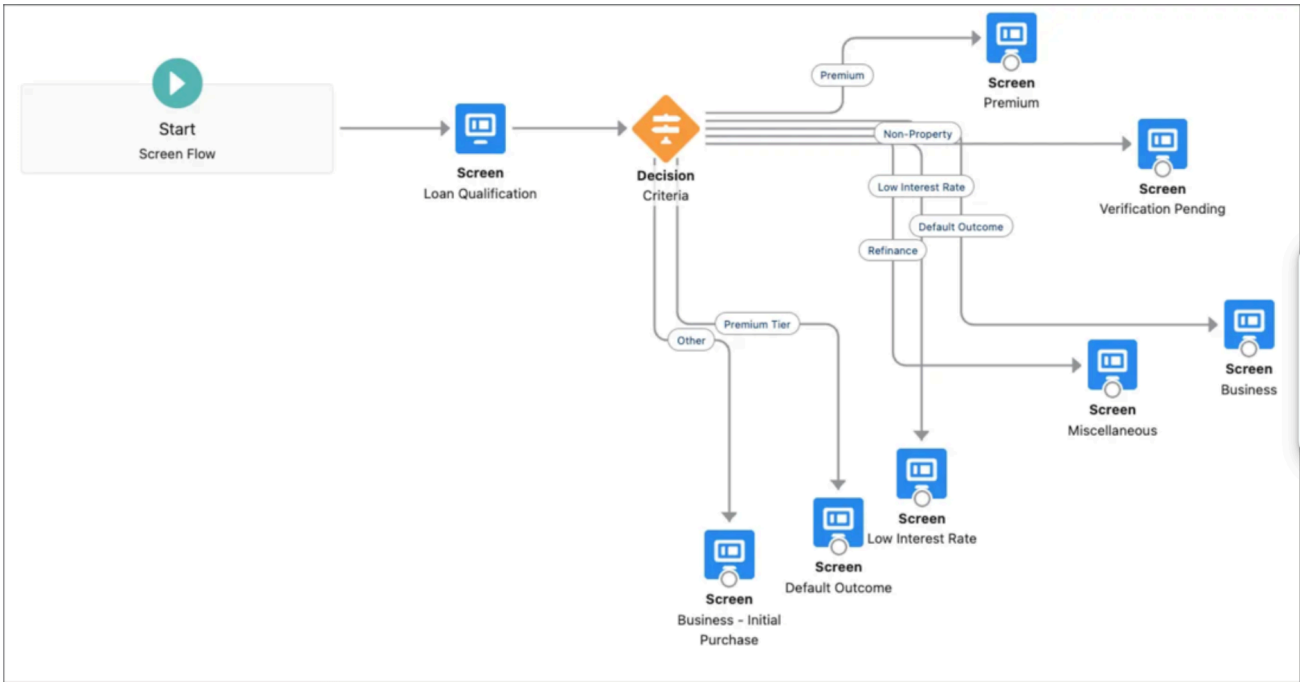
## User Roles and Permissions Management

Business Rules Engine gives you control over user permissions and profiles for designing and running business rules. It provides access management that lets you specify who can view, create, or update expression sets and decision tables. You can also govern access to Business Rules Engine objects through permission sets and criteria-based sharing. This access control ensures that only selected users have view or edit access to a particular component.

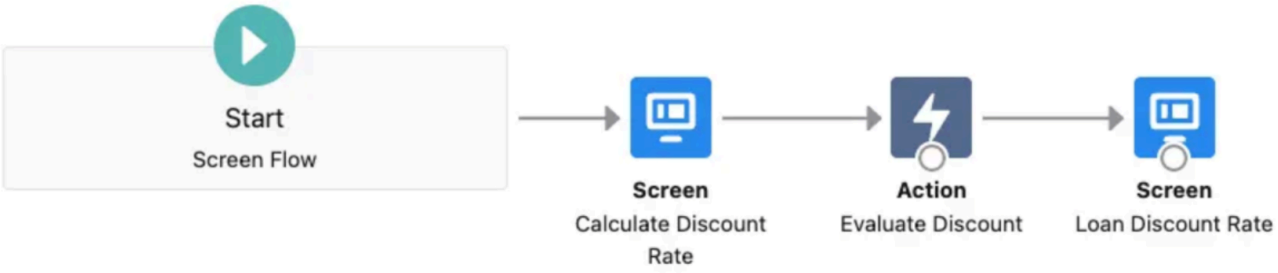
## Simplify Decision-Making in Salesforce Flow Example

Review an example to see how Business Rules Engine handles the decision-making process and helps simplify Salesforce Flow.

First, let's look at the scenario where you build complex decisions in Salesforce Flow without Business Rules Engine. When you create business rules in Salesforce Flow, it often involves adding multiple elements and branches that make the flow quite detailed. For larger workflows, building rules in Salesforce Flow can become intricate, making it challenging to add new processes or make edits to existing ones. While Flow provides powerful capabilities, managing complex rule configurations in Business Rules Engine ensures ease of design and maintenance.




Next, let's look at the scenario where you build complex decisions in Salesforce Flow with Business Rules Engine. To simplify your flow and make it efficient, create your rules in Business Rules Engine. Use the lookup tables and expression set components in Business Rules Engine to create your rules declaratively by using an intuitive UI. Then invoke the rules in Salesforce Flow by using an action element. When you transfer your decisioning steps to Business Rules Engine, you reduce the overall number of workflow steps in the flow. This makes your flow simpler and lighter. You can then use Salesforce Flow efficiently for larger workflow processes.




In conclusion, use Salesforce Flow to create your primary workflow processes and Business Rules Engine to configure your business rules to efficiently manage your overall workflow automation.

## Best Practices for Executing Expression Set Rules on Large Datasets by using Salesforce Flow

Business scenarios can involve running rules on thousands or even millions of input records to get the results you need. Business Rules Engine gives you multiple options for bulk processing rules on large datasets, according to your specific requirements. You can run business rules on large datasets in a single call or in batches.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for Industries clouds that have Business Rules Engine and Context Service enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete expression sets | Rules Engine Designer |
| To run expression sets in Business Rules Engine | Rules Engine Runtime |
| To create a flow | Manage Flow |
| To load, view, edit, and delete context definitions | Context Service Admin |

Review the options to bulk process rules on large datasets with Salesforce Flow and choose the option that best suits your business needs.

**Option 1: Use Standard Expression Sets and Batch Management**
To run rules on thousands or millions of input records in batches, use standard expression sets and batch management to process the rules. Create your rules in expression sets and batch job definitions in Batch Management. Use the batch job definitions to process your rules in batches. Finally, invoke the expression set and batch job definition in Salesforce Flow to execute your rules on input records.

**Option 2: Use Context Service and Expression Sets**
If you want to run rules on thousands of records in a single call and need faster data access for your

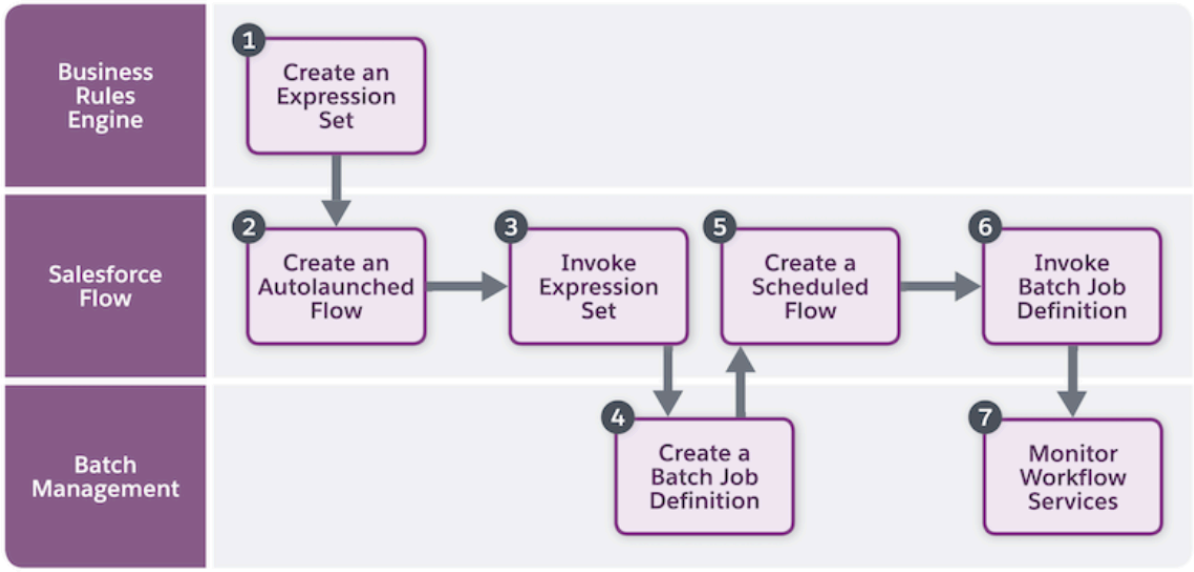
business processes, use Context Service to serve your data request. Use expression set to create your rule and fetch the data from objects through Context Service. Finally, invoke the expression set with context data in Salesforce Flow to execute your rules on the input data. You can get the results in real time by using this option.

## Option 1: Use Standard Expression Sets and Batch Management

To run rules on thousands or millions of input records in batches, use standard expression sets and batch management to process the rules. Create your rules in expression sets and batch job definitions in Batch Management. Use the batch job definitions to process your rules in batches. Finally, invoke the expression set and batch job definition in Salesforce Flow to execute your rules on input records.

Make sure you limit the batch size according to the Business Rules Engine guardrail limits. See Business Rules Engine Default Limits.

Here's the overview of the process on how to run rules on large datasets by using expression sets and Batch Management in Salesforce Flow.




Here's how you configure the process.

1. Create an Expression Set: In Business Rules Engine, create an expression set to define your business rule, configure the expression set based on your requirement, and then activate it.
2. Create an Autolaunched Flow: Create a flow to orchestrate your process and make sure you use the Autolaunched Flow type so that the flow is available in Batch Management.
3. Invoke Expression Set: In Flow Builder, add an Action element. Search for the Expression Set action and add it to your flow. The Flow uses this action to invoke the expression set. Configure the inputs for the Expression Set action and make sure that all the parameters are correctly mapped in the action configuration. Activate the flow.
4. Create a Batch Job Definition: In Batch Management, create a batch job definition by defining the process type as Flow and the execution process as Autolaunched Flow. Activate the definition.

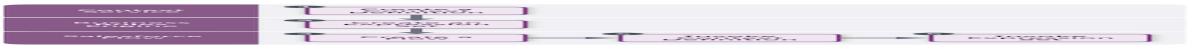
5. Create a Scheduled Flow: Create a Schedule-Triggered Flow to execute the batch job at a certain date and time based on your requirement.
6. Invoke Batch Job Definition: Add an Action element in the Schedule-Triggered Flow and search for Batch Job in the Action window. Select your batch job definition and configure the details. Activate and run the Flow.
7. Monitor Workflow Services: You can monitor the status of each batch job run in the Monitor Workflow Services section.

**Example** An online retail company uses Salesforce Loyalty Cloud to determine the loyalty points each customer earns based on their purchases and tier levels. The company uses complex loyalty rules defined in Business Rules Engine to calculate the points. The company processes 2 million daily transactions in bulk by using Batch Management. Salesforce Flow automates the points determination process by running a scheduled batch at regular intervals. Business Rules Engine and Batch Management together make sure accurate and efficient points allocation across large volumes of transactions.

## Option 2: Use Context Service and Expression Sets

If you want to run rules on thousands of records in a single call and need faster data access for your business processes, use Context Service to serve your data request. Use expression set to create your rule and fetch the data from objects through Context Service. Finally, invoke the expression set with context data in Salesforce Flow to execute your rules on the input data. You can get the results in real time by using this option.

Here’s the overview of the process on how to run rules on large datasets by using expression sets and Context Service in Salesforce Flow.



Here's how you configure the process.

1. Create a Context Definition: Create a context definition in Context Service. Add nodes and attributes with the relevant data types. Add context tags to each node and attribute. Map the nodes with objects and attributes with fields. Activate the context definition.
2. Create an Expression Set: Create an expression set and select the context definition. Use list group and list filter elements to call the data from your object. List filters use the context mapping defined in the context definition to fetch the requested object data. Use the expression set to configure your business rule and make sure to activate it.
3. Create a Flow: Create a Screen Flow to configure and execute your process. You can also use a Schedule-Triggered Flow to execute your process at a certain date and time as per your requirement.
4. Invoke Context Definition: Add an Action element in Flow Builder. Under Search Action, search for the Build Context action and add it to flow. This action creates a context instance and fetches the context definition details during runtime. In the Setup Input Values section, add the context definition ID in the Context Definition field. You can also add Context Data, Context Mapping, and Tagged Data details as per your requirement.

5. Invoke Expression Set: In Flow Builder, add an Action element. Search for the Expression Set action and add it to your flow. The Flow uses this action to invoke the Expression Set. Configure the inputs for the Expression Set action and make sure that all the parameters are correctly mapped in the action configuration. Activate and run the flow.
   Context Service also has a guardrail limit for the maximum number of records on which you can run rules in a single call. See Context Service Limits.

**Example** A government entity wants to verify if a citizen is eligible for welfare benefits based on multiple factors such as income, family size, age, disability status, and employment status. Context Service can pull the data for each employee in real time from citizen records and Business Rules Engine can apply rules to determine eligibility.

# Business Rules Engine Setup

Configure your users to use Business Rules Engine and assign permission sets so that the users can access expression sets and lookup tables.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Note** Business Rules Engine components are included with all objects in the Full, Partial, Developer, Developer, and Pro sandbox copies.

### Permission Sets for Business Rules Engine

Review the level of access and permissions provided by the permission sets that are associated with Business Rules Engine permission set licenses. The number of permission set licenses in your Salesforce org depends on your license configuration and on how your Industries solution is packaged. Permission set licenses entitle users to access features that aren't included in their user licenses.

### Create a Custom Permission Set

A Salesforce org that has Business Rules Engine comes with a Rule Engine Designer permission set. If you want to create custom permission sets, clone and then modify the Rule Engine Designer permission set. For example, you can create a read-only permission set for users to only view and run but not create, edit, or delete expression sets.

### Assign a Permission Set to Users

Assign a permission set to users to provide the users the required access to Business Rules Engine objects.

### Permissions for Community and Guest Users

Users with the community and guest user profiles can run expression sets and decision matrices when the profiles are assigned the permissions to run expression sets and decision matrices.

# Permission Sets for Business Rules Engine

Review the level of access and permissions provided by the permission sets that are associated with Business Rules Engine permission set licenses. The number of permission set licenses in your Salesforce org depends on your license configuration and on how your Industries solution is packaged. Permission set licenses entitle users to access features that aren't included in their user licenses.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| **Included Permissions** | **Business Rules Engine Permission Sets** | |
|---|---|---|
| | **Rule Engine Runtime** | **Rule Engine Designer** |
| Run expression sets and lookup tables | Yes | Yes |
| Read access to Business Rules Engine objects | Yes | Yes |
| Create, Update, Delete access to Business Rules Engine objects | No | Yes |
| Use context definition tags as list variables in expression sets | No | Yes |
| Read decision tables | Yes | Yes |
| Create, Update, Delete decision tables | No | Yes |

The permission sets give create, read, edit, and delete access to these Business Rules Engine objects.

- Expression Set
- Expression Set Version
- Decision Matrix
- Decision Matrix Version
- Decision Matrix Row
- Decision Matrix Column
- Decision Matrix Column Range

## Additional Permission Sets

Use these additional permission sets to extend Business Rules Engine features.

| Permission Set | Description |
| --- | --- |
| Context Service Admin | Provides access to Context Service objects and features.<br><br>When your Salesforce org has Business Rules Engine and Context Service enabled together, you can create and activate context definitions. Rule designers can then use the context definition's tags as list variables in expression set versions. |
| Context Service Runtime | Enables users to run expression sets that use context definition tags as list variables. |
| Decision Explainer Service Access | Provides access to Decision Explainer features and objects<br><br>When your Salesforce org has Business Rules Engine and Decision Explainer enabled together, you can use Decision Explainer to configure and show explanations for an expression set's version steps to your user. You can also capture action logs for every run of an expression set. |

For a list of all Business Rules Engine permission set licenses, see Business Rules Engine Permission Set Licenses.

## Create a Custom Permission Set

A Salesforce org that has Business Rules Engine comes with a Rule Engine Designer permission set. If you want to create custom permission sets, clone and then modify the Rule Engine Designer permission set. For example, you can create a read-only permission set for users to only view and run but not create, edit, or delete expression sets.

REQUIRED EDITIONS

| |
| --- |
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
| --- | --- |
| To clone Rules Engine Designer permission set: | Manage Profiles and Permission Sets and Rule Engine Designer |

| USER PERMISSIONS NEEDED | |
|---|---|
| To assign permission sets to users: | Assign Permission Sets |

1. From Setup, in the Quick Find box, enter `Permission Sets`, and then select **Permission Sets**.
2. In the list of permission sets, click **Clone** next to the Rules Engine Designer permission set.
3. Enter a label, such as `Engine Business User`, for the cloned permissions set.
   The API name autopopulates.
4. Save your work.
5. Select the new permission set.
6. Change the permissions for these objects so that the objects are read only:
   - Expression Set
   - Expression Set Version
   - Decision Matrix
   - Decision Matrix Version
   - Decision Matrix Row
   - Decision Matrix Column
   - Decision Matrix Column Range

   a. Under Apps, click **Object Settings**.
   b. Select an object whose permissions you want to modify.
   c. Click **Edit**.
   d. Deselect the permissions, except the read permission.

   e. Save your changes.
   The selected permissions are applied for the object.

After you create the permission set, assign the set to relevant users.

## Assign a Permission Set to Users

Assign a permission set to users to provide the users the required access to Business Rules Engine objects.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To assign a permission set to users: | Assign Permission Sets |

1. From Setup, in the Quick Find box, enter `Users`, and then select **Users**.
2. Under the Full Name column, select the name of the user.
3. Click **Permission Set Assignments** | **Edit Assignments**.
4. To update the permission sets assigned to the user, move the set from the Available Permission Sets list to the Enabled Permission Sets list.
5. Save your changes.

## Permissions for Community and Guest Users

Users with the community and guest user profiles can run expression sets and decision matrices when the profiles are assigned the permissions to run expression sets and decision matrices.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| User Permission | Description |
|---|---|
| Run Decision Matrices | Allows users with the community or guest profiles to run decision matrices that are created using Business Rules Engine. |
| Run Expression Sets | Allows users with the community or guest profiles to run expression sets that are created using Business Rules Engine. |

# Decision Matrices

A decision matrix matches input values to a table row and returns the row's output values. Expression sets and various digital procedures can call decision matrices. Decision matrices accept JSON input from, and return JSON output to the digital processes that call the matrices. Decision matrices are useful for implementing complex rules in a systematic, readable manner.

REQUIRED EDITIONS

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete decision matrices and matrix versions: | Rule Engine Designer |
| To use decision matrices in Business Rules Engine: | Rule Engine Runtime |

You can choose from these decision matrix types based on your business needs:

- Standard Decision Matrices
- Grouped Decision Matrices

Multiple versions of the same decision matrix can be active simultaneously. The start date, end date, and rank determine which version of the matrix to use. Ranks are ascending, so rank 2 is higher than rank 1. If the start date of one version of a decision matrix overlaps the end date of another, you can set rank values to determine which one to use. If a call to a decision matrix specifies an effective date, that date is used to find the correct matrix. If an effective date isn't specified, the default is today.

### Standard Decision Matrices

Standard decision matrices have characteristics, such as input columns, output columns, and versions, that are common to all decision matrices, irrespective of their type. Use standard decision matrices when you don't need the matrix to be part of a group or to be versioned row by row.

### Grouped Decision Matrices

With grouped matrices, you can group a matrix's data into different versions if several rows have the same column values. For example, in a matrix with age and gender-based transport fares, you can use gender as the matrix key, with male and female as the key values for the versions. This grouping prevents repeating male and female as row values in the matrix and groups the data for males in one version and females in another.

### Text Ranges in Decision Matrices

If your matrix has considerable data, use ranges to keep the matrix small. With text ranges, you can concisely specify a range of acceptable values in a matrix column instead of listing individual values.

### Numeric Ranges in Decision Matrices

Use numeric ranges to keep matrices with sizable data small. With number ranges, you can concisely specify a range of acceptable values in a matrix column instead of listing individual values.

### Guidelines for Creating and Editing Decision Matrices

Decision matrices require certain criteria to be met to successfully create versions or modify the data in existing versions.

### Create Standard Decision Matrices

You can create a standard decision matrix by uploading a CSV file, by creating a version of an existing matrix, or manually.

### Create Grouped Decision Matrices

You can create a decision matrix that groups rows in versions based on a group key and, if necessary, a sub group key. Versions in a grouped decision matrix can have different start date time values. When a grouped matrix is invoked, the group or subgroup keys are considered like input variables, for which you must provide values for lookup.

### Manage and Maintain Decision Matrices

Learn how to edit decision matrix rows and columns, sort and filter matrix rows, and export matrices to CSV or DMN files.

### Decision Matrix Migration Considerations

Here are a few typical migration scenarios and recommendations on how to maintain decision matrices in your target org for a successful migration.

**Decision Matrices for Developers**
Developers in your Salesforce org can employ decision matrix developer resources to use and extend decision matrices.

# Standard Decision Matrices

Standard decision matrices have characteristics, such as input columns, output columns, and versions, that are common to all decision matrices, irrespective of their type. Use standard decision matrices when you don't need the matrix to be part of a group or to be versioned row by row.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Here's how a standard decision matrix is structured and called:

- Matrix with one or more input columns:
  - Each input or set of inputs must be unique
  - Inputs from calling processes, such as attributes and remote options, must match column header names
  - A calling process includes input column header names and input data in one JSON
- Matrix with one or more output columns:
  - Outputs and sets of outputs need not be unique
  - Output column header names and output data are returned in one JSON

**Example** This standard decision matrix has two input columns and one output column.

| **Input Data** | | **Output Data** |
|---|---|---|
| **StudentAge** | **GPA (Number Range)** | **Qualified (Boolean)** |
| 25 | 3 | Not checked |
| 25 | 2 | Not checked |
| 15 | 2 | Not checked |
| 15 | 3 | Not checked |
| 16 | 2 | Not checked |
| 16 | 3 | Checked |

# Grouped Decision Matrices

With grouped matrices, you can group a matrix's data into different versions if several rows have the same column values. For example, in a matrix with age and gender-based transport fares, you can use gender as the matrix key, with male and female as the key values for the versions. This grouping prevents repeating male and female as row values in the matrix and groups the data for males in one version and females in another.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Matrix keys function as one of the input variables for lookups. Based on the key value you enter, only the matching version is considered for the lookup.

Here are some of the benefits of grouped decision matrices:

- Grouped decision matrices are suitable for scenarios that involve using several similar matrices: matrices with the same input headers and the same output headers.
- Grouped decision matrices improve management of data and reduce size requirements.
- Users who keep the decision matrix data up to date can create and activate new versions of a matrix without touching other versions.

**Example** Property and Casualty Sample Use CaseAn auto insurance carrier has different insurance rates for different jurisdictions, namely, states. The carrier wants to manage each jurisdiction separately, but doesn't want to create a separate calculation for each jurisdiction because all the calculations are the same.So the carrier creates a different decision matrix group for each state by using Jurisdiction as the group key.The input and output headers are the same, only the values differ. Each state's decision matrix group can be updated at a different time and new versions can be activated at different times.

# Text Ranges in Decision Matrices

If your matrix has considerable data, use ranges to keep the matrix small. With text ranges, you can concisely specify a range of acceptable values in a matrix column instead of listing individual values.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Note** To resolve any issues with the data in text range columns in a decision matrix, download the data from the version that has issues. Then, create a matrix version with a higher version rank, and upload the downloaded data to the new version.

**Example** You want to assign a group to users based on the letter that their last names begin with. Here's how to set up the columns. Here's how you can set up the matrix:

| **Input Data** | **Output Data** |
|---|---|
| **Last_Name_Starts_With (Text)** | **Group_Assignment (Text)** |
| A | Group 1 |
| F | Group 2 |
| L | Group 3 |
| R | Group 4 |
| W | Group 5 |

The Last_Name_Starts_With is a text range column. Each value in the column defines the start of a range. Each subsequent value represents the start of a new range, and therefore, is excluded from the previous range. The column represents these ranges:

- From A to E
- From F to K
- From L to Q
- From R to V
- From W to Z

According to the input condition in the matrix, here are the group assignments for some sample last names:

- Fowler: Group 2
- Olson: Group 3
- Bates: Group 1
- Zimmerman: Group 5

## Numeric Ranges in Decision Matrices

Use numeric ranges to keep matrices with sizable data small. With number ranges, you can concisely specify a range of acceptable values in a matrix column instead of listing individual values.

REQUIRED EDITIONS

Available in: Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled

The number of rows in a decision matrix that uses a range column is determined by multiplying the number of unique values in the range column and the other input columns. For example, a decision matrix has these input columns: one number range column, and one text column. The number range column has three unique values and the text column has two unique values. For the matrix to include all combinations for these values, the matrix has six rows.

**Note** To resolve any issues with the data in number range columns in a decision matrix, download the data from the version that has issues. Then, create a matrix version with a higher version rank, and upload the downloaded data to the new version.

**Important** You must be consistent in defining the numeric value ranges in a 'Number Range' column across all rows in the matrix.

**Example** A city's public administration office wants to define base fare for local public transport based on a citizen's age and gender. Here's how the office can set up the age criteria, gender, and the corresponding base fare by using a number range column to concisely capture the age values.

| **Input Data** | | **Output Data** |
|---|---|---|
| **Age (Number Range)** | **Gender (Text)** | **Base_Fare (Currency)** |
| 0 | Female | $0 |
| 0 | Male | $0 |
| 10 | Female | $0.75 |
| 10 | Male | $0.75 |
| 18 | Female | $2.5 |
| 18 | Male | $3 |
| 55 | Female | $1.5 |
| 55 | Male | $2 |
| 65 | Female | $0.75 |
| 65 | Male | $1 |

The Age column is a number range. Each value in the Age column defines the start of a range. Each subsequent value represents the start of a new range, and therefore is excluded from the previous range. The column represents these ranges:

- From 0 through 9.99
- From 10 through 17.99

- From 18 through 54.99
- From 55 through 64.99
- 65 or above

According to the input condition in the matrix, here's the base fare for citizens with these ages and genders:

- Age 9, Female: $0
- Age 10, Male: $0.75
- Age 44.5, Female: $2.5
- Age 56, Male: $2
- Age 65, Male: $1
- Age 70, Female: $0.75

If the city's public administration office wants to keep the same base fare for all females below 18, here's how they can set up their input and output data.

| **Input Data** | | **Output Data** |
|---|---|---|
| **Age (Number Range)** | **Gender (Text)** | **Base_Fare (Currency)** |
| 0 | Female | $0 |
| 0 | Male | $0 |
| 10 | Male | $0.75 |
| 18 | Female | $2.5 |
| 18 | Male | $3 |
| 55 | Female | $1.5 |
| 55 | Male | $2 |
| 65 | Female | $0.75 |
| 65 | Male | $1 |

According to the input condition in this matrix, here's the base fare for citizens with these ages and genders:

- Age 9, Female: $0
- Age 10, Male: $0.75
- Age 12, Female: $0
- Age 19, Female: $2.5
- Age 56, Male: $2
- Age 65, Male: $1
- Age 70, Female: $0.75

**Example** This example highlights invalid data in the number range column:

Discounts Based on Gender and Age

| Gender (Input) | Age (Input) | Discount (Output) |
|---|---|---|
| Male | 0 | 5 |
| Male | 15 | 10 |
| Male | 100 | 20 |
| Female | 0 | 6 |
| Female | 20 | 12 |
| Female | 100 | 24 |

The values shown in the table is not allowed because it attempts to define different sets of numeric ranges for the "Age" column based on the "Gender" column (e.g., Age ranges for Male: [0-14.99, 15-99.99, 100+], versus Age ranges for Female: [0-19.99, 20-99.99, 100+]). This may create ambiguous and unpredictable lookup results, as the range boundaries would shift based on other input values.

## Guidelines for Creating and Editing Decision Matrices

Decision matrices require certain criteria to be met to successfully create versions or modify the data in existing versions.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Review these guidelines to prevent errors in creating decision matrices and decision matrix versions.

| Category | Guidelines |
|---|---|
| Matrix and matrix version names | Ensure that the name of a decision matrix and a decision matrix version starts with a letter and doesn't exceed 75 characters, including spaces. Don't start names with numbers and special characters. |
| Matrix columns | Create a header for each column in a decision matrix and avoid spaces in header names.<br><br>Header names are used as input and output variables in expression set versions that call the matrix. |

| Category | Guidelines |
| --- | --- |
| Matrix rows | • Ensure that each row in a matrix is a unique combination of input values. If two rows have the same values in the input columns, only the first row is saved.<br>• Enter data in every cell of every decision matrix row. A decision matrix with empty data fields can't be saved. The exception is fields with the data type Boolean, which appear blank when set to false.<br>• For numeric data in matrix cells, ensure that every value in a column has the same number of decimal places. For example, 20 and 20.00 are considered as different values.<br><br>If a decision matrix version has more than 1,000 rows, you can see only the first 1,000 rows in the decision matrix builder. |
| Editing matrix data | UI-Based Editing: You can view and edit rows in the decision matrix builder if they have fewer than 1,000 rows.<br><br>• To modify a decision matrix version, ensure that the version isn't active. If the version is active, deactivate the version.<br>• Avoid changing the data type of an existing column header. When you change the data type, the data in the column can be corrupted, requiring you to reenter all the data.<br><br>CSV-Based Editing:<br><br>You can edit rows in the decision matrix builder if they have more than 1,000 rows.<br><br>• If you use a CSV file to update the data in a decision matrix version, maintain consistency in the header names in the CSV file and the matrix. |

## Create Standard Decision Matrices

You can create a standard decision matrix by uploading a CSV file, by creating a version of an existing matrix, or manually.

REQUIRED EDITIONS

| |
| --- |
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete decision matrices and matrix versions: | Rule Engine Designer |
| To use decision matrices in Business Rules Engine: | Rule Engine Runtime |

**Important** Plan your decision matrix structure before creating the matrix. After you choose the type of a decision matrix, you can't change the type.

**Create a Standard Decision Matrix Manually**
If a decision matrix has only a few rows of data, creating the matrix manually is easier. When you create a decision matrix, the matrix's first version is created by default.

**Create or Modify Decision Matrices by Using CSV Files**
Use CSV files to create or update decision matrices to save time and effort, especially if you're dealing with sizable data. Create the CSV file in the specified format to successfully add data to a decision matrix version.

**Create a Version of a Standard Decision Matrix**
Creating a version of an existing decision matrix is quicker than creating a matrix from nothing. You can create a new version from a matrix version that has at least one input and one output columns.

**Create a Decision Matrix Version from an Existing Version**
To copy all the data from a selected decision matrix version to a new version easily and quickly, create the version from an existing matrix version.

## Create a Standard Decision Matrix Manually

If a decision matrix has only a few rows of data, creating the matrix manually is easier. When you create a decision matrix, the matrix's first version is created by default.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete decision matrices and matrix versions: | Rule Engine Designer |
| To use decision matrices in Business Rules Engine: | Rule Engine Runtime |

Create a decision matrix named StudentDiscountEligibility. This matrix determines the eligibility of students to receive a discount on fees based on a combination of their age and Grade Point Average (GPA).

1. Click ⠿, and then find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Lookup Tables**.
3. Click **New**.
4. Select **Decision Matrix** | **Next**.
5. For name, enter `StudentDiscountEligibility`, and for Type, select **Standard**.
6. Save your changes.
7. On the Related tab on the decision matrix record page, click version 1 of the matrix.
8. On the Matrix tab, click **Add Column**, and then create three columns with these settings.

| Column Header | Header Type | Data Type |
|---|---|---|
| StudentAge | Input | Number Range |
| GPA | Input | Number Range |
| Qualified | Output | Boolean |

9. Click **Done**.
10. To create a row, click **Add Row**.
11. To add row data, click a table cell, click ✎ in the cell, and then enter the value
    Add these rows and values.

| StudentAge | GPA | Qualified |
|---|---|---|
| 15 | 2 | Not checked |
| 15 | 3 | Not checked |
| 16 | 2 | Not checked |
| 16 | 3 | Checked |
| 25 | 2 | Not checked |
| 25 | 3 | Not checked |

12. Save your changes.
13. Activate your decision matrix version:
    Expression sets can call only active matrix versions.
    a. From the actions menu for the decision matrix version, select **Edit** | **Enabled**.
    b. Save your work.

The final decision matrix looks like this.

## Create or Modify Decision Matrices by Using CSV Files

Use CSV files to create or update decision matrices to save time and effort, especially if you're dealing with sizable data. Create the CSV file in the specified format to successfully add data to a decision matrix

version.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete decision matrices and matrix versions: | Rule Engine Designer |
| To use decision matrices in Business Rules Engine: | Rule Engine Runtime |

**Considerations for Preparing a CSV File for Upload**
Let's look at the prerequisites and considerations to prepare your CSV file. Ensure that the file complies with the specified format to make sure that your CSV upload goes smoothly and that all the data is added or updated successfully.

**Create a Standard Decision Matrix by Using a CSV File**
Use a CSV file to create decision matrices, especially if you're dealing with sizable data, and save the time and effort of manually creating column headers and adding row data.

**Modify a Standard Decision Matrix by Using a CSV File**
Use a CSV file to update the data in decision matrices to save the time and effort of manually updating the data, especially if you're dealing with sizable data. When you use a CSV file to update an existing matrix version, only row data is updated.

## Considerations for Preparing a CSV File for Upload

Let's look at the prerequisites and considerations to prepare your CSV file. Ensure that the file complies with the specified format to make sure that your CSV upload goes smoothly and that all the data is added or updated successfully.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

### Prerequisites

- Ensure that each column has a header and that the header names have no spaces. Header names are used as input and output variables in the expression set that calls the decision matrix.

- If you use a CSV file to update data in an existing decision matrix version, ensure that the header names in the file match the header names in the matrix.
- If you use a CSV file to create a new decision matrix version, ensure that each row is a unique combination of input values.
- For numeric data in your CSV file, ensure that every value in the same column has the same number of decimal places. When a rule calls a decision matrix, ensure that the values in the Number, Currency, and Percent type columns match exactly. For example, 20 and 20.00 don't match.

## Considerations

- You can upload up to 50,000 from a CSV file to a matrix version at a time.
- After uploading a CSV file, you can't rename columns if the matrix has more than 1,000 rows.
- Multiple rows can contain the same output value or combination of values for different input values.
- If two rows in the CSV file have the same input values, only the first row is uploaded.
- If the cell values in the CSV file start with the special characters =, +, -, or @, the cell values are prefixed with a single quote (') in the decision matrix version. If the numeric columns in the CSV file have negative integers in cells, the minus sign for the cell values is retained in the matrix version without prefixing a single quote.
- When you use a CSV file to update a decision matrix version, the file is scanned for new rows. The file is also scanned for rows that have the same input values but different output values from the matrix. You can select one of these options:
  - **Overwrite Duplicate Rows**: If the CSV file rows and the matrix version rows have the same input values but different output values, this option updates the matrix version with the output values and new rows from the CSV file.
  - **Import New Data**: Ignores rows with duplicate input values and uploads only the new rows from the CSV file.
- When you import a CSV file to the decision matrix, any field value that begins with the +, -, @, or = operator is appended with a single quote. Prefix the values to make sure that the values are treated as text and not processed as formulas.

### Create a Standard Decision Matrix by Using a CSV File

Use a CSV file to create decision matrices, especially if you're dealing with sizable data, and save the time and effort of manually creating column headers and adding row data.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete decision matrices | Rule Engine Designer |

| USER PERMISSIONS NEEDED | |
|---|---|
| and matrix versions: | |
| To use decision matrices in Business Rules Engine: | Rule Engine Runtime |

1. Click ⠿, and then find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Lookup Tables**.
3. Click **New**.
4. Select **Decision Matrix**, and then click **Next**.
5. Enter a name for the decision matrix.
6. For Type, select **Standard**.
7. Save your changes.
8. On the Related tab of the record page for your new decision matrix, click version 1 of the matrix.
9. On the Matrix tab of the version, click **Upload CSV File** | **Upload Files**.
10. Select the CSV file from your computer.
11. Select the header type for each column as **Input** or **Output**.
    The default is Input.
12. Set the data type for each column.
    The default is Text.
13. Click **Upload** | **Done**.
14. To see the data rows, refresh the browser.

    **Note** If the number of rows exceeds 1,000, only the first 1,000 rows are shown. Any data errors appear in the affected cells.

## Modify a Standard Decision Matrix by Using a CSV File

Use a CSV file to update the data in decision matrices to save the time and effort of manually updating the data, especially if you're dealing with sizable data. When you use a CSV file to update an existing matrix version, only row data is updated.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete decision matrices and matrix versions: | Rule Engine Designer |
| To use decision matrices in Business Rules Engine: | Rule Engine Runtime |

1. Click ⠿, and then find and select **Business Rules Engine**.

2. Click the app navigation menu, and then select **Lookup Tables**.
3. Select a decision matrix.
4. On the Related tab of the matrix, select the version for which you want to modify the data.
5. Click **Upload CSV File** | **Upload Files**.
6. Select the CSV file from your computer.
7. Select an option:

| | |
|---|---|
| **Overwrite Duplicate Rows** | If the CSV file rows and the matrix version rows have the same input values but different output values, this option updates the matrix version with the output values and new rows from the CSV file. |
| **Import New Data** | Ignores rows with duplicate input values and uploads only the new rows from the CSV file. |

8. Click **Upload** | **Done**.
9. To see the data rows, refresh the browser.

**Note** If the number of rows exceeds 1,000, only the first 1,000 rows are shown. Any data errors appear in the affected cells.

## Create a Version of a Standard Decision Matrix

Creating a version of an existing decision matrix is quicker than creating a matrix from nothing. You can create a new version from a matrix version that has at least one input and one output columns.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete decision matrices and matrix versions: | Rule Engine Designer |
| To use decision matrices in Business Rules Engine: | Rule Engine Runtime |

1. Click ⠿, and then find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Lookup Tables**.
3. Click the link of the decision matrix for which you're creating another version.
4. On the record page of the matrix, on the Related tab, click **New**.
5. Enter a name and a version number for your new decision matrix version.
   For standard matrices, Salesforce recommends specifying the decision matrix name with a `Vn` suffix

and a number that's one greater than the current version number.

6. Add a start date time and, if necessary, an end date time for the version.

   **Tip** Set the start date time for the new version to the end date time of the previous version. Or set a start date time before the end date time of the previous version, and then set different rank numbers for the versions.

7. Enter a rank.
8. Save your work.
9. Click the link for the version that you created.
10. Enter the row data:
    - To enter new row data from a CSV file, click **Upload CSV File**, and then upload the appropriate file from your computer.

      **Important** Ensure that the column names in the CSV file and the decision matrix match exactly.

      If after uploading the CSV file you don't see the rows, refresh the browser.
    - To enter row data from a previous version of the matrix:
      - Download the decision matrix data to a CSV file.
      - Click **Upload CSV File**, and then upload the appropriate file from your computer.
    - To enter data manually:
      - Click **Add Row**.
      - To add row data, click a table cell, click in the cell, and then enter the value.
11. Save your changes.
12. Activate your decision matrix version:
    Expression sets can call only active matrix versions.
    a. From the actions menu for the decision matrix version, select **Edit** | **Enabled**.
    b. Save your work.

## Create a Decision Matrix Version from an Existing Version

To copy all the data from a selected decision matrix version to a new version easily and quickly, create the version from an existing matrix version.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete decision matrices | Rules Engine Designer |
| To run decision matrices | Rules Engine Runtime |

1. Click ⠿, and then find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Lookup Tables**.
3. Select a decision matrix.
4. On the Related tab of the record page for the decision matrix, select the version from which you want to create a new version.
5. In the details section of the selected version, click **New Version**.
6. Select an option:
   - If you don't want to include rows from the selected version in the new version, select **Include column headers**.
   - To include both column headers and rows in the new version, select **Include column headers and rows.**

     **Note** You can copy up to 50,000 rows simultaneously. The time taken to create a version depends on the number of rows that you copy.
7. Enter a name for the new version.
8. Add a start date time and, if necessary, an end date time for the version.
9. Enter a rank number.
   Ensure that the rank is unique within the decision matrix.
10. Enter a version number.
11. Click **Done**.
12. To activate your decision matrix version, from the actions menu for the new decision matrix version, select **Edit** | **Enabled**.
    Expression sets can call only active matrix versions.
13. Save your work.

## Create Grouped Decision Matrices

You can create a decision matrix that groups rows in versions based on a group key and, if necessary, a sub group key. Versions in a grouped decision matrix can have different start date time values. When a grouped matrix is invoked, the group or subgroup keys are considered like input variables, for which you must provide values for lookup.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete decision matrices and matrix versions: | Rule Engine Designer |
| To use decision matrices in Business Rules Engine: | Rule Engine Runtime |

**Considerations for Grouped Decision Matrix Versions**
For effective creation and management of grouped decision matrix versions, ensure that versions meet the specified criteria.

**Create and Configure a Grouped Decision Matrix**
When you create a grouped decision matrix, the first grouped version is created by default. This version has the same name as the matrix, but is suffixed with V1. Edit the name to give the version a descriptive name, and add a group key or a sub group key value. When you configure the first version of a grouped decision matrix, you also specify the column headers for all versions. You can’t create other group versions before creating the column headers in the first version.

**Create and Configure Subsequent Versions of the Grouped Decision Matrix**
After creating a grouped decision matrix and configuring the matrix's first version, you can create subsequent versions with different key or subgroup key values and varying row data. You can't create column headers for subsequent versions of a grouped decision matrix because column headers are inherited from the first version.

**How to Design a Grouped Decision Matrix (Example)**
When your business requires you to have multiple similar matrices with the same input and output headers, consider creating a grouped matrix. Learn how to create a grouped decision matrix and call it from an expression set version.

## Considerations for Grouped Decision Matrix Versions

For effective creation and management of grouped decision matrix versions, ensure that versions meet the specified criteria.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

- Ensure that the name of a decision matrix and a decision matrix version starts with a letter and doesn't exceed 75 characters, including spaces. Don't start names with numbers and special characters.
- Ensure that input and output column headers of grouped matrix versions are the same. If a group requires additional columns, create separate matrices.
- Ensure that numeric or alphabetic ranges of inputs are the same.
- Only two levels of keys are required.
- The group key and sub group key aren't ranges. The group key value and sub group key value are usually text inputs, such as jurisdiction codes or product codes.

## Create and Configure a Grouped Decision Matrix

When you create a grouped decision matrix, the first grouped version is created by default. This version has the same name as the matrix, but is suffixed with V1. Edit the name to give the version a descriptive

name, and add a group key or a sub group key value. When you configure the first version of a grouped decision matrix, you also specify the column headers for all versions. You can't create other group versions before creating the column headers in the first version.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete decision matrices and matrix versions: | Rule Engine Designer |
| To use decision matrices in Business Rules Engine: | Rule Engine Runtime |

1. From the App Launcher ( ), find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Lookup Tables**.
3. Click **New**.
4. Select **Decision Matrix** | **Next**.
5. Enter a name for the decision matrix.
6. For Type, select **Grouped**.
7. Enter a group key, and then, if necessary, a sub group key.
   For example, to group by geographic region, you can set the group key to Country and the sub group key to State.
8. Save the matrix.
   The first grouped version of the matrix is created by default.
9. On the record page of the decision matrix, on the Related tab, select the first version.
10. In the version header, click **Edit**.
11. To enter a description for the group, edit the version name.
    For example, if the matrix is named Shipping Fees and the group key is State, you can name the version Shipping Fees CA.
12. Add a start date time, and then, if necessary, an end date time for the version.
    The default start date time is today.
13. Enter a rank number.
14. Enter a group key value.
    If you specified a sub group key at the time of creating the matrix, enter a sub group key value. For example, if you set the group key to State while creating the grouped matrix, your group key value can be California.

    **Tip** You can't add columns and rows before specifying the key values.

    When you invoke a grouped matrix that has multiple simultaneously active versions, the group and sub group key values that you enter determine which version is considered.
15. Save your changes.
16. Enter data for the group:

a. On the Matrix tab, click **Add Column**, and then specify the column header name, header type, and data type.
   Don't include spaces in the column header.
b. To add more columns, in the Add Column window, click **Add Column**, specify the details for additional columns, and then click **Done**.
c. To add rows, click **Add Row**.
d. To add row data, click a table cell, click in the cell, and then enter the value.

To enter data by uploading a CSV file, click **Upload CSV File**.

**Note** If after uploading the CSV files you don't see the rows, refresh the browser.

17. Save the matrix version.
18. To activate the version, from the matrix version header, click **Activate**.
    Expression sets and other processes can call only matrices that have at least one active version.

## Create and Configure Subsequent Versions of the Grouped Decision Matrix

After creating a grouped decision matrix and configuring the matrix's first version, you can create subsequent versions with different key or subgroup key values and varying row data. You can't create column headers for subsequent versions of a grouped decision matrix because column headers are inherited from the first version.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete decision matrices and matrix versions: | Rule Engine Designer |
| To use decision matrices in Business Rules Engine: | Rule Engine Runtime |

1. Click , and then find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Decision Matrices**.
3. Click the link of the grouped decision matrix for which you want to create additional versions.
4. On the record page of the matrix, on the Related tab, click **New**.
5. Enter a name for the version.
   For example, if the matrix is named Shipping Fees and the group key value is State, then you can name the version for New York as Shipping Fees NY.
6. Enter a version number.
   To create the version number, Salesforce recommends adding one to the current version number. For example, if the current version number is 1, then the new number recommended is 2.
7. Add a start date time and, if necessary, an end date time for the version.

The default start date time is today.

8. Enter a rank number.
9. Enter a group key value.
   If you specified a sub group key while creating the grouped decision matrix, enter a sub group key value. For example, if your group key is State and you're creating this version for including data on Shipping Fees for NY, your group key value can be New York.

   **Warning** If you add columns or rows before specifying the key values, you can't save the data.
10. Save your changes.
11. Click the link for the version that you created.
12. Add row data:
    - To enter row data from a CSV file, click **Upload CSV File**.

      **Important** Ensure that the column names in the CSV file and the grouped decision matrix are a perfect match. See Create a Standard Decision Matrix by Using a CSV File.

      If after uploading the CSV file you don't see the rows, then refresh the browser.
    - To enter row data manually:
      - On the Matrix tab, click **Add Row**.
      - Click a table cell, click in the cell, and then enter the value.
13. Save your changes.
14. Activate your decision matrix version:
    Expression sets and other processes can call only active matrix versions.
    a. From the matrix version header, click Show Actions ( ▼ ), and then select **Edit** | **Enabled**.
    b. Save your changes.

## How to Design a Grouped Decision Matrix (Example)

When your business requires you to have multiple similar matrices with the same input and output headers, consider creating a grouped matrix. Learn how to create a grouped decision matrix and call it from an expression set version.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

1. **Create a Grouped Matrix for Car Rental Plans**
   Store the details of car rental plans for different cities using grouped matrix versions. City names are used as the values for each version's group key.
2. **Call the Rental Plans Matrix from an Expression Set Version**
   Add a lookup table step to an expression set version to call the grouped matrix. Next, simulate the version to test if the grouped matrix works as expected.

## Create a Grouped Matrix for Car Rental Plans

Store the details of car rental plans for different cities using grouped matrix versions. City names are used as the values for each version's group key.

REQUIRED EDITIONS

| USER PERMISSIONS NEEDED | |
|---|---|
| To create a grouped decision matrix and configure and activate its versions: | Rule Engine Designer |

1. From the navigation menu in Business Rules Engine, select **Lookup Tables**.
2. Click **New**.
3. Select **Decision Matrix** | **Next**.
4. For name, enter `Car Rental Fees`.
5. For type, select **Grouped**.
6. For group key, enter `City`.
   The matrix and its first version are created.
7. On the Related tab of the matrix, select **Car Rental Fees V1**.
8. Edit the version.
   a. Click **Edit**.
   b. Change the version name to `Car Rental Fees NYC`.
   c. In Group Key Value, enter `New York City`.
      Use this version to store the data specific to New York City.
   d. Save the version.
9. Configure the version data.
   a. To configure the column headers, click **Add Column**.
   b. Create these columns, and click **Done**.

   c. To add row data, click **Add Row** for each new row that you want to add, and configure these rows.

      The values 0 and 7 in the Rental_Days column represent the ranges: 0–6 and 7 and above respectively.

   d. Save and activate the version.
10. Create a version to store the car rental details for San Francisco.
    a. In the header of the Car Rental Fees NYC version, click **New Version**.
    b. Select **Include column headers and rows**.
    c. For name, enter `Car Rental Fees SF`.
    d. Enter `2` for the version rank and number.
    e. Save the version.
    f. To modify the row data, click ✎ in a cell and change the cell's value.

Here's the modified row data for the San Francisco version.

g. Save and activate the version.

11. Create a version to store the car rental details for Chicago.
    a. In the header of the Car Rental Fees SF version, click **New Version**.
    b. Select **Include column headers and rows**.
    c. For name, enter `Car Rental Fees Chicago`.
    d. Enter `3` for the version rank and number.
    e. To modify the row data, click in a cell and change the cell's value.
    f. In this example, Chicago doesn't have rental plans for SUVs. To delete rows with Car_Type as SUV, select the row checkboxes, and click **Delete Rows**.

    Here's the modified row data for the Chicago version.

    g. Save and activate the version.

## Call the Rental Plans Matrix from an Expression Set Version

Add a lookup table step to an expression set version to call the grouped matrix. Next, simulate the version to test if the grouped matrix works as expected.

REQUIRED EDITIONS

| USER PERMISSIONS NEEDED | |
|---|---|
| To create and activate an expression set: | Rule Engine Designer |
| To run an expression set | Rule Engine Designer<br>OR<br>Rule Engine Runtime |

1. From the App Launcher ( ), find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Expression Sets**.
3. Click **New**.
4. Name the expression set `Get Car Rental Fees`.
5. Select the Default usage type and save the expression set.
6. In the Expression Set Versions area, select **Get Car Rental Fees V1**.
7. Click , and enter `1` as the version rank.
8. To call the grouped decision matrix from the expression set version, add a lookup table step.
    a. Click in the expression set canvas, and select **Lookup Table**.
    b. In the Lookup Table Details field, start typing `Car Rental`, and select **Car Rental Fees**.

The group key of the matrix, City, appears as one of the input variables of the selected grouped matrix.

9. Save the version.
10. To verify that the grouped matrix works as expected, simulate the version with a few sample input values.
    a. Click **Simulate**.
    b. In Car_Type, enter `Hatchback`, in City, enter `San Francisco`, and in Rental_Days, enter `10`.
    c. Simulate the version.

    Here, San Francisco (3) is one of the values of the matrix key City. Based on this input value, the matrix version (1) whose key value is San Francisco is considered, and the output value (2) that corresponds to the values of the input variables is returned.

    Simulate the expression set version with a few different values for the matrix's input variables.

To activate the expression set version, click **Back**, and then click **Activate**.

**Note** You can invoke only the expression sets that have at least one active version from calling processes such as Flow.

## Manage and Maintain Decision Matrices

Learn how to edit decision matrix rows and columns, sort and filter matrix rows, and export matrices to CSV or DMN files.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

### Edit Decision Matrix Rows and Columns

You can't edit an active decision matrix version. To edit an active decision matrix version, disable the version first. You can edit decision matrix columns if only one version of a decision matrix exists and the version has fewer than 1,000 rows.

### Sort and Filter Decision Matrix Rows

You can sort decision matrix rows in ascending or descending order of the values in a column. You can also filter rows based on column values.

### Download Decision Matrix Data to a CSV File

Download a decision matrix as a CSV file to reuse the data to create another decision matrix or to modify a large volume of data before reuploading the CSV file.

**Export and Download Matrix Versions Exported to DMN Files**
Export decision matrix versions to Decision Model Notation (DMN) files from the Business Rules Engine app. The exported files can be downloaded from the record detail page of a decision matrix version. DMN is a standardized specification for writing business rules. With DMN files, you can reuse matrix versions created in Business Rules Engine in any tool that supports DMN files.

## Edit Decision Matrix Rows and Columns

You can't edit an active decision matrix version. To edit an active decision matrix version, disable the version first. You can edit decision matrix columns if only one version of a decision matrix exists and the version has fewer than 1,000 rows.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete decision matrices and matrix versions: | Rule Engine Designer |
| To use decision matrices in Business Rules Engine: | Rule Engine Runtime |

**Warning** Don't edit matrix column and row objects directly. Use the table on the Decision Matrix Version page.

1. Click the app navigation menu of Business Rules Engine, and then select **Decision Matrices**.
2. Click the decision matrix that you want to edit.
3. On the record page of the decision matrix, on the Related tab, click the version for which you want to modify data.
4. To edit or reorder the columns, on the Matrix tab of the version, click the actions menu, and then select **Edit Columns** or **Reorder Columns**.

**Important** Salesforce recommends not changing the Data Type of an existing column header. When you change the Data Type, the data in the column can be corrupted, requiring you to reenter all the data.

5. To modify row data, click a table cell, click in the cell, and then enter the new value.

# Sort and Filter Decision Matrix Rows

You can sort decision matrix rows in ascending or descending order of the values in a column. You can also filter rows based on column values.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete decision matrices and matrix versions: | Rule Engine Designer |
| To use decision matrices in Business Rules Engine: | Rule Engine Runtime |

1. Click , and then find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Decision Matrices**.
3. Click the link for the decision matrix that you want to open, and then click the link for the version of the decision matrix.
4. To sort rows in the decision matrix version:
   a. Click a column header.
   Clicking repeatedly toggles between ascending and descending sort orders. You can sort numerically or alphabetically.

5. To create and apply row filters:
   a. On the Matrix tab of the version, click .
   b. In the Filters window, click **Add Filter**.
   c. From the Field dropdown list, select a column name.
   d. Select an operator.
   e. In the Value field, enter the value on which to filter.
   For a boolean field, select **True** or **False**.
   f. Click **Done**.
   g. To filter the rows, click **Apply**.

   **Note** For a large matrix, this process can take a few minutes. Including row edits in filters can also take time.

   **Note** Filters persist until you remove them.

6. To remove a filter:
   a. Click .
   b. In the Filters popup, click **X** for each filter that you want to remove.

c. When you finish removing filters, click **Apply**.

## Download Decision Matrix Data to a CSV File

Download a decision matrix as a CSV file to reuse the data to create another decision matrix or to modify a large volume of data before reuploading the CSV file.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete decision matrices and matrix versions: | Rule Engine Designer |
| To use decision matrices in Business Rules Engine: | Rule Engine Runtime |

1. On the Decision Matrix tab, click the name of the matrix.
2. On the record page of the matrix, on the Related tab, click the link for the matrix version.
3. Click **Generate CSV File**.
4. From the App Launcher, find and select **Files**.
5. Find the generated CSV file.
   The file name syntax is `matrix version name_timestamp.csv`. For example: `StudentDiscountEligibility V1_2021-08-04 14_37_45.csv`.
6. From the Show Actions menu for the file, click **Download**.
   The CSV file downloads to your computer.

## Export and Download Matrix Versions Exported to DMN Files

Export decision matrix versions to Decision Model Notation (DMN) files from the Business Rules Engine app. The exported files can be downloaded from the record detail page of a decision matrix version. DMN is a standardized specification for writing business rules. With DMN files, you can reuse matrix versions created in Business Rules Engine in any tool that supports DMN files.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To view and download decision matrix versions exported to DMN files: | Rule Engine Designer |

You can view the matrix versions exported to DMN files in the Related tab of a matrix version, only if the version has been exported to a DMN file using connect API.

1. Click ⠿, and then find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Lookup Tables**.
3. Select a decision matrix.
4. From the Related tab of the matrix, open the version that is to be exported to a DMN file.
5. Click **Export to DMN File**.
   When the export is complete, the DMN file is available for download from the Related tab of the version.
6. From the Related tab of the version, select the DMN file.
7. Download the file.

**Limitations for Exporting Matrix Versions to DMN Files**
Decision matrix versions can be exported to Decision Model Notation (DMN) files from the Business Rules Engine app or by using Connect REST API. Before you attempt to export a matrix version, review these limitations.

### Limitations for Exporting Matrix Versions to DMN Files

Decision matrix versions can be exported to Decision Model Notation (DMN) files from the Business Rules Engine app or by using Connect REST API. Before you attempt to export a matrix version, review these limitations.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

- Decision matrix versions can't be exported to DMN files if the version columns are of the data type Text Range or Number Range.
- Columns with Currency and Boolean data types are converted to Number data type in the exported file.
- The export fails if the DMN file size exceeds 2 GB.

## Decision Matrix Migration Considerations

Here are a few typical migration scenarios and recommendations on how to maintain decision matrices

in your target org for a successful migration.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

The result of decision matrix migration depends on whether the decision matrix and its versions exist in the target org and on the status of such versions.

**Note** When you migrate a decision matrix version, only the column headers of the version migrate to the target org. Row data isn't migrated. Before you migrate a decision matrix version, specify a rank and activate the version in the source org. Then, migrate the version or the entire decision matrix to the target org. The migrated decision matrix version is invoked in the target org based on the specified rank. Migration fails if the target org already contains a decision matrix version of the specified rank.

| Decision Matrix Configuration | Migration Behavior | Migration Recommendations |
|---|---|---|
| The target org doesn't have the decision matrix. | The decision matrix and its versions migrate successfully. | • Deploy the decision matrix in the target org.<br>• Migrate the decision matrix versions independently. |
| The target org has the decision matrix with one or more versions, where some or all the versions are active. | The migration fails. | Deactivate active decision matrix versions in the target org and then try again. |
| The target org has the decision matrix but all matrix versions are inactive. | The decision matrix migrates successfully and all the versions from the source org replace the versions in the target org. | Migrate individual decision matrix versions separately, rather than migrating an entire decision matrix with its versions together.<br><br>This approach gives you more flexibility to evaluate what a decision matrix version contains and change your migration approach accordingly. |
| The target org has the decision matrix with only a subset of the versions from the source org, and all the versions are inactive. | • The decision matrix migrates successfully and all the versions from the source org replace the inactive versions in the target org.<br>• The versions that didn't exist in the target org are added to the target org under the | |

| Decision Matrix Configuration | Migration Behavior | Migration Recommendations |
|---|---|---|
| | parent decision matrix. | |

## Scenario 2: Migrate a Decision Matrix Version

Here, you're migrating a decision matrix version independently, and not the entire decision matrix to which the version belongs.

**Note** When you migrate a decision matrix version, only the column headers of the version migrate to the target org. Row data isn't migrated.

| Decision Matrix Configuration | Migration Behavior | Migration Recommendations |
|---|---|---|
| The target org has the same decision matrix version in active state. | The migration fails. | Deactivate the decision matrix version in the target org and then try again.<br><br>The migrated decision matrix version from the source org overwrites the inactive decision matrix version in the target org. |
| The target org the same decision matrix version, but in inactive state. | The version from the source org migrates successfully and overwrites the inactive version in the target org. | We recommend this migration approach. |
| The target org doesn't have the decision matrix to which the version from the source org belongs. | The migration fails. | Deploy the decision matrix in the target org and then migrate the version again. |

# Decision Matrices for Developers

Developers in your Salesforce org can employ decision matrix developer resources to use and extend decision matrices.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine |

| |
|---|
| enabled |

## Connect REST API

- Look up decision matrices
- Export decision matrix versions to DMN files

## Metadata API

- Create decision matrices

## Invocable Actions

- Invoke a decision matrix in a flow using the decision matrix action

# Decision Tables for Business Rules Engine

Decision tables are complex lookup tables that read business rules with multiple inputs, and return multiple outputs for records in your Salesforce org, or for the values that you specify. Rules are records in a standard object, custom object, or a custom metadata type.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Note** If you use decision tables as part of Loyalty Management or Rebate Management, see Decision Table.

Create decision tables in Business Rules Engine, and use them in flows or expression sets. If you use custom or third-party tools to create workflows, use Connect APIs to integrate decision tables in your workflows.

**Important** If a decision table returns multiple outputs, the expression set reads only the first result from that decision table.

**Important** If you are using a custom profile in Salesforce, ensure that you have enabled the View Setup and Configuration permission from the Setup menu to have access to decision tables.

### Decision Table Terms in Business Rules Engine

Here are the key terms used in decision tables.

### Supported Data Types and Operators in Decision Tables

Decision tables support specific field types as inputs and outputs. A field's data type determines the field's available operators.

### How Grouping Works in Decision Tables

When you create a decision table, you can group the table's rows by input fields. Grouping splits each unique combination of values in the grouping field rows into their individual group. By categorizing the data logically, when you provide input values from the grouping fields for lookup, the lookup considers only the group associated with that value. Grouping helps streamline the lookup process by focusing only on the relevant information for the given input values.

### How Source Filters Work in Decision Tables

Narrow down the source object rows by using source filters when the source object has many rows. Use source filters to consider only relevant records from a large pool of records for decision-making.

### Input Field Conditions in Decision Tables

When you create a decision table, select a source object for the decision table. Next, select the fields that you want to use as input and output. Finally, specify the condition logic for the input fields so that the decision table can evaluate input values or records from the object against the logic that you specify.

### How Result Filters Work

When you create a decision table, you can specify how you want the table's results to be filtered. The filter you choose determines whether you get a single matching result or multiple. If there are multiple results, filtering also determines the order in which the results appear.

### How Null Values Work in Decision Tables

Use the Consider Null Values for Lookup option to handle the behavior of how rows with null values are treated during lookup. When you enable this option, rows with null values in the decision table are matched only when the input explicitly includes null, which ensures precise matches. When you disable this option, and any of the rows have null values, then the column values that pertain to the null value are also considered as null.

### Considerations for Creating Decision Tables

Before you plan and create decision tables in your Salesforce org, review these considerations.

### Decision Table Asynchronous Events

Trigger actions when decision table operations like Activation, Refresh, or CSV uploads complete asynchronously.

### Best Practices for Optimum Performance and Scalability of Decision Tables

When you use decision tables for your decision-making process, it's important that you optimize them for scalability and performance. Scalability refers to the ability of decision tables to handle large datasets without compromising performance. Performance makes sure that decision tables operate at their highest efficiency with minimal latency and reduced processing time.

### How to Choose Between Standard and Advanced Decision Table Types

Optimize your rule management based on the size and complexity of data by selecting the right decision table type. This categorization improves system performance and storage efficiency, ensuring fast and scalable decision-making processes. Customize your tables to improve flexibility and resource allocation.

### Create a Decision Table

Create decision tables to read business rules and use the matching results in expression sets or flows. When you create a decision table, you can set the volume and complexity required for the table to match your data and business needs. You can either use a combination of source objects and related objects or a CSV file to define the source data for your decision tables. You can also use ready-to-use, industry-based templates to quickly create a decision table.

### Activate a Decision Table

You can use decision tables in expression sets in Business Rules Engine, and in flows only when the decision tables are active.

### Refresh a Decision Table

To meet evolving business requirements, you add, update, or remove decision table rows. If you update the object or the custom metadata type that contain the row data, refresh the decision table to ensure that the outcomes are based on the latest data.

### Tips to Manage Decision Tables

You can modify or delete decision tables used in expression set versions to keep your rules up to date with changing business requirements.

### Considerations for Migrating Decision Tables

Before you plan to migrate a decision table from a source org to a target org, review the migration considerations.

### Decision Tables for Developers

Developers in your Salesforce org can employ developer resources to use and extend decision tables.

## Decision Table Terms in Business Rules Engine

Here are the key terms used in decision tables.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Note** If you use decision tables as part of Loyalty Management or Rebate Management, see Decision Table.

Key Terms in Decision Tables

| Term | Description |
|---|---|
| Business Rules | Set of rules based on which a decision table provides outcomes. The business rules are part of a standard object, custom object, or custom metadata type. |
| Conditions | Fields from the business rules that the decision |

| Term | Description |
| --- | --- |
| | table uses to evaluate records. |
| Operator | Identifies how values are evaluated against the input field. Select an operator for each business rule field that's used as a decision table input. |
| Results | When one or more rules match, the decision table return's the field's values as outcomes. |
| Input Fields Condition | A condition for the input fields. The decision table input fields evaluate values against the condition. When the condition matches, the decision table provides an outcome. |
| Advanced | A type of decision table that supports a large volume of rows and lesser complexity. |
| Standard | A type of decision table that supports complex conditions. |
| Custom Logic | Must match for the decision table to provide outcomes. |

## Supported Data Types and Operators in Decision Tables

Decision tables support specific field types as inputs and outputs. A field's data type determines the field's available operators.

REQUIRED EDITIONS

| |
| --- |
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Note** If you use decision tables as part of Loyalty Management or Rebate Management, see Decision Table.

Here are the supported data types and operators in decision tables:

| Data Type | Supported Operators |
| --- | --- |
| Text | • Equals<br>• Not Equals |

| Data Type | Supported Operators |
| --- | --- |
| | • Matches<br>• Does Not Match<br>• Contains |
| Number | • Equals<br>• Not Equals<br>• Less Than<br>• Less or Equal<br>• Greater Than<br>• Greater or Equal |
| AutoNumber | • Equals<br>• Not Equals<br>• Matches |
| Percentage | • Equals<br>• Not Equals<br>• Less Than<br>• Less or Equal<br>• Greater Than<br>• Greater or Equal |
| Date | • Equals<br>• Not Equals<br>• Less Than<br>• Less or Equal<br>• Greater Than<br>• Greater or Equal |
| Date/Time | • Equals<br>• Not Equals<br>• Less Than<br>• Less or Equal<br>• Greater Than<br>• Greater or Equal |

| Data Type | Supported Operators |
| --- | --- |
| Currency | • Equals<br>• Not Equals<br>• Less Than<br>• Less or Equal<br>• Greater Than<br>• Greater or Equal |
| Checkbox | • Equals<br>• Not Equals |
| Picklist | • Equals<br>• Not Equals<br>• Matches |
| Picklist (Multi-Select) | • Exists In<br>• Does Not Exist In<br><br>For multi-select picklist fields, you can pass only one input value during a lookup.<br><br>For example, imagine you have a multi-select picklist field called "Genre" that stores the genres applicable to a book. This field has the following picklist values:<br><br>• Genre1 includes ART and LITERATURE<br>• Genre2 includes SCIENCE, FINE ART, and FICTION<br>• Genre3 includes LITERATURE and SCIENCE FICTION<br><br>When you pass an input value for EXISTS IN:<br><br>If the input is "Art," there is no match because the lookup is case sensitive. If the input is "ART," it matches Genre1. If the input is "FINE ART," it matches Genre2. If the input is "LITERATURE," it matches both Genre1 and Genre3.<br><br>When you pass an input value for DOES NOT |

| Data Type | Supported Operators |
| --- | --- |
| | EXIST IN:<br><br>If the input is "MYSTERY," it matches all genres because "MYSTERY" is not in any of the picklist values. If the input is "LITERATURE," it matches Genre2 because "LITERATURE" is not in Genre2. If the input is "Art," it matches all genres because the lookup is case sensitive and "Art" does not match "ART." |
| Lookup | • Equals<br>• Not Equals |
| Master Detail | • Equals<br>• Not Equals |
| Email | • Equals<br>• Not Equals<br>• Matches |
| URL | • Equals<br>• Not Equals<br>• Matches |

### Matches Operator in Decision Tables

Use the Matches operator to retrieve an output if the query value is an exact match or a subset of the decision table row value. The Matches operator is case-sensitive and applies only to input fields in the decision table.

### Does Not Match Operator in Decision Tables

Use the Does Not Match operator to retrieve an output if the query value does not match the decision table row value and is also not a subset of the decision table row value. The Does Not Match operator is case-sensitive and applies only to input fields in the decision table.

### Contains Operator in Decision Tables

Use the Contains operator to retrieve an output if the decision table row value is an exact match or subset of the query value. The Contains operator is case-sensitive and applies only to input fields in the decision table.

# Matches Operator in Decision Tables

Use the Matches operator to retrieve an output if the query value is an exact match or a subset of the decision table row value. The Matches operator is case-sensitive and applies only to input fields in the decision table.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Here are some examples of how the Matches operator works for the data types that support the operator.

| Data Type | Description | EXAMPLE |
|---|---|---|
| Text | Matches your input string to an entire value or a part of it in decision table cells and returns the output values of the rows that the cells belong to. The output is sorted by the sort order for the input columns that's defined in the decision table. | A decision table column is configured like this:<br>• Column: Applicant Code<br>• Data Type: Text<br>• Operator: Matches<br>• Sort Order: Ascending with Blank Values First<br><br>The Applicant Code column has ABCDEXIJ, TMPUHBK, XYZABC, and NNABCOP as the row values.<br><br>`ABC` is passed as the input string for lookup. The table returns the output values for the matching input cells in ascending order starting with blank values, followed by ABCDEXIJ, NNABCOP, and XYZABC.<br><br>Here, ABCDEXIJ, XYZABC, and NNABCOP are considered as matching values because they contain the string ABC. |

| Data Type | Description | EXAMPLE |
|---|---|---|
| Text | Matches your input string to an entire value or a part of it in decision table cells and returns the output values of the rows that the cells belong to. The output is sorted by the sort order for the input columns that's defined in the decision table. | A decision table column is configured like this:<br>• Column: Job Title<br>• Data Type: Text<br>• Operator: Matches<br>• Sort Order: Ascending with Blank Values First<br>The Job Title column includes the following row values: Senior Regional General Manager, General Manager, Regional General Manager, and Assistant General Manager.<br>When `Regional General Manager` is passed as the input string for lookup with the Matches operator, the table returns the output values for the matching input cells in ascending order, starting with blank values.<br>The matching values are Regional General Manager and Senior Regional General Manager, as they contain the string "Regional General Manager." |
| AutoNumber | Matches your input number to an entire value or a part of a value in decision table cells and returns the output values of the rows that the cells belong to. The output is sorted by the sort order for the input columns that's defined in the decision table. | A decision table column is configured like this:<br>• Column: Serial Number<br>• Data Type: AutoNumber<br>• Operator: Matches<br>• Sort Order: Ascending with Blank Values First<br>The Serial Number column has 10, 100, 101, 102, 103, 111, and |

| Data Type | Description | EXAMPLE |
| --- | --- | --- |
| | | 120 as the row values.<br><br>`10` is passed as the input value for lookup. The table returns the output values for the matching input rows in ascending order starting with blank values, followed by 10, 100, 101, 102, and 103. |
| Email | Matches your input string to an entire email or a part of it in decision table cells and returns the output values for the rows that the cells belong to. The output is sorted by the sort order for the input columns that's defined in the decision table. | A decision table column is configured like this:<br><br>• Column: Contact Email<br>• Data Type: Email<br>• Operator: Matches<br>• Sort Order: Descending with Blank Values Last<br><br>The Contact Email column has rose@salesforce.com, lily@salesforce.com, jasmine@salesforce.com, salesforce.support@salesforce.com, lily@example.com, and rosa@example.com as the row values.<br><br>The string, `sales`, is passed as the input value for lookup. The table returns the output values for the matching input rows in the order salesforce.support@salesforce.com, rose@salesforce.com, lily@salesforce.com, jasmine@salesforce.com, followed by output values for blank cells in the Contact Email column. |
| URL | Matches your input string to an entire URL or a part of it in | A decision table column is |

| Data Type | Description | EXAMPLE |
| --- | --- | --- |
| | decision table cells, and returns the output values for the rows that the cells belong to. The output is sorted by the sort order for the input columns that's defined in the decision table. | configured like this:<br>• Column: Company Website<br>• Data Type: URL<br>• Operator: Matches<br>• Sort Order: Ascending with Blank Values Last<br><br>The Company Website column has these row values:<br>• www.salesforce.com<br>• www.salesforce.org<br>• www.example.com/home<br>• www.example.org<br><br>The string, `org`, is passed as the input value for lookup. The table returns the output values for the matching input rows in this order starting with blank values: www.example.org/home and www.salesforce.org. |
| Picklist | Matches your input value to an entire value or a part of the value in decision table cells, and returns the output values for the rows that the cells belong to.<br><br>Output sort order doesn't apply to columns of the Picklist data type. | A decision table column is configured like this:<br>• Column: Status<br>• Data Type: Picklist<br>• Operator: Matches<br><br>The Status column has Draft, Active, Draft, and Not Applicable as the row values.<br><br>The string, `Dra`, is passed as the input value for lookup. The table returns the output values for the rows whose Status column cells contain the term Dra and the rows where the cells in the Status column are blank. |

# Does Not Match Operator in Decision Tables

Use the Does Not Match operator to retrieve an output if the query value does not match the decision table row value and is also not a subset of the decision table row value. The Does Not Match operator is case-sensitive and applies only to input fields in the decision table.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Here are some examples of how the Does Not Match operator works for the data types that support the operator.

| Data Type | Description | EXAMPLE |
|---|---|---|
| Text | Retrieves an output if the query text isn't an exact match or subset of the input cell text. The output is sorted by the sort order for the input columns that's defined in the decision table. | A decision table column is configured like this:<br>• Column: Department<br>• Data Type: Text<br>• Operator: Does Not Match<br>• Sort Order: Ascending with Blank Values First<br><br>The Department column has sales, presales, postsales, salessupport, support, and marketing as the row values.<br><br>`sales` is passed as the input string for lookup. The table returns the output values for the input cells in ascending order starting with blank values, marketing, and support. Here, the query value isn't an exact match or subset of the other input values. |
| Text | Retrieves an output if the query text isn't an exact match or subset of the input cell text. The output is sorted by the sort order | A decision table column is configured like this: |

| Data Type | Description | EXAMPLE |
|---|---|---|
| | for the input columns that's defined in the decision table. | • Column: Job Title<br>• Data Type: Text<br>• Operator: Does Not Match<br>• Sort Order: Ascending with Blank Values First<br><br>The Job Title column includes the following row values: Senior Regional General Manager, General Manager, Regional General Manager, and Assistant General Manager.<br><br>When `Regional General Manager` is passed as the input string for lookup with the Does Not Match operator, the table returns the output values for the non-matching input cells.<br><br>The non-matching values are General Manager and Assistant General Manager. |
| Picklist | Retrieves an output if the query value isn't an exact match or subset of the input cell value.<br><br>Output sort order doesn't apply to columns of the Picklist data type. | A decision table column is configured like this:<br><br>• Column: Status<br>• Data Type: Picklist<br>• Operator: Does Not Match<br><br>The Status column has draft, inactive, active, draft, and not applicable as the row values.<br><br>The Status column has inactive, active, draft, and not Applicable as the row values.<br><br>The string `active` is passed as the query value to look up. The table returns the output values draft and not applicable since they aren't an exact match or |

| Data Type | Description | EXAMPLE |
|---|---|---|
| | | subset of the query. |

## Contains Operator in Decision Tables

Use the Contains operator to retrieve an output if the decision table row value is an exact match or subset of the query value. The Contains operator is case-sensitive and applies only to input fields in the decision table.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Here are some examples of how the Contains operator works for the data types that support the operator.

| Data Type | Description | EXAMPLE |
|---|---|---|
| Text | Retrieves the output value if the input text is an exact match or subset of the query text. The output is sorted by the sort order for the input columns that's defined in the decision table. | A decision table column is configured like this:<br>• Column: Applicant Name<br>• Data Type: Text<br>• Operator: Contains<br>• Sort Order: Ascending with Blank Values First<br><br>The Applicant Code column has PQRSTV, PRQST, PQRS, NNABCOP, and PQRSTUVW as the row values.<br><br>`PQRSTUVW` is passed as the query string for lookup. The table returns the output values for the matching input cells in ascending order starting with blank values, followed by PQRS, PQRST, and PQRSTUVW |

| Data Type | Description | EXAMPLE |
|---|---|---|
| | | Here PQRS, PQRST, and PQRSTUVW are considered as matching values because they're the subset or exact match to the query string PQRSTUVW. |
| Text | Retrieves the output value if the input text is an exact match or subset of the query text. The output is sorted by the sort order for the input columns that's defined in the decision table. | A decision table column is configured like this:<br>• Column: Job Title<br>• Data Type: Text<br>• Operator: Contains<br>• Sort Order: Ascending with Blank Values First<br><br>The Job Title column includes the following row values: Senior Regional General Manager, General Manager, Regional General Manager, and Assistant General Manager.<br><br>When `Regional General Manager` is passed as the input for lookup with the Contains operator, the table returns the output values for the cells that contain an exact match or subset of the input string.<br><br>The values that contain an exact match or subset of "Regional General Manager" are General Manager and Regional General Manager. |
| Picklist | Retrieves the output value if the input value is an exact match or subset of the query value.<br><br>Output sort order doesn't apply to columns of the Picklist data type. | A decision table column is configured like this:<br>• Column: Status<br>• Data Type: Picklist<br>• Operator: Contains |

| Data Type | Description | EXAMPLE |
| --- | --- | --- |
| | | The Status column has Inactive, Active, Draft, and Not Applicable as the row values.<br><br>The string `Inactive` is passed as the query value to look up. The table returns the output values Active and Inactive since both the values are an exact match or subset of the query value. |

## How Grouping Works in Decision Tables

When you create a decision table, you can group the table's rows by input fields. Grouping splits each unique combination of values in the grouping field rows into their individual group. By categorizing the data logically, when you provide input values from the grouping fields for lookup, the lookup considers only the group associated with that value. Grouping helps streamline the lookup process by focusing only on the relevant information for the given input values.

REQUIRED EDITIONS

| |
| --- |
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Here are some considerations for grouping decision table rows.

- You can group a table's rows by up to two input fields that use the Equals operator.
- You can group a decision table's rows when the input condition uses the All, Any, or Custom logic.
  If you use the Any or Custom logic, grouping fields aren't part of the input condition logic. However, the grouping fields are appended to the logic by default.
  For example, a decision table has four input fields (3). The table's rows are grouped by the Name and Id fields (2). The table is configured with the Any condition logic, which uses the OR operator. Here, the grouping fields are excluded from the condition logic (1).

- When you group a table's rows by two input fields, you must pass lookup values for both the fields.
- You can group a decision table's rows when the number of input fields used for grouping is less than the total number of input fields in the table.
  For example, if your table has three input fields, you can use up to two fields for grouping. If your table has two input fields, you can use one of the fields for grouping. If your table has only one input field, you can't group the table's rows.

- If the field that you used for grouping has empty cells, the rows with the empty cells are consolidated into a separate group. When you provide an input value, the group that corresponds to the input value is looked up. When you don't provide an input value for the grouping field, only the group with the rows where the grouping field cells are empty is looked up.
  This lookup behavior is different from the lookup behavior for tables without row grouping, where rows with both the input value and any empty cells in the grouping field are looked up.

**Example** A decision table with these columns and rows is grouped by a single input field:

| INPUT | | | OUTPUT |
|---|---|---|---|
| **Serial Number** | **Product Name** | **Status** | **Product CODE** |
| 1 | P1 | Active | P101 |
| 2 | P2 | Inactive | P202 |
| 3 | P2 | Active | P200 |
| 4 | P1 | Inactive | P108 |
| 5 | P1 | Active | P105 |
| 8 | P1 | Active | P202 |
| 9 | P2 | Inactive | P101 |
| 11 | | Active | P303 |
| 12 | | Inactive | P300 |

The table is grouped by the Product Name field and the rows are split into these smaller groups.This group corresponds to the value P1 for the Product Name field.

| INPUT | | | OUTPUT |
|---|---|---|---|
| **Serial Number** | **Product Name** | **Status** | **Product CODE** |
| 1 | P1 | Active | P101 |
| 4 | P1 | Inactive | P108 |
| 5 | P1 | Active | P101 |
| 8 | P1 | Active | P101 |

This group corresponds to the value P2 for the Product Name field.

| INPUT | | | OUTPUT |
|---|---|---|---|
| **Serial Number** | **Product Name** | **Status** | **Product CODE** |
| 2 | P2 | Active | P202 |

| INPUT | | | OUTPUT |
|---|---|---|---|
| **Serial Number** | **Product Name** | **Status** | **Product CODE** |
| 3 | P2 | Inactive | P200 |
| 9 | P2 | Inactive | P101 |

This group is specific to the cells in the Product Name field being empty.

| INPUT | | | OUTPUT |
|---|---|---|---|
| **Serial Number** | **Product Name** | **Status** | **Product CODE** |
| 11 | | Active | P303 |
| 12 | | Inactive | P300 |

When P1 is passed as the input value to the table, the rows in the table that's grouped by Product Name field value P1 are looked up.

**Example** A decision table with these columns and rows is grouped by two input fields:

| INPUT | | | OUTPUT |
|---|---|---|---|
| **Serial Number** | **Product** | **Fabric** | **Product CODE** |
| 1 | Shirt | Cotton | P101 |
| 2 | Dress | Silk | P202 |
| 3 | Dress | Cotton | P200 |
| 4 | Shirt | Linen | P108 |
| 5 | Shirt | Cotton | P105 |
| 8 | Shirt | Cotton | P202 |
| 9 | Dress | Silk | P101 |
| 11 | Jacket | | P303 |
| 12 | Jacket | | P300 |

The table is grouped by the Product and Fabric fields. The rows are split into these smaller groups based on these unique combinations of values in the Product and Fabric fields:

- Shirt, Cotton
- Shirt, Linen
- Dress, Silk
- Dress, Cotton

- Jacket, <no value>

This group corresponds to the values Shirt and Cotton for the Product and Fabric fields.

| INPUT | | | OUTPUT |
|---|---|---|---|
| **Serial Number** | **Product** | **Fabric** | **Product CODE** |
| 1 | Shirt | Cotton | P101 |
| 5 | Shirt | Cotton | P105 |
| 8 | Shirt | Cotton | P202 |

This group corresponds to the values Shirt and Linen for the Product and Fabric fields respectively.

| INPUT | | | OUTPUT |
|---|---|---|---|
| **Serial Number** | **Product** | **Fabric** | **Product CODE** |
| 4 | Shirt | Linen | P108 |

This group corresponds to the values Dress and Silk for the Product and Fabric fields respectively.

| INPUT | | | OUTPUT |
|---|---|---|---|
| **Serial Number** | **Product** | **Fabric** | **Product CODE** |
| 2 | Dress | Silk | P202 |
| 9 | Dress | Silk | P101 |

This group corresponds to the values Dress and Cotton for the Product and Fabric fields respectively.

| INPUT | | | OUTPUT |
|---|---|---|---|
| **Serial Number** | **Product** | **Fabric** | **Product CODE** |
| 3 | Dress | Cotton | P200 |

This group corresponds to the value Jacket and empty cells for the Product and Fabric fields respectively.

| INPUT | | | OUTPUT |
|---|---|---|---|
| **Serial Number** | **Product** | **Fabric** | **Product CODE** |
| 11 | Jacket | | P303 |
| 12 | Jacket | | P300 |

When the values Shirt and Cotton are provided as input for the lookup table, only the group where the Product and Fabric field values are Shirt and Cotton is looked up.

# How Source Filters Work in Decision Tables

Narrow down the source object rows by using source filters when the source object has many rows. Use source filters to consider only relevant records from a large pool of records for decision-making.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Source Filter Data Types and Operators

| Supported Data Type | Operators |
|---|---|
| Text<br>Static enum<br>Dynamic enum | • Equals<br>• Not Equals<br>• Contains<br>• Is Not Null<br>• Is Null |
| Number<br>Double<br>Percent<br>Currency<br>Integer<br>Long<br>DateOnly<br>DateTime | • Equals<br>• Not Equals<br>• Less Or Equal<br>• Less Than<br>• Greater Or Equal<br>• Greater Than<br>• Is Not Null<br>• Is Null |
| Boolean<br>ID | • Equals<br>• Not Equals<br>• Is Not Null<br>• Is Null |

**Example** Consider a decision table with these columns and rows:

| INPUT | | | OUTPUT |
|---|---|---|---|
| **Serial Number** | **Product Name** | **Status** | **Product CODE** |
| 1 | P1 | Active | P101 |
| 2 | P2 | Inactive | P202 |
| 3 | P2 | Active | P200 |
| 4 | P1 | Inactive | P108 |
| 5 | P1 | Active | P105 |
| 8 | P1 | Active | P202 |
| 9 | P2 | Inactive | P101 |
| 11 | | Active | P303 |
| 12 | | Inactive | P300 |

To consider only the rows that have Product Name as P1 and Status as Active, then create the source filter this way.

- Product Name Equals P1
- Status Equals Active

The results for these source filters are rows with Serial Number 1, 5, and 8. The decision table performs the lookup during actual run time for only these records from the source objects.

## Input Field Conditions in Decision Tables

When you create a decision table, select a source object for the decision table. Next, select the fields that you want to use as input and output. Finally, specify the condition logic for the input fields so that the decision table can evaluate input values or records from the object against the logic that you specify.

REQUIRED EDITIONS

Available in: Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled

**Note** If you use decision tables as part of Loyalty Management or Rebate Management, see Decision Table.

To determine when a decision table provides outcomes, select one of these input field conditions.

| Condition | How it Works | Business Rules (Example) | Evaluated Values (Example) | Output (Example) |
|---|---|---|---|---|
| All conditions are met (AND) | The decision table evaluates the business rules against the values provided. If the evaluated values match the corresponding input fields, the decision table provides the output field values of the matched business rules as the outcomes. | Three business rules contain three input fields and one output field.<br><br>Rule 1:<br>• Input 1: Order Quantity < 50<br>• Input 2: Product Name = Running Shoes<br>• Input 3: Product Category = Shoes<br>• Output: Discount = 5%<br><br>Rule 2:<br>• Input 1: Order Quantity >=50<br>• Input 2: Product Name = Running Shoes<br>• Input 3: Product Category = Shoes<br>• Output: Discount = 10%<br><br>Rule 3:<br>• Input 1: Order Quantity >=40<br>• Input 2: Product Name = Athletic Shorts<br>• Input 3: Product Category = Accessories<br>• Output: | The decision table provides outcomes for each of the three rules.<br><br>Order 1:<br>• Order Quantity: 60<br>• Product Name: Running Shoes<br>• Product Category: Shoes<br><br>Order 2:<br>• Order Quantity: 70<br>• Product Name: Football Shoes<br>• Product Category: Shoes<br><br>Order 3:<br>• Order Quantity: 20<br>• Product Name: Athletic Shorts<br>• Product Category: Accessories | • Order 1: 10%<br>• Order 2: No output<br>• Order 3: No output |
| Any condition is met (OR) | The decision table evaluates the business rules against the values provided. If any one of the evaluated values matches the corresponding input field, the decision table provides the output field values of the matched business rules as the outcomes. | | | • Order 1: 5%, 10%, 7%<br>• Order 2: 5%, 10%, 7%<br>• Order 3: 5%, 7% |
| Custom logic | The decision table evaluates the business rules against the values provided. Based on the custom logic, if the values match, the | | | Logic: (1 AND 2) OR 3<br>• Order 1: 10%,<br>• Order 2: No output<br>• Order 3: 7% |

| Condition | How it Works | Business Rules (Example) | Evaluated Values (Example) | Output (Example) |
|---|---|---|---|---|
| | decision table provides the output field values of the matched business rules as the outcomes.<br><br>You can use a field's sequence number only once in a custom logic. For example, the custom logic `(1 AND 2) OR (1 AND 3)` is invalid. Instead, you can define the logic as `1 AND (2 OR 3)`. | Discount = 7% | | |

## How Result Filters Work

When you create a decision table, you can specify how you want the table's results to be filtered. The filter you choose determines whether you get a single matching result or multiple. If there are multiple results, filtering also determines the order in which the results appear.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

When you create a decision table, choose one of these filtering options based on your requirements and on how you want to use the results in the subsequent steps of a workflow or a process.

| Filter Result By | Result Type | Description |
|---|---|---|
| First Match | Single Outcome | If your input yields multiple results from the lookup, this option shows the first record in the decision table's source object. |

| Filter Result By | Result Type | Description |
|---|---|---|
| Output Order | Multiple Outcomes | If your input yields multiple results, this option shows all the matching results based on the order of the matching records in the decision table's source object. |
| Any Value | Single Outcome | When your table has multiple rows with an identical combination of input and output values, this option returns the output values from only one of the rows.<br><br>If the combination of your lookup values has only one matching output, this option returns the single matching output value.<br><br>If the combination of your lookup values has multiple, different matching outputs, this option returns an empty response. |

**Example** A decision table is used to look up the daily fees to rent a car based on the car type and the city. Here's how the decision table is structured.

| Serial Number | Car Type (Required Input) | City (Optional Input) | Daily Fees (Output) |
|---|---|---|---|
| 1 | Hatchback | New York | 50 |
| 2 | Hatchback | San Francisco | 48 |
| 3 | Hatchback | Chicago | 45 |
| 4 | Sedan | New York | 60 |
| 5 | Sedan | Chicago | 55 |
| 6 | Sedan | San Francisco | 58 |
| 7 | Sedan | New York | 60 |
| 8 | SUV | New York | 80 |
| 9 | SUV | Chicago | 75 |
| 10 | SUV | San Francisco | 78 |

When you call the decision table from a flow, the flow returns the results based on the filter you specified when you created the table. Let's look at a few scenarios where different lookup values yield varying results based on the specified filter.

| Results Filtered By | Flow Configuration | Lookup Values | Flow Output | Explanation |
|---|---|---|---|---|
| First Match | Include the Car Type input column for lookup | Sedan | 60 | Here, the flow looks for the value Sedan in the decision table. Rows 4, 5, 6 have the value Sedan in the Car Type input column. But the flow returns 60, which corresponds to Sedan and New York because it's the first matching value in the table. |
| Output Order | Include the Car Type input column for lookup | SUV | 80, 75, 78 | Here, the flow looks for the value SUV in the decision table. Rows 8, 9, 10 have the value SUV in the Car Type input column. The flow returns the matching results for the three rows based on their order in the source object. |
| Any Value | Include the Car Type and City input columns for lookup | Sedan, New York | 60 | Here, the flow looks for the values Sedan and New York in the decision table. Rows 4 and 7 have the values Sedan in the Car Type and City input columns respectively. Because the input and output values for the two columns are identical, the output returns the single value 60. |

## How Null Values Work in Decision Tables

Use the Consider Null Values for Lookup option to handle the behavior of how rows with null values are treated during lookup. When you enable this option, rows with null values in the decision table are matched only when the input explicitly includes null, which ensures precise matches. When you disable this option, and any of the rows have null values, then the column values that pertain to the null value are also considered as null.

REQUIRED EDITIONS

Available in: Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled

**Note** Decision tables consider null values for the input conditions that are mandatory and have the operator set as Equal.

**Example** Consider the scenario where a decision table has input and output columns per this table:

| **INPUT** | | | **OUTPUT** |
|---|---|---|---|
| **Product** | **Payment type** | **Price** | **Discount** |
| A | one-time | 1000 | 5 |
| B | null | 500 | 10 |
| B | one-time | 500 | 15 |

When **Consider Null Values for Lookup** is enabled, the output matches a null input to the explicit null value in the payment type column. When **Consider Null Values for Lookup** is disabled, the output matches a null input to all the values in the payment type column. Here is a table that compares the outputs for when **Consider Null Values for Lookup** is enabled and disabled.

| **Product** | **Payment Type** | **Price** | **Output with Consider Null Values for Lookup Enabled** | **Output with Consider Null Values for Lookup Disabled** |
|---|---|---|---|---|
| A | one-time | 1000 | 5 (first row is an exact match) | 5 (first row is an exact match) |
| B | null | 500 | 10 (second row matches explicit null in input) | 10 and 15 (second and third rows match the product and price columns, and null matches all values in the payment type column) |
| A | null | 1000 | No output values (first row's 'one-time' doesn't match explicit null) | 5 (first row matches the product and price and null matches all payment type |

| Product | Payment Type | Price | Output with Consider Null Values for Lookup Enabled | Output with Consider Null Values for Lookup Disabled |
|---|---|---|---|---|
| | | | | values) |
| B | one-time | 500 | 15 (third row is an exact match) | 15 (third row is an exact match) |

## Considerations for Creating Decision Tables

Before you plan and create decision tables in your Salesforce org, review these considerations.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Note** If you use decision tables as part of Loyalty Management or Rebate Management, see Decision Table.

- You can use only the objects that have the CreatedDate and IsDeleted fields as source objects to create a decision table.
- You can group a table's rows by up to two input fields that use the Equals operator.
- You can use a field's sequence number only once in a custom logic.
  For example, the custom logic `(1 AND 2) OR (1 AND 3)` is invalid. Instead, define the logic as `1 AND (2 OR 3).`
- By default, a decision table can have 30 input fields and five output fields.
  Configure input fields so that during lookup, a maximum of 10 input fields are considered. This field aggregation can be achieved by using the AND operator in the table's input condition logic to join up to 21 required columns that use the Equals operator.

**Example** Configure a Decision Table Logic to Bring Input Columns Down to 10A table has seven required input fields that use the Equals operator.Here's how the table's input condition logic is configured.

```
(1 AND 2 AND 3) OR 4 OR (5 AND 6 AND 7)
```

When the table is activated, fields 1, 2, and 3 are joined and considered as one field. Field 4 is considered as an independent field. Fields 5, 6, and 7 are joined and considered as one field.During lookup, even though the active table is considered to have three input fields, you can pass input

values for all seven input fields. The output is based on the values passed for the seven input fields.

# Decision Table Asynchronous Events

Trigger actions when decision table operations like Activation, Refresh, or CSV uploads complete asynchronously.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Decision table actions such as activation, refresh, CSV upload, and CSV download trigger events once they are completed in **DecisionTableActionEvent**. You can trigger actions either by using a platform-event triggered flow or by Apex triggers.

Here are the status values for each decision table action event.

| Action | Status Values |
|---|---|
| Activation | Completed, Failed |
| Refresh | Completed, Failed |
| CSV Upload | Completed, CompletedWithErrors, Failed |
| CSV Download | Completed, Failed |

**Consume Decision Table Asynchronous Events with Flows**
Trigger actions by using flows when decision table operations complete asynchronously.

**Consume Decision Tables Asynchronous Events with Apex Code**
Trigger actions by using Apex code when decision table operations complete asynchronously.

## Consume Decision Table Asynchronous Events with Flows

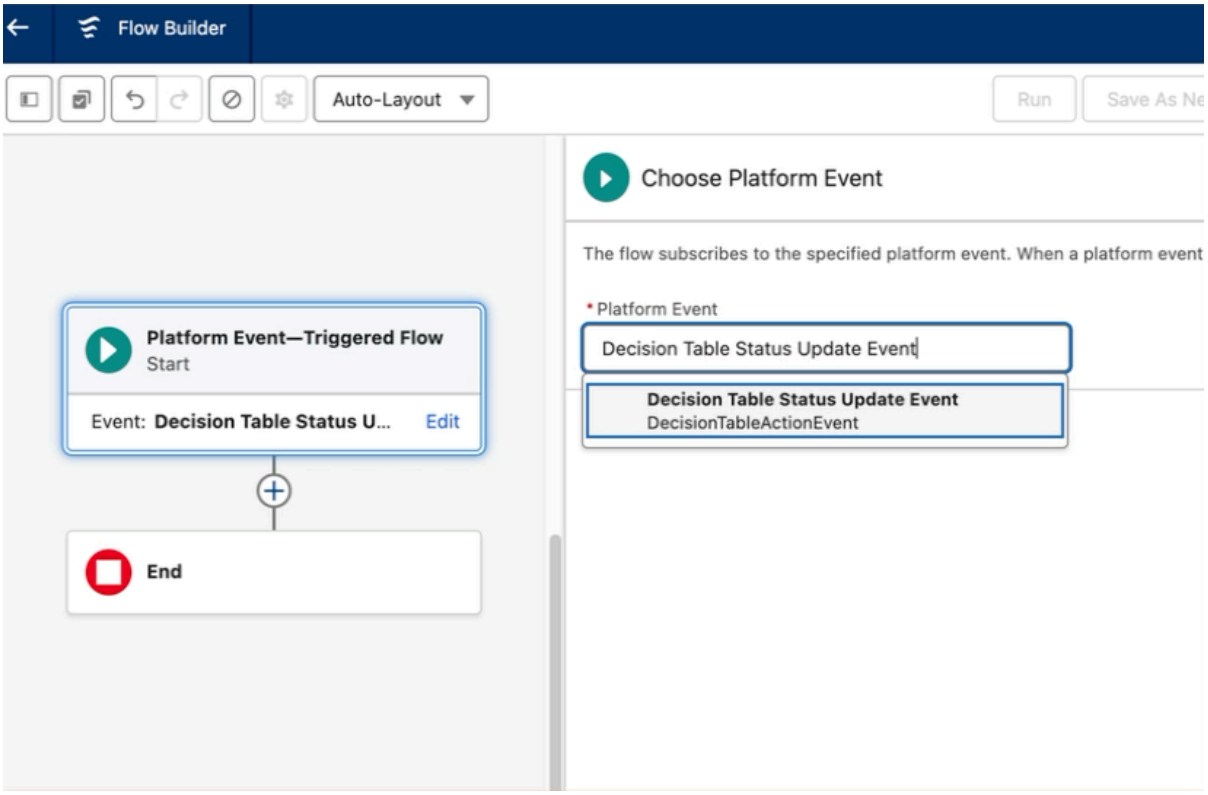
Trigger actions by using flows when decision table operations complete asynchronously.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, edit, and activate a decision table: | Rule Engine Designer |
| To run decision tables: | Rule Engine Designer<br>OR<br>Rule Engine Runtime |

1. In Setup, find and select **Flows**.
2. Click **New Flow**.
3. Select **Triggered Automations**, then select **Platform Event-Triggered Flow**.
4. In the Flow Builder, in the Platform Event-Triggered Flow element, click **Choose Platform Event**.
5. In Platform Event, find and select **DecisionTableActionEvent**.




6. Configure the flow.
   For example, you can update the Decision Table description with the event details.
7. **Save** and **Activate** your flow.

When a DecisionTableActionEvent is completed asynchronously, it triggers your flow.

## Consume Decision Tables Asynchronous Events with Apex Code

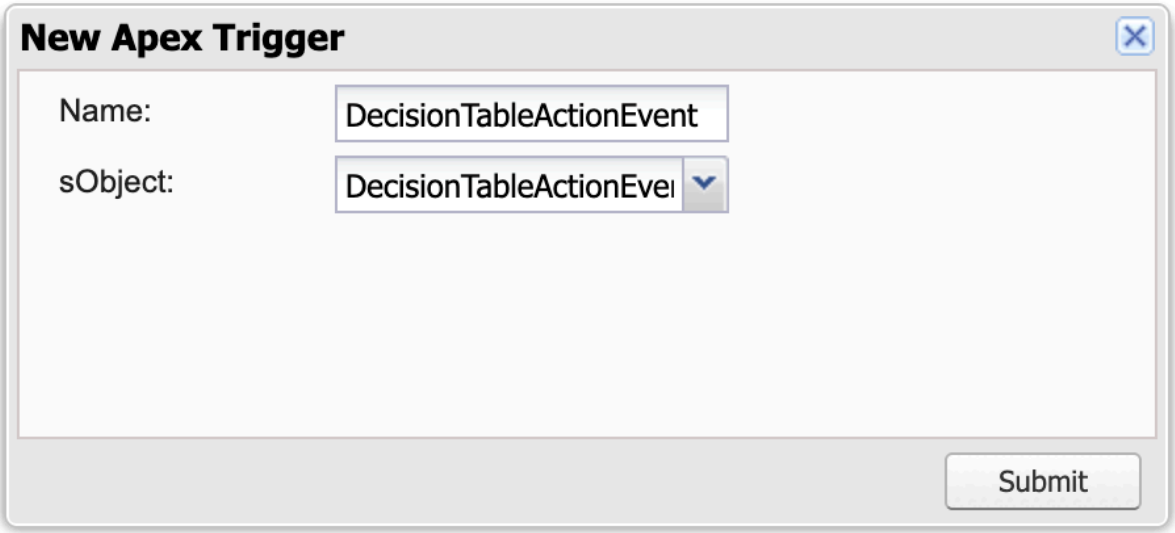
Trigger actions by using Apex code when decision table operations complete asynchronously.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, edit, and activate a decision table: | Rule Engine Designer |
| To run decision tables: | Rule Engine Designer<br>OR<br>Rule Engine Runtime |

1. Open the Developer Console.
2. Click **Create New**
3. Click **Apex Trigger**.
4. Select **DecisionTableActionEvent**.




5. Add the provided trigger code:

```
trigger deci on DecisionTableActionEvent (after insert) {
for (DecisionTableActionEvent event : Trigger.new) {
// Log the event details to the debug log
System.debug('*** APEX DecisionTableActionEvent received! ' +
'Event ID: ' + event.DecisionTableApiName +
', Event Message: ' + event);

// Add more debug statements for specific fields based on your requirements
// Example: System.debug('New Status: ' + event.New_Status__c);
```

```
}
}
```

6. Save the trigger.

To trigger the event, activate your decision table.

To make sure that the event is consumed, in Debug Logs:

- Set **Traced Entity Type** to **Automated Process**:
- Set Apex logging to FINER or FINEST.
- Set the **Expiration Date** to a future date.

See Also

Platform Events Developer Guide: Add a Trace Flag Entry for the Default Automated Process User
Enhance Salesforce with Code: Set Up Apex Class and Trigger Trace Flags
Force.com IDE Developer Guide: Apex Code Settings: Log Category and Log Level

# Best Practices for Optimum Performance and Scalability of Decision Tables

When you use decision tables for your decision-making process, it's important that you optimize them for scalability and performance. Scalability refers to the ability of decision tables to handle large datasets without compromising performance. Performance makes sure that decision tables operate at their highest efficiency with minimal latency and reduced processing time.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Optimize your decision tables for scalability to accommodate larger datasets such as large volumes of rows or columns while maintaining stability. For quicker and efficient querying and evaluation of your decision tables, optimize them for performance.

**Important** Before you follow the best practices for decision tables, review the default limits for Business Rules Engine components and their usage in Business Rules Engine Default Limits. To raise any default limits, contact Salesforce Customer Support.

Before you create decision tables, understand these best practices for the different types of decision tables.

**Best Practices for Standard Decision Tables**

Follow the best practices for optimizing your standard decision tables for performance and scalability.

**Best Practices for Advanced Decision Tables**
Follow the best practices for optimizing your advanced decision tables for performance and scalability.

## Best Practices for Standard Decision Tables

Follow the best practices for optimizing your standard decision tables for performance and scalability.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Best Practices for Optimal Performance

- If you have a decision table with a large volume of rows, consider partitioning the decision table for better performance. Make sure that each partition has a unique partition key and the partition size doesn't exceed 25,000 rows.
- When you create a standard decision table, if you have mandatory conditions that use the Equals operator, and the condition type is set to Custom, make sure that these conditions are grouped together in the Effective Logic field. Group these conditions to minimize the lookup complexity and improve the rule evaluation efficiency.
  Consider this example where five conditions are created and the condition type is set to Custom when you create a standard decision table.

  | condition | source object field | operator | cOLUMN NAME | Optional? |
  |---|---|---|---|---|
  | 1 | BillingCity | Equals | BillingCity | No |
  | 2 | BillingState | Equals | BillingState | No |
  | 3 | Industry | Equals | Industry | No |
  | 4 | Type | Equals | Type | No |
  | 5 | OwnerId | Equals | OwnerId | No |

  For this scenario, the correct effective logic is `(1 AND 2 AND 3) AND (4 OR 5)`.
- Condition sequence affects the lookup performance. Place the condition that filters out the most rows early in the condition logic to minimize unnecessary evaluations. Identify the columns that filter out the most records early. Prioritize columns that have unique or highly varying values over those with repetitive values.
  For example, when you set BillingCity as the column in the first condition, more table lookups are required because a significant number of rows are not reduced early. However, when you place Industry as the column in the first condition, irrelevant records are filtered out early making

subsequent conditions more efficient.

- When you define conditions with input fields such as City or Property that have multiple eligible values, use separate rows for each value with the Equals operator instead of combining values into a single row with a partial match.
  For example, if the input field is City and the eligible values are New York, London, and Mumbai, create three separate conditions, each using the Equals operator as shown in this table:

| condition | source object field | operator | cOLUMN NAME | Optional? |
|---|---|---|---|---|
| 1 | City | Equals | NewYork | No |
| 2 | City | Equals | London | No |
| 3 | City | Equals | Mumbai | No |

- Ensure that you provide an input to a Required field that uses the Equals operator to avoid errors.

Best Practices for Optimal Scalability

If you have a decision table with a large volume of rows, consider partitioning the decision table. Make sure that each partition has a unique partition key and the partition size doesn't exceed 25,000 rows.

## Best Practices for Advanced Decision Tables

Follow the best practices for optimizing your advanced decision tables for performance and scalability.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Best Practices for Optimal Performance

- When you create an advanced decision table, if you have mandatory conditions that use the Equals operator, and the condition type is set to Custom, make sure that these conditions are grouped together in the Effective Logic field. Group these conditions to minimize the lookup complexity and improve the rule evaluation efficiency.
  For example, consider this advanced decision table where five conditions are created and the condition type is set to Custom.

| condition | source object field | operator | cOLUMN NAME | Optional? |
|---|---|---|---|---|
| 1 | BillingCity | Equals | BillingCity | No |

| condition | source object field | operator | cOLUMN NAME | Optional? |
|---|---|---|---|---|
| 2 | BillingState | Equals | BillingState | No |
| 3 | Industry | Equals | Industry | No |
| 4 | Type | Equals | Type | No |
| 5 | OwnerId | Equals | OwnerId | No |

For this scenario, the correct effective logic is (1 AND 2 AND 3) AND (4 OR 5).

- For smaller data changes to the object or the custom metadata associated with a decision table, use the Decision Table Refresh Action invocable action instead of doing a refresh from the decision table details page. See Refresh a Decision Table.

Best Practice for Optimal Scale

- To design a decision table with a large number of columns, group the columns by using the AND operator in the effective logic when you create the decision table.
- Use source filters when you create a decision table to pre-filter the dataset before the decision table is created. Source filters reduce the number of records that must be evaluated in the decision logic.

## How to Choose Between Standard and Advanced Decision Table Types

Optimize your rule management based on the size and complexity of data by selecting the right decision table type. This categorization improves system performance and storage efficiency, ensuring fast and scalable decision-making processes. Customize your tables to improve flexibility and resource allocation.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, edit, and activate a decision table: | Rule Engine Designer |
| To run decision tables: | Rule Engine Designer<br>OR<br>Rule Engine Runtime |

## Complex Rule in Decision Tables

A rule is complex if it uses only OR conditions. This is because the decision table looks through all the fields in a database to find the match, increasing latency and affecting performance. AND conditions, in contrast, specify which fields to check, making the lookup faster.

## Differences Between Standard and Advanced Tables

These are the key differences between standard and advanced decision table types.

| **Support** | **Standard** | **Advanced** |
|---|---|---|
| Number of Rows | A maximum of 100,000 rows can be processed. It can be extended if required. | A maximum of 2 million rows can be processed. It can be extended if required. |
| Number of Active Decision Tables | 20 decision tables can be simultaneously active in an org. It can be extended if required. | 40 decision tables can be simultaneously active in an org. It can be extended if required. |
| Rule Complexity | Complex rules that use only OR conditions.<br><br>Simple rules that use only AND conditions or a combination of AND and OR conditions. | Simple rules that use only AND conditions or a combination of AND and OR conditions. The rule must meet these criteria:<br><br>• At least one AND condition,<br>• At least one Required input condition column,<br>• At least one input condition column that uses the Equals operator.<br><br>Example: (1 OR 2) AND (3 OR 4) this is not supported in Advanced decision tables as there is no Required input condition.<br><br>Example: 1 AND (2 OR (3 OR 4)) is supported in Advanced decision tables where at least one input column is Required and uses the Equals operator. |
| Number of AND and OR Conditions | The maximum number of logical conditions, including both AND | The maximum number of OR conditions is 10 and the |

| Support | Standard | Advanced |
|---|---|---|
| | and OR, is 30. | maximum number of AND conditions is 31. |
| CSV File | You can't upload data as CSV files. | You can upload data as CSV files. |
| Null Values | You get the option to include Null Values when you create a decision table. | Null Values are considered by default in the decision table. |
| Standard Objects | Not Available | Standard objects like Account, Lead, Contact, Opportunity, and Case are available. |

## When to Use Standard and Advanced Decision Tables

Here are some scenarios to help you choose between standard and advanced decision tables.

| Scenario | Description | Decision Table Type |
|---|---|---|
| Less volume and high complexity | If you have 100,000 or less rows with complex rules. | Standard |
| Less volume and less complexity | If you have 100,000 or less rows with simple rules. | Standard, Advanced |
| Large Volume and less complexity | If you have more than 100,000 rows with simple rules. | Advanced |
| Large Volume and high complexity | If you have more than 100,000 rows and complex rules. | • Advanced - Simplify your rules that use at least one 'AND' condition and then use advanced decision tables.<br>• Standard - If you can't simplify the rules, contact Salesforce admin to extend the row limits for standard decision tables. |

These additional features are available only in advanced decision table type. If you want to use any of these features, you must use the advanced decision table type.

| Features | Description |
|---|---|
| CSV Data Upload | If you want to upload data through a CSV file. |

| Features | Description |
|---|---|
| Standard Objects | If you want to use standard objects like Account, Lead, Contact, Opportunity, and Case |

When you need complex rules and want to access features that are available only in advanced decision tables, simplify your rules that use at least one AND condition. Then, use the advanced decision tables.

**Example** Standard Decision Table A bank wants to offer a special loan to customers who meet at least one of these criteria:

- Condition 1: The customer has a credit score greater than 700.
- Condition 2: The customer has been an active user of the bank for more than 5 years till 2024.
- Condition 3: The customer has a monthly income greater than $5,000.
- Condition 4: The customer holds investments worth more than $100,000 in the bank.

The decision to offer the loan is based on OR conditions across these criteria. The decision table condition logic for this scenario is 1 OR 2 OR 3 OR 4.To determine loan eligibility, review the sample customer details. Each criteria is captured as an input condition field and the loan eligibility output is captured as a result field in the decision table. Customers are eligible if any one of the conditions is met.

| Input | | | | | Output |
|---|---|---|---|---|---|
| **Customer ID** | **Credit Score (Condition 1)** | **Joining Date (Condition 2)** | **Monthly Income (Condition 3)** | **Investments (Condition 4)** | **Loan Eligibility** |
| C001 | 720 | 11.11.2023 | $4,800 | $90,000 | Yes |
| C002 | 680 | 03.03.2015 | $5,200 | $80,000 | Yes |
| C003 | 680 | 07.10.2021 | $4,800 | $90,000 | No |
| C004 | 690 | 01.05.2022 | $4,500 | $150,000 | Yes |

As the rule in this scenario uses only OR conditions, it's considered complex, and a standard decision table type must be used.

**Example** Advanced Decision TableA utility company wants to reward residential customers who cut energy use. They are offering financial incentives to customers who use less energy, use energy-efficient technology, or join energy-saving programs. A customer's eligibility for the energy-saving incentive is determined by these criteria:

- Condition 1: Energy Consumption Reduction: The customer reduces their energy consumption by at least 15% compared to the previous year.
- Condition 2: Energy-Efficient Technology: The customer has installed an energy-efficient heat pump.
- Condition 3: Participation in Utility Programs: The customer participates in at least one utility energy-saving program, such as a demand response program or time-of-use pricing.

- Condition 4: Annual Energy Usage: The customer uses less than 20,000 kWh annually.

The decision to award the incentive is based on a combination of AND and OR conditions. The decision table condition rule logic for this scenario is 1 AND (2 OR 3) AND 4. To get the energy-saving incentive, a customer must satisfy these conditions:

- 1 and 4
- Either 2 or 3

Review some of the sample customer details based on which the customer's eligibility is determined. Each criteria is captured as an input condition field and the incentive eligibility output is captured as a result field in the decision table.

| **Input** | | | | | **Output** |
|---|---|---|---|---|---|
| **Customer ID** | **Energy Reduction <15% (Condition 1)** | **Installed Heat Pump (Condition 2)** | **Participated in Utility Program (Condition 3)** | **Annual Energy Usage < 20,000 kWh (Condition 4)** | **Incentive Eligibility** |
| C001 | Yes | Yes | Yes | Yes | Yes |
| C002 | Yes | No | Yes | Yes | Yes |
| C003 | Yes | Yes | No | Yes | Yes |
| C004 | No | Yes | Yes | Yes | No |
| C005 | Yes | No | Yes | No | No |

As the rule in this scenario uses a combination of AND and OR conditions, it's considered simple, and an advanced decision table type must be used.

**Example** Advanced Decision TableA health insurance company wants to offer a wellness incentive to customers who meet specific health criteria. The eligibility for the wellness incentive is determined by these criteria:

- Condition 1: The customer has completed an annual health check-up.
- Condition 2: The customer has a BMI (Body Mass Index) within the healthy range (18.5 - 24.9).
- Condition 3: The customer participates in a fitness program.
- Condition 4: The customer has a non-smoker status.

The customer must have completed an annual health check-up, and must either have a BMI within the healthy range, or participate in a fitness program, or be a non-smoker. The decision to award the wellness incentive is based on a combination of AND and OR conditions. The decision table condition rule logic for this scenario is 1 AND (2 OR (3 OR 4)). Review some of the sample customer details based on which the customer's eligibility is determined. Each criterion is captured as an input condition field and the incentive eligibility output is captured as a result field in the decision table.

| Input | | | | | Output |
|---|---|---|---|---|---|
| **Customer ID** | **Annual Health Check-up (Condition 1)** | **BMI within Healthy Range (Condition 2)** | **Participates in Fitness Program (Condition 3)** | **Non-Smoker Status (Condition 4)** | **Wellness Incentive Eligibility** |
| C001 | Yes | Yes | No | Yes | Yes |
| C002 | Yes | No | Yes | No | Yes |
| C003 | Yes | No | No | Yes | Yes |
| C004 | Yes | Yes | Yes | No | Yes |
| C005 | No | Yes | Yes | Yes | No |

As the rule in this scenario uses a combination of AND and OR conditions, and at least one input column is required and uses the Equals operator, it's considered simple, and an advanced decision table type must be used.

## Create a Decision Table

Create decision tables to read business rules and use the matching results in expression sets or flows. When you create a decision table, you can set the volume and complexity required for the table to match your data and business needs. You can either use a combination of source objects and related objects or a CSV file to define the source data for your decision tables. You can also use ready-to-use, industry-based templates to quickly create a decision table.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for Industries and Revenue clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, edit, and activate a decision table: | Rule Engine Designer |
| To run decision tables: | Rule Engine Designer<br>OR<br>Rule Engine Runtime |

1. From the App Launcher( ), find and select **Business Rules Engine**.

2. Click the app navigation menu, and select **Lookup Tables**.
3. Click **New**.
4. Select **Decision Table**, and then click **Next**.
   The Basic Details page appears.
5. Provide these details:

| Field | Description |
| --- | --- |
| Name and API Name | Enter the name of the decision table. The API name is auto-populated from the decision table name that you enter. If necessary, you can change the API name. |
| Description | Enter a meaningful description for your decision table. For example, calculate loan eligibility. |
| Application Usage Type | Relate your decision table to an existing usage type that is based on your business application. Selecting a usage type categorizes the decision table to that usage type.<br><br>When you select a decision table in an expression set version's lookup table step, you can view only the tables that share the expression set's usage type and the tables without a usage type.<br><br>If your usage type has ready-to-use industry templates related to it, then these templates appear when you select the usage type. You can quickly create a decision table by using these templates. The templates are available in your Salesforce org only if your Industries solution has shipped it. |
| Decision Table Type | Select the type of decision table that you want to use based on whether you want to set your decision table for volume or complexity.<br><br>To create a decision table that supports a large volume of rows and lesser complexity, use an advanced decision table. You can include only optional columns in the logic as a combination of AND and OR condition.<br><br>In advanced decision tables, if the Equals |

| Field | Description |
| --- | --- |
| | operator has optional conditions, then the set of all optional conditions is always followed by AND to the remaining effective logic. For example, consider there are four conditions 1, 2, 3, and 4. Conditions 1 and 2 are mandatory, and conditions 3 and 4 are set as optional. The optional conditions also have the operator set as Equals. When you select the condition type as Custom, and include the optional conditions by using OR, the set of optional conditions are added to the effective logic with an AND.<br><br>1 AND 2 AND (3 OR 4).<br><br>To create a complex condition for this scenario, you can set 1 as mandatory and the remaining conditions as optional. You can then apply the complex logic on the remaining conditions. For example, 1 AND (2 AND (3 OR 4))<br><br>To create a decision table that supports complex conditions, use Standard. You can include any column in the logic by using the OR condition. |

**Note** Decision tables consider null values for the input conditions that are mandatory and have the operator set as Equal.

6. Select **Create Decision Table** and then click **Save & Next**.

   **Important** If you use a ready-to-use industry template based on your application usage type to create a decision table, then all the fields are pre-populated. You can modify only the name, API name, and description. To modify the other pre-populated fields of the template, you must save the decision table, and then modify it.

   The Condition & Results page appears.
7. To create a decision table by using source objects, provide these details from the Salesforce Object tab:
   a. From Source Object, select the object that has the data that you want to use in the decision table.
   b. From Related Objects, select the objects that are related to the source object.
      You can select up to four objects that are related to the source object.

      **Important** The related objects must be within three levels of nesting from the source object.

   c. If the source object has a large number of rows, select **Source Filter,** if necessary.
      You can use the source filter as a prelogic to narrow down the source object rows. Add filters to your

source field by selecting the required field, operator, and value. To add additional filters, select **Add Filter**. If you have multiple filter conditions, you can specify the logic for all filters by using AND or OR in the Filter Logic field.

**Important** You can create up to five source filters when you create a decision table.

**Important** You cannot use the IsDeleted, LastModifiedDate, SystemModStamp, and CreatedDate fields as conditions.

d. To create a condition, in the Conditions pane, select the required field from the Object Field list, and a comparison operator from the Operator list.
Operators specify how values in the column are evaluated. For example, when you use the Equals operator for an input column, you define the particular values that the column's cells must have to meet the table's lookup condition. You can also set a condition as optional.

**Note** You can edit the column name only for the fields from related objects.

**Warning** When an optional column in a decision table does not have a value, it is evaluated as TRUE in a condition.

e. To group the decision table's rows by input fields, select **Group Condition Settings**.
Grouping splits each unique combination of values in the grouping field rows into their individual group.

**Important** The Group Condition setting is available only if you select the Standard decision table type.

f. To create multiple conditions, click **Add Condition**.
You can select Source Object Fields that aren't compound fields

**Tip** The Number column indicates the sequence number of the fields. For example, let's consider that you have three input fields. To use custom logic that provides an output when fields 1 and 2 match or when field 3 matches, specify the condition as: (1 AND 2) OR 3.

g. To specify the condition logic for the input fields, select a predefined condition type from the Condition Type list.
The decision table evaluates input values or records from the object against the logic that you specify.

h. From the Results pane, select the fields that you want to use as the output columns.
Output columns define the specific data that's returned when a query matches the input criteria. When you input a query into the table, the decision table evaluates the input rows against the defined conditions. If there are matching rows, the values from the output columns for those rows are generated as the result.

To create multiple output columns, click **Add Result**.



8. Click **Save & Next**.
The Behaviors page appears.

9. To create a decision table by using a CSV file, use the **CSV** tab.
   Use a CSV file to create decision tables, to save the time and effort for manually creating column headers and adding row data. You must first configure the input columns and output columns when you create the decision table. You can upload the CSV file after you create the decision table. Later, you can bulk edit CSV-based decision tables by downloading them as CSV files, and overwriting the data in the decision table.

   **Note** You can use a CSV file as the source only for advanced decision tables. You can't add, delete, or edit columns of a decision table that uses a CSV file as the source. You can't sort data in decision tables that use CSV files as a source.

   a. From the Conditions pane, configure the input columns by specifying the column name, data type, and operator.
   b. To add more input columns, click **Add Condition**.
   c. To specify the condition logic for the input fields, select a predefined condition type from the Condition Type list.
      A decision table evaluates input values or records from the object against the logic that you specify.
   d. From the Results pane, configure the output columns by specifying the column name and data type.
   e. To add more output columns, click **Add Result**.
10. Click **Save & Next**.
    The Behaviors page appears.
11. From the Results pane, specify how you want the table's results to be filtered by using the Filter Result By list.
    The filter that you choose determines whether you get a single matching result or multiple results. For multiple results, the filters also determine the order that the results appear in.

    **Note** An expression set can read only one result even if a decision table provides multiple results. So, if you plan to use the decision table in an expression set, select **First Match** or **Any Value**, which are both specific to single values.

12. If needed, sort the decision table rows by a table field and select a sort order.
13. Click **Save & Next** .
    The Preview & Save page appears. You can preview various aspects of your decision table such as source of data, and optional fields.
14. To create your decision table, click **Finish**.

Activate the decision table to start using it in expression sets or flows. If you update the object or the custom metadata type that contains the rules, refresh the decision table to make sure that the outcomes are based on the latest rules.

**Important**

- If you use multiple Salesforce objects as the source to create a decision table, the Table tab isn't available on the decision table details page.
- If you make modifications, such as the deletion of a field, change of data type, and change of name, to the source object of the decision table, you can't use the decision table again.

**Add CSV Data to Decision Tables**
When you use CSV as the source data for your decision table, you can configure only the input and output columns for the mapping. You can upload the CSV data for your decision table only after you finish creating a decision table.

**Edit Decision Tables with CSV Files**
Download an existing decision table as a CSV file, make bulk edits directly in the file, and upload it back. You can either append new rows of data to an existing table or overwrite the existing values.

## Add CSV Data to Decision Tables

When you use CSV as the source data for your decision table, you can configure only the input and output columns for the mapping. You can upload the CSV data for your decision table only after you finish creating a decision table.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, edit, and activate a decision table: | Rule Engine Designer |
| To run decision tables: | Rule Engine Designer<br>OR<br>Rule Engine Runtime |

1. From the App Launcher( ), find and select **Business Rules Engine**.
2. Click the app navigation menu, and select **Lookup Tables**.
3. Select the decision table that you created by using CSV as the source.
4. From the Table tab, click **Upload CSV**.
   The Add Data to Decision Table dialog box appears.
5. Select the CSV file from your computer.

   **Note** The first row in your CSV file must be column names or headers.

6. To add the data from the CSV file to your decision table, click **Add Data**.
   To view the data rows in your decision table, refresh the page. You can verify the status of the CSV upload by checking the CSV Upload Status field in the Table tab.

# Edit Decision Tables with CSV Files

Download an existing decision table as a CSV file, make bulk edits directly in the file, and upload it back. You can either append new rows of data to an existing table or overwrite the existing values.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, edit, and activate a decision table: | Rule Engine Designer |
| To run decision tables: | Rule Engine Designer<br><br>OR<br><br>Rule Engine Runtime |

**Note** You can download only those decision tables that were created using a CSV file.

1. From the App Launcher, find and select **Business Rules Engine**.
2. Click the app navigation menu, and select **Lookup Tables**.
3. Select the decision table that you created by using a CSV file as the source.
4. On the Table tab, click **Add CSV Data**. The Add Data to Decision Table dialog box appears.
5. Select how you'd like to upload the CSV data.

| | |
|---|---|
| **To add rows of data to the existing table** | a. In Manage duplicate rows, select **Append**.<br>b. Click **Upload Files**, and select the CSV file from your system that contains the new rows.<br>c. Click **Add Data**. The data in the CSV file is added to the decision table as new rows.<br><br>The first row in your CSV file must match the column names of the decision table. |
| **To overwrite the rows of data in the existing table** | a. In Manage duplicate rows, select **Overwrite**.<br>b. Click **Prepare file for download**.<br><br>The existing data in the decision table is converted into a CSV file, which is hosted in |

| | |
|---|---|
| | Files.<br><br>It takes a few minutes to prepare the CSV file. CSV files can be downloaded at any time from Files.<br><br>The CSV File contains row IDs that correspond to data in the decision tables.<br>c. From the App Launcher, find and select **Files**.<br>d. Download the CSV file, then edit the data in the downloaded file and save your changes.<br><br>You can delete rows of data from the table by removing the values corresponding to the row IDs in the CSV file.<br><br>Make sure that the row IDs and the commas are retained in the edited CSV file.<br><br>e. Click **Upload Files**, and select the updated CSV file from your system.<br>f. Click **Add Data**. |

The data in the CSV file is added to or overwrites the data in the decision table.

To view the updated rows in your decision table, refresh the page. To verify the status of the CSV file being uploaded, see the CSV Upload Status on the Table tab.

## Activate a Decision Table

You can use decision tables in expression sets in Business Rules Engine, and in flows only when the decision tables are active.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, edit, and activate a decision table: | Rule Engine Designer |

| USER PERMISSIONS NEEDED | |
|---|---|
| To run decision tables: | Rule Engine Designer<br>OR<br>Rule Engine Runtime |

**Note** If you use decision tables as part of Loyalty Management or Rebate Management, see Decision Table.

**Note** You can activate decision tables created by only you in your org.

**Warning** During an activation or refresh action, any ongoing data modifications on the underlying sObject data may not be integrated. To ensure the decision table incorporates the complete and latest configuration, trigger a Refresh action after all updates to the source sObject data are completed.

1. Click , and then find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Lookup Tables**.
3. From the list of lookup tables, select the decision table that you want to activate.
4. Activate the decision table.

**Note** A decision table can take several minutes to get activated.

## Refresh a Decision Table

To meet evolving business requirements, you add, update, or remove decision table rows. If you update the object or the custom metadata type that contain the row data, refresh the decision table to ensure that the outcomes are based on the latest data.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To refresh any decision table from Setup: | System Administrator |

By default, you can perform a full refresh of a decision table, where all the rows are updated with the latest data from sObject records. A full refresh can take some time based on the size of the decision

table.

To save time, you can update the decision table data only with changes made to sObject data by invoking an incremental refresh action. You must perform a full refresh at least once on an active decision table before performing an incremental refresh. If an incremental refresh fails, a full refresh is automatically triggered. See Decision Table Refresh Action.

**Note** You can refresh decision tables created by only you in your org.

**Warning** During an activation or refresh action, any ongoing data modifications on the underlying sObject data may not be integrated. To ensure the decision table incorporates the complete and latest configuration, trigger a Refresh action after all updates to the source sObject data are completed.

1. From Setup, in the Quick Find box, enter `Decision Tables`, and then select **Decision Tables**.
2. From the decision tables list, select an active decision table that you want to refresh.
3. To perform a full refresh, click **Refresh**.

When the refresh is completed, the Refresh Status and Last Refreshed Date is updated.

## Tips to Manage Decision Tables

You can modify or delete decision tables used in expression set versions to keep your rules up to date with changing business requirements.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Here are some tips to help you manage decision tables.

| Scenario | Tips |
|---|---|
| You want to modify the columns of a decision table referenced in an expression set version. | • If the expression set version is active, deactivate it.<br>• Deactivate the decision table.<br>• Update the decision table columns to add, delete, or change the data types.<br>• Save and activate the decision table.<br>• Refresh the expression set version to synch the changes made in the decision table. |

| Scenario | Tips |
| --- | --- |
| | • Utilize the new decision table variables in subsequent steps and fix any errors that appear.<br>• Save and activate the expression set version. |
| You want to delete a decision table referenced in expression set versions. | • If a decision table is referenced in any expression set version, you can't delete it.<br>• First, deactivate, and delete the decision table reference in each expression set version that it's referenced in.<br>• Fix any errors that appear in the expression set version steps and save.<br>• After the decision table reference is removed from all expression set versions, deactivate the decision table.<br>• Now, delete the decision table. |
| A decision table is referenced in an expression set version. The usage type of the decision table is now modified. | The expression set version step throws an error. Ensure that the usage type of the decision table and expression set match. |
| The source object permissions or the source object fields used in the decision table are modified. | The decision table is no longer valid. Recreate the decision table with the updated source object configuration. |

## Considerations for Migrating Decision Tables

Before you plan to migrate a decision table from a source org to a target org, review the migration considerations.

REQUIRED EDITIONS

| |
| --- |
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Important** If you use custom fields in your decision table, you can't migrate the table from one Salesforce org to another using Package Manager in a single step.

**Note** Before deploying changes to the metadata of a decision table in an org where the decision

table already exists, you must deactivate the decision table in the target org. Ensure that you use separate deployment steps to remove a decision table parameter, and activate the decision table.

**Note** After you upload a CSV file to a CSV-based decision table in the target org, you can't deploy changes to the metadata of that decision table.

Here's how you can migrate decision tables that use custom fields:

- First, migrate the custom fields in your decision table to the target org by using Package Manager.
- Next, verify that the custom fields are present in the target org.
- Finally, migrate the decision table by using Package Manager.

# Decision Tables for Developers

Developers in your Salesforce org can employ developer resources to use and extend decision tables.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

## Connect REST API

- Look up decision tables
- Look up decision tables by passing input values in bulk

## Metadata API

- Create a decision table

## Invocable Actions

- Look up decision tables using the custom invocable action
- Refresh decision tables using the custom invocable action

## Error Codes for Decision Tables

| Final Codes Representation | Error/Warning Message |
|---|---|
| BREDT201 | One or more fields don't exist in the decision table inputs |

| Final Codes Representation | Error/Warning Message |
| --- | --- |
| BREDT401 | You can't override operators for this decision table as override type is set to condition.<br><br>You can't override conditions for this decision table as override type is set to operator. |
| BREDT402 | Incompatible operator entered for field name |
| BREDT403 | ConditionCriteria values {0} are invalid. Enter valid conditions and try again. |
| BREDT404 | The decision table must have at least one Input and one Output field |
| BREDT405 | Condition criteria can't contain ROWCRITERIA parameter |
| BREDT406 | The condition criterion does not contain the sequence number of one or more input fields. Specify the sequence number of each input field and try again.<br><br>The condition criteria does not contain the sequence number for one or more input fields. Specify the sequence number for each input field that's not used for grouping.<br><br>The condition criteria does not contain the sequence number for one or more input fields. Specify the sequence number for each input field whose operator isn’t Equals. |
| BREDT407 | The condition criteria contains one or more invalid sequence numbers. Specify valid sequence numbers for each input field and try again. |
| BREDT408 | The condition criteria is invalid. Specify a valid condition criteria and try again |
| BREDT409 | Specify a condition criteria for the custom logic and try again. |
| BREDT410 | Invalid operator symbol. |
| BREDT501 | You can't override condition criteria without overriding condition type with it. |

| Final Codes Representation | Error/Warning Message |
|---|---|
| BREDT502 | Overriding operator for a combination of Equals, Required, and All conditions are met (AND) fields is not allowed. |
| BREDT503 | Operator overriding is not enabled for this row.<br><br>Condition overriding is not enabled for this row. |
| BREDT504 | You can't specify an OR type condition criteria when the Condition Type is AND. Either change the Condition Type to OR or specify an AND condition criteria.<br><br>You can't specify an AND type condition criteria when the Condition Type is OR. Either change the Condition Type to AND or specify an OR condition criteria. |
| BREDT505 | The specified conditionType doesn't support grouping fields in the conditionCriteria parameters. Remove the fields where isGroupByField is set to true and try again.<br><br>The specified conditionType doesn't support Equals operator fields in the conditionCriteria parameters. Remove the fields where operator is Equals and try again. |
| BREDT506 | Parsing Error |
| BREDT507 | Illegal Argument exception Error for Row Level Condition Parser |
| BREDT508 | Guardrail limit exceded exception for Row Level Condition Parser |
| BREDT509 | Condition criteria contains AND and OR operators without brackets. Group the conditions with brackets and try again.<br><br>When conditionType is set to Any or All, enter only alphanumeric characters in conditionCriteria without brackets. |

# Expression Sets

Expression sets–the calculation engine of business rules–consist of a series of steps connected in a logical flow built from variables, constants, conditions, calculations, lookups, and aggregations. Expression sets accept JSON input from and return JSON output to the digital processes that call the sets.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

An expression set can have multiple versions with different step elements. Each version is required to have a start date time and a rank, and if necessary, an end date time. Multiple versions of an expression set can be enabled simultaneously. When more than one enabled version matches an expression set call, and the date time periods overlap, the version with the highest rank is chosen. For example, if two enabled versions have rank 1 and rank 2, the version with rank 2 is chosen.

You can associate a context definition with the expression set, and use the definition's tags as list variables in the expression set version.

### Expression Set Resources

Expression set versions can have these resource types: variables and constants. Define and use variables–the placeholders for dynamic values–and constants–the placeholders for fixed values–in expression sets for inputs, outputs, and values passed between steps. Numeric variables and constants follow specific assignment rules.

### Object Variables in Expression Sets

Use fields from objects as variables in an expression set. To use object fields as variables in expression sets, the fields must have field aliases defined. You can either use the field aliases available in your org, or have your admins create aliases for the objects fields that you plan to use in an expression set. Aliases are relevant and user-friendly names that are created for underlying source object fields. Field aliases are grouped under an object alias. Object and field aliases are created in Setup and are categorized by a usage type and usage subtype. When you create or modify an expression set, the field aliases related to the expression set's usage type and usage subtype are available to you as variables in the expression set.

### List Variables in Expression Sets and Subexpressions

With the integration of Context Service with Business Rules Engine, you can use attribute tags from context definitions as list variables in expression sets and subexpressions. An expression set or subexpression is context-aware if it is related to a context definition. When you create an expression set, select a context definition that's available in your Salesforce org to associate the definition with the expression set. When you open a version of the expression set in Expression Set Builder, you see the context's tags as list variables in the Resources list in the builder.

### Supported Data Types in Expression Sets

Expression set versions support resources of specific data types. A resource's data type determines the operators that can be used with the resource.

### Operators and Functions in Expression Sets

Formulas in expression sets support logical, comparison, mathematical, and string operators, and functions for dates and numbers.

### How Null Values Are Handled in Expression Sets

You can make sure that null values or preconfigured default values in decision tables are preserved during calculations in expression sets based on your business requirements.

### Step Elements to Build Your Expression Sets

Step elements are the building blocks of an expression set. A new expression set version is always blank. Each element forms a step in an expression set. Use the elements in the Expression Set Builder to form logical steps in the expression set. Steps in an expression set run sequentially. Depending on the type of the step element, a step can perform calculations, lookups, or condition evaluations that determine how the next step in the expression set is processed.

### Expression Set Templates

An expression set template is a file-based expression set with prebuild a new expression set or an expression set version.

### Create An Expression Set

Creating an expression set is a multistep process. When you create an expression set, the set's first version is created by default. Then, you configure the version and create variables and constants for inputs, outputs, and values that are used by and passed between the steps. Next, you add a logical sequence of steps that perform calculations or evaluate conditions using variables, constants, and decision matrices. After you create the flow, you simulate your expression set version with test values to verify that the set is working as expected, and then activate the expression set. To update an expression set version, save it as a version in a new expression set or as a new version within the existing expression set.

### How to Create an Expression Set That Uses Context Definition

This example shows how to associate a context definition with an expression set, and how to use the definition's tags as list variables in the expression set version. The example also shows how to verify if the expression set version works as expected.

### Manage Expression Sets

Learn how to clone an expression set, open an expression set version in the builder, and deactivate an expression set version.

### Considerations for Migrating Expression Sets

Generally, expression sets are migrated to production orgs after they're tested in the development and staging sandbox orgs, and verified to work as expected. Expression sets can contain references to dependent elements such as decision matrices, decision tables, subexpressions, context definitions, field and object aliases, and explainability message templates.

### Expression Sets for Developers

Developers in your Salesforce org can employ expression set developer resources to use and extend expression sets.

# Expression Set Resources

Expression set versions can have these resource types: variables and constants. Define and use variables–the placeholders for dynamic values–and constants–the placeholders for fixed values–in expression sets for inputs, outputs, and values passed between steps. Numeric variables and constants follow specific assignment rules.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Here are some considerations for creating and using expression set resources.

- When you create an expression set version resource, make sure that the resource name begins with a letter. A resource name can contain periods, but not special characters, spaces, and consecutive underscores.
  Examples of valid resource names: Example1, Example_1, Example.1, example, example1
  Examples of invalid resource names: Example__1, 1Example, Example%1, _example1
- When you create an expression set, you can initialize resources with null or default values.
  - To initialize the expression set version resources with null values, turn off the Initialize resources to default values option when you create an expression set. To customize how null values are handled, use the IsNull and IsNotNull operators and functions in the condition, list filter, and calculation steps.
  - To initialize the expression set version resources with default values, turn on the Initialize resources to default values option.
- If you use the lookup table element as a step in an expression set version, the input and output variables from the table are imported as local resources of the expression set version.
- When you create new resources in the expression set version, make sure that the resource names aren't the same as the input variable names of the imported lookup tables.
- When you enter your calculation formula or use the advanced mode to enter the condition logic manually, use a space before and after an operator.
- Ensure that the name of a field alias that’s available for an expression set’s usage type or subtype, and the name of a variable aren’t the same.
- Ensure that you map usage subtypes to the correct usage type.

## Guidelines for Using Numeric Variables in Expression Sets

- If a variable is to the left of one equal sign (=), the variable's current value is used in the calculation. For example, `x + variable = y` means that the value of `y` is `x` plus the current value of *variable*.
- If a variable is to the right of one equal sign (=), the equation's output is the new variable value. For example, `x + y = variable` means that the value of *variable* is the sum of `x` and `y`.
- The current value of a variable on either side of a comparison operator (==, >, >=, <, <=, or <>) is used

to determine whether the comparison is true.
For example, `x + y == variable` returns `true` if the sum of `x` and `y` is equal to *variable*.

- A variable keeps the assigned value until another operation changes the value.
  For example:

```
x = 2
y = 3
variable = 0
```

Step 1: `x + y = variable` // 2 + 3 = 5 // variable = 5
Step 2: `y + y = variable` // 3 + 3 = 6 // variable = 6
Step 3: `variable + y = variable` // 6 + 3 = 9 // variable = 9
Step 4: `variable + variable = variable` // 9 + 9 = 18 // variable = 18

See Also

*Salesforce Help*: Supported Data Types in Expression Sets
*Salesforce Help*: Operators and Functions in Expression Sets

# Object Variables in Expression Sets

Use fields from objects as variables in an expression set. To use object fields as variables in expression sets, the fields must have field aliases defined. You can either use the field aliases available in your org, or have your admins create aliases for the objects fields that you plan to use in an expression set. Aliases are relevant and user-friendly names that are created for underlying source object fields. Field aliases are grouped under an object alias. Object and field aliases are created in Setup and are categorized by a usage type and usage subtype. When you create or modify an expression set, the field aliases related to the expression set's usage type and usage subtype are available to you as variables in the expression set.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

With object field aliases, the hierarchy of the underlying source field in the object is separated from the object's alias. This separation makes the creation of aliases that are relevant to your industry's requirement easy.

**Example** An expression set finds the nearest emergency service for a patient. To calculate the distance from the patient's address to the closest emergency service, the expression set uses the PostalCode field from the Account object. The admin creates the alias, Patient, for the Account object, and another alias, ZipCode, for the PostalCode field in the Account object. The ZipCode alias maps to the Account.Contact.Address.PostalCode field. With this mapping, when you build an expression set, you avoid scanning through the hierarchy of the fields in the Account object to get to

the PostalCode field. You select Patient as the object alias, and all its field aliases are available to you. You then select the field alias Patient.ZipCode. When the expression set runs, the underlying PostalCode field from the Account object is considered.

### Object and Field Alias Considerations

Let's look at some points to consider when you create, modify, or delete object and field aliases.

### Create Source Field Aliases

You can use only object fields that have field aliases as variables in expression sets. Create field aliases from Setup. When you create a field alias, you can group the alias under an existing object alias, or create an object alias and then add the field alias.

### Use Source Field Aliases in Expression Sets

Use field aliases to use object fields as variables in expression set conditions.

See Also

*Salesforce Help*: Supported Data Types in Expression Sets
*Salesforce Help*: Operators and Functions in Expression Sets

## Object and Field Alias Considerations

Let's look at some points to consider when you create, modify, or delete object and field aliases.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

- You can create field aliases for source fields only if you have access to the underlying source object to which the source field belongs.
- You can't modify or delete a field alias that's used in an active expression set.
- You can modify the source field of a field alias only if the field alias isn't used in an expression set version.https://help.salesforce.com/s/articleView?id=ind.netzero_outcome_management_setup.htm&type=5&language=en_US
- You can modify the source field for a field alias that's used in an inactive expression set version if:
  - The source field belongs to the same source object that's used to create the field alias.
  - The data type of the source field is unchanged.
- You can't modify the usage type of a field alias after the alias is created.
- Reference fields that point to multiple objects or do not point to any valid sObjects are not supported.
- Compound fields are not supported.
- The following field types are supported as field aliases:
  - PERCENT
  - CURRENCY
  - NUMERIC

- INTEGER
- LONG
- DOUBLE
- INT
- DATE
- DATEONLY
- BIRTHDAY
- DUEDATE
- BOOLEAN
- AUTONUMBER
- STATICENUM
- DYNAMICENUM
- RECORDTYPE
- PHONE
- FAX
- EMAIL
- ENUMORID
- URL
- ADDRESSCOUNTRY
- ADDRESSSTATE
- CURRENCYCODE
- MULTILINETEXT
- TEXT
- TEXTAREA
- PICKLIST
- STRING
- ENTITYID
- PERSONNAME
- SWITCHABLE_PERSONNAME

## Create Source Field Aliases

You can use only object fields that have field aliases as variables in expression sets. Create field aliases from Setup. When you create a field alias, you can group the alias under an existing object alias, or create an object alias and then add the field alias.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create a field alias: | System Administrator |
| To add a field alias as a variable in an expression set: | Rules Engine Designer |
| To run an expression set that uses field aliases as variables: | Rules Engine Runtime |

1. From Setup, in the Quick Find box, enter `Business Rules Engine`, and then select **Business Rules Engine**.
2. Under Business Rules Engine, select **Object and Field Aliases**.
3. Click **New**.
4. Select a usage type.
5. Create a field alias for the field:
   - If the object has an object alias:
     - Select **Existing Object Alias**.
     - Select an object alias.
     - For Source Field Name, select a field from the source object.
       Specify up to five levels of objects with the last value as an object field.
     - Enter a field alias for the selected field.
   - If the object doesn't have an object alias:
     - Select **New Object Alias**.
     - Enter an alias for the object.
     - For Source Field Name, from the source object for which you just defined the alias, select the field.
       Specify up to five levels of objects with the last value as an object field.
     - Enter a field alias for the selected field.
6. Save your changes.

## Use Source Field Aliases in Expression Sets

Use field aliases to use object fields as variables in expression set conditions.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create a field alias: | System Administrator |
| To use a field alias as a variable in an expression | Rules Engine Designer |

| USER PERMISSIONS NEEDED | |
|---|---|
| set: | |
| To run an expression set that uses field aliases: | Rules Engine Runtime |

1. Open the expression set version in Expression Set Builder.
   For more information, see Open an Expression Set Version in Expression Set Builder

   **Note** You can modify only inactive expression set versions.
2. Add a step element that supports defining a formula or a condition.
3. Use field aliases either as variables in the formula or as conditions in a selected element including business elements:
   - Use field aliases in formulas:
     - In the Formula field of the selected step element, search for and select the name of the object alias to view the object's field aliases.

       **Tip** To see the object aliases available in an expression set, click , and then expand Object Aliases.
     - Select a field alias.
     - Select an output variable.

     **Note** You can't use field aliases as output variables.
   - Use field aliases in conditions:
     - In the Resource field of the selected step element, scroll down to the Object Aliases section.
     - To view the field aliases under an object alias, select the object alias.
     - Select a field alias.
     - Select an operator.
     - In the Value field, enter a supported variable, constant, or field alias.
4. Save your changes.

## List Variables in Expression Sets and Subexpressions

With the integration of Context Service with Business Rules Engine, you can use attribute tags from context definitions as list variables in expression sets and subexpressions. An expression set or subexpression is context-aware if it is related to a context definition. When you create an expression set, select a context definition that's available in your Salesforce org to associate the definition with the expression set. When you open a version of the expression set in Expression Set Builder, you see the context's tags as list variables in the Resources list in the builder.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine |

| enabled |
| --- |

Context definitions are comprehensive, custom data structures that have nodes and attributes. Nodes are hierarchical objects, and attributes are fields in such objects. Nodes and attributes can be mapped to sObjects and their fields. Each node and attribute can have a tag, which abstracts the underlying data structure and enables expression sets to directly point to the attribute and use its data at run time.

When an expression set that uses list variables runs, the results of the expression set are written back to the context definition that the list variables belong to.

## Considerations for Using List Variables in Expression Sets

- A context definition's tags can't be used as list variables in an expression set if the context definition is inactive. Only active context definitions appear in the Context Definition field at the time of creating an expression set.
- Only the nodes and attributes that have tags appear as list variables in Expression Set Builder.
- Attribute tags under the nodes can be used as list variables in expression set version steps.
- Node tags can't be used as list variables in step elements other than business elements.
- List variables can't be used in an expression set version that uses object field aliases.

## Considerations for Using Context-aware Subexpressions

- Context-aware expression sets can include both context-aware and plain subexpressions.
- Context-aware subexpressions must use the same context as their parent expression set.
- Subexpressions in a list group within a context-aware expression set must be context-aware.
- In a list group, filtered values are passed to subexpressions, with filters applying to context tags at the same or lower hierarchical level than the list filter.
- If the subexpression has multiple levels of nesting, only the filter from the immediate parent expression set applies.

**Example** A Salesforce org has an active context definition that's named Video. Here, Reel (1) and Metadata (2) are nodes in the context definition. The tag for the Reel node is Video_Reel, and all attributes in this node have their corresponding tags (3).You created an expression set named Example that uses the Video context definition. You open a version of the Example expression set in Expression Set Builder. The context definition's tags are listed under Resource Manager and are available to be used as list variables in the expression set version.

# Supported Data Types in Expression Sets

Expression set versions support resources of specific data types. A resource's data type determines the operators that can be used with the resource.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Here are the data types available to you to create resources from Resource Manager in Expression Set Builder, and the operators that you can use for each of the data types:

| Data Type | Supported Operators |
|---|---|
| Text | • Equals<br>• Not Equals |
| Number | • Equals<br>• Not Equals<br>• Less Than<br>• Less Than or Equal<br>• Greater Than<br>• Greater Than or Equal |
| Percent | • Equals<br>• Not Equals<br>• Less Than<br>• Less Than or Equal<br>• Greater Than<br>• Greater Than or Equal |
| Currency | • Equals<br>• Not Equals<br>• Less Than<br>• Less Than or Equal<br>• Greater Than<br>• Greater Than or Equal |
| Date | • Equals<br>• Not Equals<br>• Less Than<br>• Less Than or Equal |

| Data Type | Supported Operators |
| --- | --- |
| | • Greater Than<br>• Greater Than or Equal |
| Date & Time | • Equals<br>• Not Equals |
| Boolean | • Equals<br>• Not Equals |

**Note** For decision tables that are used in expression set versions, table columns with these additional data types are also supported: Autonumber, Checkbox, Picklist, Lookup, Master-Detail, Email, and URL. When you use decision tables with these data types in an expression set version, the values are converted to the Text data type.

# Operators and Functions in Expression Sets

Formulas in expression sets support logical, comparison, mathematical, and string operators, and functions for dates and numbers.

REQUIRED EDITIONS

| |
| --- |
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

## Supported Operators in Expression Sets

Mathematical operators take precedence over comparison and string operators, which take precedence over logical operators. Within mathematical operators, `^` and `%` have the highest precedence, followed by `*` and `/`, then `+` and `-`. When precedences are equal, operations are performed from left to right. You can use parentheses, `( )`, to explicitly specify precedence.

**Note** Always include a space before and after an operator.

Logical and Comparison

| Operator | Description |
| --- | --- |
| `&&`<br>`AND` | True if all the joined expressions are true. For example, `A AND B` returns true if both A and B are true. |
| `\|\|`<br>`OR` | True if one of the joined expressions is true. For example, `A AND B` returns true if either A or B is true. |
| `=` | Assigns the value of the second expression to the first expression. For example, `A = 2` sets the value of A to 2. |
| `==`<br>`Equals` | True if the values of the first and second expressions are equal. For example, `2 == 2` returns true. |
| `!=`<br>`Not Equals` | True if the values of the first and second expressions aren't equal. For example, `3 != 2` returns true. |
| `>`<br>`Greater Than` | True if the first expression has a greater value than the second. For example, `3 > 2` returns true. |
| `>=`<br>`Greater Than or Equal` | True if the first expression has a greater value than the second or if the values are equal. For example, both `3 >= 2` and `2 >= 2` return true. |
| `<`<br>`Less Than` | True if the second expression has a greater value than the first. For example, `2 < 3` returns true. |
| `<=`<br>`Less Than or Equal` | True if the second expression has a greater value than the first or if the values are equal. For example, both `2 <= 3` and `2 <= 2` return true. |
| `Is Null` | True if the value of a resource is null. |
| `Is Not Null` | True if the value of a resource isn't null. |

| Operator | Description |
| --- | --- |
| `If` | Returns a given value if true and another value if false.<br><br>`IF(logical_test, value_if_true, value_if_false)` where `logical_test` is the expression you want evaluated; `value_if_true` is the value you want returned if the expression is true; `value_if_false` is the value you want returned if the expression is false.<br><br>Make sure your `value_if_true` and `value_if_false` expressions are the same data type.<br><br>• Example 1: Filtering a list of phones to show only Apple phones. IF(phones:brand == “Apple”, phones:name, null)<br>• Example 2: Removing the dollar sign for calculations IF(SUBSTRING(price,0,1) == “$”, SUBSTRING(price,1),price)<br>• Example 3: Setting a discount value for drivers under 25 if they are good students, and drivers over 25 if they refer two or more people. IF(DriverAge >= 25, IF(FriendsReferred >= 2, true, false), IF(GPA >= 3.0, true, false)) |

Mathematical

| Operator | Description |
| --- | --- |
| `+` | Adds two numbers. |
| `-` | Subtracts the second number from the first. |
| `*` | Multiplies two numbers. |
| `/` | Divides the first number by the second. |
| `%` | Returns the percentage of the first number specified by the second number. For example, `50 % 20` returns 10. |

| Operator | Description |
| --- | --- |
| `^` | Returns the power of the first number specified by the second number. For example, `2 ^ 3` returns 8. |

## Supported Functions in Expression Sets

Aggregate

| Function | Description | Example |
| --- | --- | --- |
| `LISTAVG` | Returns the average of numeric (Number, Currency, Percent) values in a list variable.<br><br>This function is applicable to context definition tags, which are used as list variables in expression set versions. | `LISTAVG(OrderAmount)`, where `OrderAmount` is a list variable of Currency data type that contains multiple records of order amount.<br><br>The function returns the average of all the values in the list variable. |
| `LISTSIZE` | Returns the number of items in a list.<br><br>This function is applicable to context definition tags, which are used as list variables in expression set versions. | `LISTSIZE(ProductName)`, where `ProductName` is a list variable of Text data type that contains multiple product names.<br><br>The function returns the total number of items or records in the ProductName variable. |
| `LISTSUM` | Returns the sum of numeric (Number, Currency, Percent) values in a list variable.<br><br>This function is applicable to context definition tags, which are used as list variables in expression set versions. | `LISTSUM(OrderAmount)`, where `OrderAmount` is a list variable of Currency data type that contains multiple records of order amount.<br><br>The function returns the sum of all the values in the list variable. |

Date and Time

| Function | Description | Example |
| --- | --- | --- |
| `ADDDAY` | Adds the specified number of days to the specified date and returns the resulting date. | `ADDDAY(MyDate, 100)`, where *MyDate* is a variable of the data type Date with the assigned value of `1999-01-01`.<br><br>The function returns `1999-04-11 00:00:00`. |
| `ADDMONTH` | Adds the specified number of months to the specified date and returns the resulting date. | `ADDMONTH(MyDate, 100)`, where *MyDate* is a variable of the data type Date with the assigned value of `1999-01-01`.<br><br>The function returns `2007-05-01 00:00:00`. |
| `ADDYEAR` | Adds the specified number of years to the specified date and returns the resulting date. | `ADDYEAR(MyDate, 100)`, where *MyDate* is a variable of the data type Date with the assigned value of `1999-01-01.`<br><br>The function returns `2099-01-01 00:00:00`. |
| `AGE` | Returns the age for the specified birthdate. | `AGE(Birthdate)`, where *Birthdate* is a variable of the data type Date.<br><br>The function returns the age on the present date for the value assigned to the Birthdate variable. |
| `AGEON` | For a given birth date, returns the age on a specified date. | `AGEON(Birthdate, MyDate)`, where `Birthdate` and *MyDate* are variables of the data type Date with the assigned values of `1999-01-01` and `2001-01-01` respectively. |

| Function | Description | Example |
| --- | --- | --- |
| | | The function returns `2`. |
| `DATEDIFF` | Returns the number of days between two specified dates. If the first date is greater than the second date, the value is returned as a negative number. | `DATEDIFF(StartDate,EndDate)`, where *StartDate* and *EndDate* are variables of the data type Date with the assigned values of `1900-01-1` and `2000-01-01` respectively.<br><br>The function returns `36524`. |
| `DATEVALUE` | Returns the date value of a Date & Time resource based on the timezone settings in your Salesforce org. | `DATEVALUE(BirthDate)`, where `BirthDate` is a resource of Date & Time data type with the assigned value of `2003-05-31T12:15:20.123Z`.<br><br>The function returns `May 31, 2003`. |
| `EOM` | For a specified date, returns the date of the last day of the month. | `EOM(MyDate)`, where `MyDate` is a variable of the data type Date, with the assigned value of `1999-01-11`.<br><br>The function returns `1999-01-31`. |
| `MONTH` | For a specified date, returns the month as an integer (1-12). | `MONTH(MyDate)`, where `MyDate` is a variable of the data type Date, with the assigned value of `1999-01-11`.<br><br>The function returns `1`. |
| `TODAY` | Returns today's date. | `TODAY`() returns today's date. |
| `YEAR` | Returns the year of the specified date as an integer. | `YEAR(MyDate)`, where *MyDate* is a variable of the data type Date, with the assigned value of |

| Function | Description | Example |
|---|---|---|
| | | `1999-01-11`.<br><br>The function returns `1999`. |
| `NOW` | Returns a date/time representing the current moment. | `NOW` () returns the current date and time. |

Logical and Comparison

| Function | Description | Example |
|---|---|---|
| `ISNULL` | True if the value of a resource is null. | `ISNULL(Example)`, where *Example* is a variable without an assigned value.<br><br>The function returns `True`. |
| `ISNOTNULL` | True if the value of a resource isn't null. | `ISNOTNULL(City)`, where *City* is a variable with an assigned value of Mumbai.<br><br>The function returns `True`. |
| `LIKE` | True if the second parameter contains the first parameter. | `LIKE('A', 'ABC')` returns `true`. |
| `NOTLIKE` | True if the second parameter doesn't contain the first parameter. | `NOTLIKE('A', 'ABC')` returns `false`. |
| `BEGINS` | Determines if text begins with specific characters and returns TRUE if it does. Returns FALSE if it doesn't. | `BEGINS(MyString,CompareText)`, where *MyString*, and *CompareText* are variables of the data type Text .<br>`BEGINS(""Expression Set","Express")` returns TRUE. |
| `ISNUMBER` | Returns TRUE if the string is a number. Returns FALSE if a string isn't a number. | `ISNUMBER(MyString)`, where *MyString* is a variable of the data type Text .<br>`ISNUMBER("10")` returns TRUE.<br>`ISNUMBER("tty")` returns FALSE. |

Mathematical

| Function | Description | Example |
|---|---|---|
| `ABS` | Returns the absolute value of a number. The absolute value of a number is the number without its positive or negative sign. | `ABS(MyNumber)`, where *MyNumber* is a variable of the data type Number with the assigned value of `-25`.<br><br>The function returns `25`. |
| `AVG` | Returns the average of two values. | `AVG(3,5)` returns `4`. |
| `EXP` | Returns a value for e raised to the power of a number you specify. | `EXP(Number)`, where *Number* is a variable of the data type Number, with the assigned value of 5.<br><br>The function returns `148`. |
| `LOG` | Returns the base 10 logarithm or the power that the number 10 must be raised to for returning the number specified. | `LOG(MyNumber)`, where *MyNumber* is a variable of the data type Number with the assigned value of `100`.<br><br>The function returns `2`. This means that 10 must be raised to the power of 2 to get 100. |
| `LN` | Returns the natural logarithm or the power that the constant e (2.718281828) must be raised to for returning the specified number. | `LN(MyNumber)`, where *MyNumber* is a variable of the data type Number with the assigned value of `25`.<br><br>The function returns `3.22`. This means that e, which has the constant value of 2.718281828, raised to the power of 3.22 returns 25. |
| `MAX` | Returns the maximum value out of two values. | `MAX(2,10)` returns `10`. |
| `MIN` | Returns the minimum value out of two values. | `MIN(2,10)` returns `2`. |

| Function | Description | Example |
|---|---|---|
| `POWER` | Returns the result of the first number raised to the power of the second number. | `POWER(2, 3)` returns 8 (2 x 2 x 2). |
| `SQRT` | Calculates the square root of a value. | `SQRT(12 * 3)` returns `6`. |
| `SUM` | Returns the sum of two values. | `SUM(1,2)` returns `3`. |
| `MOD` | Returns a remainder after a number is divided by a specified divisor. | `MOD(4, 3)` returns `1` |
| `ROUND` | Rounds the specified number or expression. | ROUND(3.1415, 2) returns 3.14<br><br>ROUND(3.1415 * 3, 0) returns 9. |
| `UP` (Use with ROUND) | Use with ROUND to specify that the rounding result is rounded up (away from zero). | ROUND(2.572, 2, UP) returns 2.58. |
| `DOWN` (Use with ROUND) | Use with ROUND to specify that the rounding result is rounded down (towards zero). | ROUND(2.572, 2, DOWN) returns 2.57. |
| `HALF_UP` (Use with ROUND) | Use with ROUND to specify that therounding result is rounded towards the nearest neighbor (up or down) unless the neighbors are equidistant, in which case it rounds up. | ROUND(2.575, 2, HALF_UP) returns 2.58. |
| `HALF_DOWN` (Use with ROUND) | Use with ROUND to specify that the rounding result is rounded rounded towards the nearest neighbor (up or down) unless the neighbors are equidistant, in which case it rounds down. | ROUND(2.575, 2, HALF_DOWN) returns 2.57. |
| `HALF_EVEN` (Use with ROUND) | Use with ROUND to specify that the rounding result is rounded rounded towards the nearest neighbor (up or down) unless the neighbors are equidistant, in | ROUND(2.57, 1, HALF_EVEN) returns 2.6, which is rounded up to the nearest even one-decimal place. |

| Function | Description | Example |
|---|---|---|
| | which case it rounds to the nearest even neighbor.<br><br>Behaves as for ROUND HALF_UP if the digit to the left of the discarded fraction is odd; behaves as for ROUND HALF_DOWN if it is even. | |

String

| Function | Description | Example |
|---|---|---|
| `CONTAIN` | True if the list of strings in the first parameter contains the string in the second parameter.<br><br>Separate the strings in the list of strings in the first parameter with semicolons (;). | `CONTAIN ("Business;Rules;Set", "Rules")` returns `true`.<br><br>`CONTAIN ("Business;Rules;Set", "Rule")` returns `false`. |
| `REVERSE` | Returns the characters of the string in reverse order. | `REVERSE(MyString)`, where *MyString* is a variable of the data type Text. `REVERSE("Salesforce")` returns "ecrofseIaS". |
| `FIND` | Returns the position of a string in another string | `FIND(search_text,text)`, where *search_text* and *text* are variables of data type Text. `FIND("@","admin@salesforce.com)` returns 6. |
| `LEFT` | Returns the specified number of characters from the beginning of a text string. | `LEFT(MyString,MyPos)`, where *MyString* is a variable of the data type Text and *MyPos* is a variable of the data type Number. `LEFT("www.salesforce.com",3)` returns "www". |
| `MID` | Returns the specified number of characters from the starting position of a text string. | `MID(MyString,NumOfChar,StartingPos)`, where *MyString* is a variable of the data type Text, *NumOfChar* and *StartingPos* are |

| Function | Description | Example |
|---|---|---|
| | | variables of the data type Number. `MID("www.salesforce.com",10,5)` returns “salesforce”. |
| `RIGHT` | Returns the specified number of characters from the end of a text string. | `RIGHT(MyString,MyPos)`, where *MyString* is a variable of the data type Text and *MyPos* is a variable of the data type Number. `RIGHT("www.salesforce.com",3)` returns “com”. |
| `LEN` | Returns number of characters in the string | `LEN(MyString)`, where *MyString* is a variable of the data type Text. `LEN("salesforce”)` returns 10. |
| `INITCAP` | Returns string with initial caps - the first character of each word in upper case and the rest of the string in lowercase. | `INITCAP(MyString)`, where *MyString* is a variable of the data type Text . `INITCAP(“expression set”)` returns “Expression Set”. |
| `LOWER` | Converts all the characters in the string in lowercase. | `LOWER(MyString)`, where *MyString* is a variable of the data type Text . `LOWER(“Expression Set”)` returns “expression set”. |
| `UPPER` | Converts all letters in the specified text string to uppercase. | `UPPER(MyString)`, where *MyString* is a variable of the data type Text . `UPPER(“expression set”)` returns “EXPRESSION SET”. |
| `SUBSTITUTE` | Substitutes old text with new text in a text string. | `SUBSTITUTE(MyString,OldText,NewText)`, where *MyString*, *OldText*, and *NewText* are variables of the data type Text . `SUBSTITUTE(“xyz@salesforce.com”,“xyz”,“itadmin”)` returns “itadmin@salesforce.com”. |

| Function | Description | Example |
|---|---|---|
| `TEXT` | Converts a percent, number, date, date/time, or currency type field into text anywhere formulas are used. | `TEXT(MyValue)`. `TEXT(Revenue)` where *Revenue* is $20000 returns “20000”. |
| `TRIM` | Removes the spaces and tabs from the beginning and end of a text string. | `TRIM(MyString)`, where *MyString* is a variable of the data type Text . `TRIM(“ Expression Set ”)` returns “Expression Set”. |
| `VALUE` | Converts a text string to a number. | `VALUE(MyString)`, where *MyString* is a variable of the data type Text . `VALUE(“10”)` returns the number 10. |
| `BEGINS` | Determines if text begins with specific characters and returns TRUE if it does. Returns FALSE if it doesn't. | `BEGINS(MyString,CompareText)`, where *MyString*, and *CompareText* are variables of the data type Text . `BEGINS(““Expression Set”,“Express”)` returns TRUE. |
| `LPAD` | Inserts characters that you specify to the left-side of a text string. | `LPAD(MyString,PadLength,PadString)`, where *MyString*, and *PadString* are variables of the data type Text . *PadLength* is a variable of data type Number. `LPAD('my_company.com',15,'z')` returns “zmycompany.com”. |
| `RPAD` | Inserts characters that you specify to the right-side of a text string. | `RPAD(MyString,PadLength,PadString)`, where *MyString*, and *PadString* are variables of the data type Text . *PadLength* is a variable of data type Number. `RPAD('my_company.com',15,'z')` returns “mycompany.comz”. |

# How Null Values Are Handled in Expression Sets

You can make sure that null values or preconfigured default values in decision tables are preserved during calculations in expression sets based on your business requirements.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

When you create an expression set, you can initialize resources with null or default values.

- To initialize the expression set version resources with null values, turn off the Initialize resources to default values option when you create an expression set. To customize how null values are handled, use the IsNull and IsNotNull operators and functions in the condition, list filter, and calculation steps.
- To initialize the expression set version resources with default values, turn on the Initialize resources to default values option.

If you use the lookup table element as a step in an expression set version, the input and output variables from the table are imported as local resources of the expression set version.

| Initialize Resources to Default Values | Decision Table Output | Default Value | Expression Set Output | Comment |
|---|---|---|---|---|
| On | Null | 100 (value set by user) | 100 | The default value overwrites the Null value from the decision table if **Initialize resources to default values** is on. |
| On | Null | none | 0 (internal default value) | NA |
| Off | Null | - | Null | The Null value from the decision table is preserved if **Initialize resources to default values** is off. |

| Initialize Resources to Default Values | Decision Table Output | Default Value | Expression Set Output | Comment |
|---|---|---|---|---|
| On/Off | 10 (Any non-null value) | - | 10 | The non-null value from the decision table is set as the expression set output. |

**Note** When the decision table doesn't have any matching row, the variables in the expression set retain their previous values. The step is marked Executed in the simulation page.

See Also

*Salesforce Help*: How Null Values Work in Decision Tables

# Step Elements to Build Your Expression Sets

Step elements are the building blocks of an expression set. A new expression set version is always blank. Each element forms a step in an expression set. Use the elements in the Expression Set Builder to form logical steps in the expression set. Steps in an expression set run sequentially. Depending on the type of the step element, a step can perform calculations, lookups, or condition evaluations that determine how the next step in the expression set is processed.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

To start building your expression set version, open the version in the Expression Set Builder canvas, click ⊕, and then select an element.

### Calculation

A calculation step element performs functions and mathematical operations on expression set resources–constants, variables, field aliases, and list variables.

### Conditional Group and Condition

A conditional group step element runs a group of steps if the expression in the condition element in the group evaluates to true. A conditional group can have these step elements: condition, calculation, lookup table, and subexpression. A condition step element is always the first element in the group. A condition step runs the step that follows if the condition expression evaluates to true. A condition step can only be used as part of a conditional group element, and not as a standalone step element.

### Branch

A branch step element can have multiple conditional paths that an expression set version can take based on the priority that you assign to each path. A branch can have up to six paths, including the default path. Paths have priorities in a descending order from left to right. The left-most path has the highest priority. In a non-default path, the first step is the entry condition for the path. This condition determines whether to enter the path. If the branch step doesn't meet the entry conditions for the paths, it takes the default path–the path with no entry condition.

### Lookup Table

A lookup table step element matches your input values with the input rows in a decision matrix or a decision table. The table then returns the matching row's output to the expression set version that the table is used in.

### Subexpression

When you add a subexpression step element to an expression set version, you can select another expression set that you want to use within the version. The subexpression then returns the result of the expression set to the version that the subexpression is used in. You can use resources from a subexpression, including any input and output variables from lookup tables in the subexpression in other steps in the expression set version. You can use only expression sets with at least one active version as subexpressions in an expression set version.

### List Group and List Filter

A list group step element filters items in a list based on the filter conditions, and then performs further operations on the filtered list. A list group can have these elements: list filter, calculation, lookup table, and business element. A list filter is always the first step in a list group, and defines how items in list variables are filtered. A list group can have only one list filter, but multiple calculation, lookup table, or business elements.

### Aggregate and Aggregate Group

An Aggregate step performs a SUM, AVG, MAX, or MIN function on multiple runs of the preceding steps in an expression set version. Aggregate Groups run the previous steps in the version one time for each set of input values, and then aggregate the results of the steps based on the aggregation logic. Aggregate Groups can have these step elements: Aggregate and Calculation. You can also use Business Elements in an Aggregate Group, if the elements are shipped with your Industries cloud and available in your Salesforce org. Aggregate is always the first step element in an Aggregate Group. Aggregate steps can only be part of Aggregate Groups and can't be used as standalone elements in an expression set version.

### Business Element

Business elements perform the action that's defined in the element. Each business element can have a specific purpose based on your cloud's use cases. If a business element is available in your Salesforce org and is specific to the usage type of your expression set, then you can use it as a step in your expression set version.

## Calculation

A calculation step element performs functions and mathematical operations on expression set resources–constants, variables, field aliases, and list variables.

REQUIRED EDITIONS

Available in: Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled

**Example** Use the AGE function to calculate a person's age from their birth date.

- Add a calculation step element in an expression set version. In the Formula field, select **AGE (Date)**.
- Replace (Date) in the Formula field with a variable of the Date data type.
- In the Output Variable field, select a variable of the Number data type.

Here, `StudentBirthDate` is a variable of the Date data type that stores the birth dates of students. `StudentAge` is a variable of the Number data type that stores the output returned by the AGE function.

## Conditional Group and Condition

A conditional group step element runs a group of steps if the expression in the condition element in the group evaluates to true. A conditional group can have these step elements: condition, calculation, lookup table, and subexpression. A condition step element is always the first element in the group. A condition step runs the step that follows if the condition expression evaluates to true. A condition step can only be used as part of a conditional group element, and not as a standalone step element.

REQUIRED EDITIONS

Available in: Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled

**Example** A rule designer wants to create a conditional rule that calculates the discounted student fees for students aged 60 and above. The designer has created these expression set resources to be used in the Conditional Group element.

| resource name | type | data type |
|---|---|---|
| Student_Age | Variable | Number |
| Student_Fees | Constant | Currency |
| Discounted_Student_Fees | Variable | Currency |

Here's how the designer has configured the conditional group: Here, in the condition element, the values in the Resource, Operator, and Value fields represent this condition: The student's age is greater

than or equal to 60.The calculation element defines the action that is taken if the condition evaluates to true. The calculation runs only if a student's age is 60 or above. Student_Fees is the constant with a value of $2,000. The result of calculating 2,000 minus 30% of 2,000 is stored in the output variable Discounted_Student_Fees.

**Default and Advanced Modes in the Condition Element**

The condition element in expression sets has two modes: default and advanced. When you add a condition element to an expression set, the condition opens in the default mode. The default mode has a no-code user interface to build your condition expression. You can also switch to the Advanced mode in the condition element to enter your expression manually.

## Default and Advanced Modes in the Condition Element

The condition element in expression sets has two modes: default and advanced. When you add a condition element to an expression set, the condition opens in the default mode. The default mode has a no-code user interface to build your condition expression. You can also switch to the Advanced mode in the condition element to enter your expression manually.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

To add a condition to an expression set, open the expression set in the Expression Set Builder, click ⊕, and then select **Condition**.

The condition element opens in default mode.

## Default Mode

Select a condition requirement from the dropdown menu.

To add more conditions to the conditional group, click **Add Condition**.

In the Resource field, select a variable or a constant from the list of resources that you created for the expression set. Next, select an operator, and then enter a value for the variable or the constant that you selected.

## Advanced Mode

To switch to the Advanced mode, click the show actions menu for the condition, and then select **Switch to Advanced Mode**.

The advanced mode automatically converts the selections you made in the default mode to an expression. If needed, you can manually modify the expression.

## Branch

A branch step element can have multiple conditional paths that an expression set version can take based on the priority that you assign to each path. A branch can have up to six paths, including the default path. Paths have priorities in a descending order from left to right. The left-most path has the highest priority. In a non-default path, the first step is the entry condition for the path. This condition determines whether to enter the path. If the branch step doesn't meet the entry conditions for the paths, it takes the default path–the path with no entry condition.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Example** A university offers discounted fees to students based on a student's age and gender. An expression set contains the calculations to evaluate a students' age and return the discounted fee amount based on the entry conditions configured in the paths in branch element.

Conditions for Branching

| paths and entry condition | path priority |
|---|---|
| Path 1: Female students aged 50 and above are eligible for a 40% discount on fees. | 1 |
| Path 2: Female students aged 35 and above, but under 50 are eligible for a 25% discount on fees. | 2 |
| Default Path: Students who don't meet the first two conditions are eligible to a 10% discount on fees. | Default |

The expression uses these variables in defining calculations and conditions.

Variables in the Expression Set

| resource name | resource type |
|---|---|
| Age | Number |
| Birthdate | Date |
| IsFemale | Boolean |
| UniversityFees | Currency |
| DiscountedFees | Currency |

The first step in the expression set calculates the age of the student from their birthdate, using the AGE function. The second step in the expression set takes the Age value calculated in the first step, evaluates paths from left to right priority-wise, and returns the discounted fee. The expression set is simulated with the following test input values for the variables:

- IsFemale: True
- Birthdate: January 1, 1977
- UniversityFees: 50,000

The first step in the expression set returns the student age as 45. This age meets the conditions set in the second path. The expression set enters the second path and returns the discounted fee of 37,500 based on the formula in the second path.

## Lookup Table

A lookup table step element matches your input values with the input rows in a decision matrix or a decision table. The table then returns the matching row's output to the expression set version that the table is used in.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

A lookup table element can look up a decision matrix or a decision table. Only active decision tables and active decision matrices with at least one active version can be selected for lookup. If you select a decision matrix that has multiple active versions, the active version with the highest rank is considered.

**Example** A rule designer wants to create an expression set that calculates the discounted base fare for public transport based on certain criteria. The discount eligibility criteria are defined in a decision matrix. The matrix's input columns capture the age range and family income level, and the

output column captures the discount eligibility based on the age and family income. Based on the outcome of the decision matrix lookup, the designer further calculates the discounted base fare for eligible citizens. Here's how the rule designer uses the lookup table element and the conditional group element in the expression set version. Here, the expression set version looks up the decision matrix, and updates the value of the Eligible_For_Discount variable as true or false based on the input values. The conditional group uses the Eligible_For_Discount variable to define the condition that determines whether a discounted base fare is to be calculated. The calculation step calculates the discount if the value of the Eligible_For_Discount variable is true.

### Variable Mapping in Lookup Tables

When you add a lookup table element step in an expression set version, the input and output variables in the table are imported into the version. You can then map the table's variables to the resources from the expression set version. When you map the lookup table's resources to the resources from the expression set version that the table is used in, a clear alignment between the two types of resources is established. This alignment simplifies the process of building an expression set. For example, you can map the output variable of a step that precedes a lookup table step to the input variable in the lookup table. With this mapping, the output of the previous step is automatically passed to the subsequent lookup table as the input.

### Lookup of Multiple Input Values

When you add a lookup table to an expression set version that is associated with a context definition, you can map the table's input variables to list variables from the definition. With this mapping, you can look up multiple values in the list variable at the same time and get their corresponding output values.

## Variable Mapping in Lookup Tables

When you add a lookup table element step in an expression set version, the input and output variables in the table are imported into the version. You can then map the table's variables to the resources from the expression set version. When you map the lookup table's resources to the resources from the expression set version that the table is used in, a clear alignment between the two types of resources is established. This alignment simplifies the process of building an expression set. For example, you can map the output variable of a step that precedes a lookup table step to the input variable in the lookup table. With this mapping, the output of the previous step is automatically passed to the subsequent lookup table as the input.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Note** An expression set can read only one result even if a decision table provides multiple results. So, if you plan to use the decision table in an expression set, while creating the decision table, set Filter Result By to **First Match** or **Any Value**, which are both specific to single values.

After you select a lookup table in an expression set version, and the table's variables are imported into the version, here's how the mapping works:

| Scenario | Mapping Behavior |
| --- | --- |
| A lookup table is used in one lookup table step in the expression set version and one or more table variables are mapped to the version resources | The table variables that are mapped to the version resources are removed from the resources list.<br><br>If the mapping is removed, the imported variables reappear in the resources list. |
| A lookup table is used in more than one lookup table step in the expression set version, but the table variables are mapped to the version resources in only one of the steps | The resource list shows both the imported variables and the expression set version resources that the variables are mapped to. |

**Example** Anne, a rule designer, creates an expression set version to determine the eligibility of citizens for a discount on the base fare for public transport, based on their age and family income level. She defines Citizen_Birthdate and Citizen_Age as the local resources in the version. Anne uses a calculation step element to get the citizen's age from their birth date. She then uses a lookup table step to determine a citizen's discount eligibility based on certain parameters. Here's how she configured the decision matrix that she uses in the lookup table step.

| Column Name | Column Type | Data Type |
| --- | --- | --- |
| Age | Input | Number range |
| Family_Income_Level | Input | Text |
| Eligible_For_Discount | Output | Boolean |

In this example, the table's input variables Age and Family_Income_Level, and the output variable Base_Fare_Eligible_For_Discount are imported in the version and appear as local resources in Resource Manager. To pass the value of the Citizen_Age variable from the calculation step to the Age column in the lookup table, Anne clicks **Map Variables** in the lookup table step. She then maps the imported input variable Age to Citizen_Age. Because of the mapping, the imported lookup table variable Age is removed from the resource list, and the lookup table step replaces the table's input variable Age with the mapped resource Citizen_Age.

## Lookup of Multiple Input Values

When you add a lookup table to an expression set version that is associated with a context definition, you can map the table's input variables to list variables from the definition. With this mapping, you can look up multiple values in the list variable at the same time and get their corresponding output values.

REQUIRED EDITIONS

Available in: Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled

**Note** Only users with the Rule Engine Advanced Designer permission set can map lookup table variables to list variables.

**Example** Donisha, a rule designer, wants to design an expression set version to calculate the discounted price on certain items in an apparel catalog. Here's how she configures the expression set:

- Creates an expression set that's associated with a context definition that has tags. The tags store various details related to apparel.
- Creates a decision matrix named Apparel_Price to look up the price of an apparel item based on the item's size, brand, and fabric.
- Adds a lookup table step to the expression set version and selects the decision matrix.
- Adds a calculation step to perform further calculation on the prices retrieved from the lookup table step.

The matrix column names are imported into the version as the version's local resources. To get the prices of multiple apparel items from the lookup table simultaneously, Donisha maps the table's variables to list variables from the ApparelCatalog context definition that's associated with the expression set. She knows that the mapping is successful when the local variables that are imported from the lookup table are removed from the resources list. The calculation step takes the corresponding prices for the inputs from the lookup table and calculates the revised prices at a discount of 15%. The revised price is written back to the context definition associated with the Price tag. To test whether the steps worked as expected, Donisha simulates the expression set version.

**Note** To simulate expression set versions that use list variables in the steps, use the Advanced input mode to enter your inputs in JSON format.

Here's how Donisha simultaneously passes two sets of input values for the mapped list variables to the expression set version.

```
{
  "ApparelCatalog": [
    {
      "Size": "L",
      "Fabric": "Wool",
      "Brand": "ABC",
      "Name": "Sweater"
    },
    {
      "Size": "S",
```

```
      "Fabric": "Denim",
      "Brand": "NA",
      "Name": "Dress"
    }
  ]
}
```

When the version is simulated, it generates the results for the two sets of inputs for both the lookup table step and the calculation step.

## Subexpression

When you add a subexpression step element to an expression set version, you can select another expression set that you want to use within the version. The subexpression then returns the result of the expression set to the version that the subexpression is used in. You can use resources from a subexpression, including any input and output variables from lookup tables in the subexpression in other steps in the expression set version. You can use only expression sets with at least one active version as subexpressions in an expression set version.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Here are the considerations for using expression sets as subexpressions:

- Expression sets can have a depth of up to two levels of subexpressions. For example, an expression set A that doesn't reference any subexpressions is considered to have a depth of 0. If A references B, whose depth is 0, A has a depth of 1. If B references another subexpression C, then B has a depth of 1, and A has a depth of 2. However, if C further references D, then the depth of A becomes 3 levels, which isn't supported.
- When an expression set that has multiple active versions is used as a subexpression in another expression set, the subexpression version with the highest rank is considered.
- You can't modify the versions of a subexpression if the modifications change the input and output variables in the parent expression set.
- You can modify a subexpression if your changes are additions that don't modify the existing variables in the parent expression set.

  **Tip** Because you can't modify an active subexpression version, clone the version that you want to modify. Make the necessary changes. Next, give the version a higher rank than the previous versions, and save and activate the version. When the parent expression set is run the next time, the modified subexpression version with the higher rank is considered.

- If the expression set version has a context definition, the subexpression must be associated with the same context definition as its parent.

  **Tips to Work with Subexpressions**
  You can't modify a subexpression version when it's active. You can reuse an active subexpression without creating a subexpression from scratch. You can map the subexpression's input and output variables to the local resources of the expression set version that the subexpression is used in. You can directly replace or delete subexpressions added in expression set versions, without deleting mapped variables or subsequent steps that reference the subexpression's variables.

## Tips to Work with Subexpressions

You can't modify a subexpression version when it's active. You can reuse an active subexpression without creating a subexpression from scratch. You can map the subexpression's input and output variables to the local resources of the expression set version that the subexpression is used in. You can directly replace or delete subexpressions added in expression set versions, without deleting mapped variables or subsequent steps that reference the subexpression's variables.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| Scenario | Tips |
|---|---|
| A subexpression version with the highest rank is used as a step in an expression set version. You want to update the steps in the subexpression version without changing the input and output variables.<br><br>For example, the subexpression version has 10 input and output variables. Without changing any variable, you want to update the calculation formulas or the conditions that the variables are used in. | • Save the subexpression version as a new version.<br>• Update the step elements and save the version.<br>• Give the version the highest rank.<br>• Activate the version.<br><br>When the parent expression set version is run the next time, the updated subexpression version that has the highest rank is considered. |
| A subexpression version with the highest rank is used as a step in an expression set version. You want to use some steps and variables in the subexpression version, but you also want to make other changes that require you to add or remove variables. | • Deactivate the version of the parent expression set that references the subexpression.<br>• Remove the subexpression from the parent expression set version.<br>• Deactivate the subexpression version that you |

| Scenario | Tips |
| --- | --- |
| | want to modify.<br>• Modify the version.<br>• Verify that the version has the highest rank.<br>• Activate the version.<br>• Add the subexpression back to the parent expression set version, and activate the version.<br><br>When the parent expression set version is run the next time, the version considers the updated subexpression version. |
| When a subexpression version is added in an expression set version step, variables are automatically created in the expression set version corresponding to the subexpression's variables. You now want to map the subexpression's variables to different local resources in the expression set version. | • Click the **Map Variables** button on the subexpression step of the expression set version. The subexpression's input and output variables are listed.<br>• Choose the expression set version's local resource that you want to map your subexpression's variables to.<br>• After you map the subexpression's variables explicitly, the corresponding variables that were automatically created in the expression set version are now deleted. |
| A subexpression is used in an active expression set version and its output variables are used in subsequent calculation steps of the expression set version. You now want to delete the subexpression from the expression set version. | • Delete the subexpression step in the expression set version.<br>• The subexpression is deleted without explicitly deleting the subsequent expression set version steps that reference the subexpression's variables.<br>• Variables that were automatically created in the expression set version corresponding to the deleted subexpression's variables are now deleted.<br>• Fix any errors shown in the subsequent expression set version steps that reference the deleted subexpression's variables. |
| A subexpression is used in an active expression set version and its output variables are used in subsequent calculation steps. You now want to substitute the subexpression in the expression set | • In the subexpression step in the expression set version, click **X** to remove the subexpression. |

| Scenario | Tips |
| --- | --- |
| version with another one that better suits your business needs. | • Select the new subexpression that you want to use.<br>• The subexpression is changed without explicitly deleting the subsequent expression set version steps that reference the old subexpression's variables.<br>• Input variables of the new subexpression that match the old subexpression's variables are retained.<br>• Explicitly map the output variables of the new subexpression.<br>• Fix the errors that are shown in subsequent expression set version steps that reference the old subexpression's variables. |

## List Group and List Filter

A list group step element filters items in a list based on the filter conditions, and then performs further operations on the filtered list. A list group can have these elements: list filter, calculation, lookup table, and business element. A list filter is always the first step in a list group, and defines how items in list variables are filtered. A list group can have only one list filter, but multiple calculation, lookup table, or business elements.

REQUIRED EDITIONS

| |
| --- |
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Note** You can add a business element to a list group only if your Salesforce org has a business element. The business element must have the same usage type as the expression set whose version the element is to be added to.

**Example** Donisha, a rule designer, wants to design an expression set version to filter items in an apparel catalog based on the item's fabric and size. She then wants to apply a 10 percent discount on the price of such filtered list items. Here's how she configures the expression set and its version:

- Creates an expression set that's associated with a context definition that has attribute tags that store various details related to the apparel catalog.
- Adds a list group step element to the expression set version.

- Configures the list filter in the list group, to filter items from the catalog that are made from cotton or have the size S.
- Adds a calculation step to calculate the updated price of the filtered items after deducting 10% from the price.

In this image, the items listed under ApparelCatalog are context definition tags that appear as list variables in the version. In the list filter, Donisha adds two conditions: The Fabric is cotton or the Size is S. The filter shows the items from the apparel list that meet either condition. Donisha then adds a calculation step to get the updated price for the filtered items. Price is a tag from the ApparelCatalog context definition and is used as a list variable in the calculation. Therefore, when the expression set runs, the updated price is written back to the context definition.She then simulates the version to verify that her rule works as expected.

**Note** To simulate expression set versions that use list variables in the steps, use the Advanced input mode to enter your inputs in JSON format.

Here's how she simultaneously passes multiple sets of input values for the simulation.

```
{
  "ApparelCatalog": [
    {
      "Size": "M",
      "Price": "50",
      "Fabric": "Silk"
    },
    {
      "Size": "S",
      "Price": "45",
      "Fabric": "Linen"
    },
    {
      "Size": "S",
      "Price": "40",
      "Fabric": "Cotton"
    },
    {
      "Size": "L",
      "Price": "55",
      "Fabric": "Cotton"
    }
  ]
}
```

The simulation results indicate that the filter condition evaluated to true for three of the four values passed for the list variables. Here's the corresponding discounted price of the filtered items.

## Aggregate and Aggregate Group

An Aggregate step performs a SUM, AVG, MAX, or MIN function on multiple runs of the preceding steps in an expression set version. Aggregate Groups run the previous steps in the version one time for each set of input values, and then aggregate the results of the steps based on the aggregation logic. Aggregate Groups can have these step elements: Aggregate and Calculation. You can also use Business Elements in an Aggregate Group, if the elements are shipped with your Industries cloud and available in your Salesforce org. Aggregate is always the first step element in an Aggregate Group. Aggregate steps can only be part of Aggregate Groups and can't be used as standalone elements in an expression set version.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Note** If you use a business element in an aggregation group, then pass the output of the aggregate step in the group as input to the business element.

## Business Element

Business elements perform the action that's defined in the element. Each business element can have a specific purpose based on your cloud's use cases. If a business element is available in your Salesforce org and is specific to the usage type of your expression set, then you can use it as a step in your expression set version.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Note** You can't create business elements or customize the ones available in your Salesforce org. When business elements are available in your Salesforce org, you see them in the Elements panel in Expression Set Builder with the other step elements.

**Tip** If you use a business element in an aggregate group, then pass the output of the aggregate step in the group as input to the business element.

# Expression Set Templates

An expression set template is a file-based expression set with prebuild a new expression set or an expression set version.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

To see the expression set templates available in your Salesforce org, from the Business Rules Engine app, select **Expression Set Templates**.

An expression set template can be of one of these types:

| Type | Description |
|---|---|
| Executable template | Executable expression set templates are ready-to-use, active expression sets. You can directly call the expression sets from flows or use them as subexpressions. The context definitions used in executable templates are also in the org in an active state.<br><br>Executable expression set templates can't use object field aliases or have lookup table steps. Steps in executable expression sets also can't reference explainability message templates.<br><br>You can't directly modify an executable expression set template. To make changes, save the template as an expression set or a version of an expression set. |
| Read-only template | Read-only expression set templates are inactive and can't be used in subexpressions, flows, or other calling processes. You can use a read-only template as a reference to build your own expression set version.<br><br>To reuse or modify a ready-only template, save it as an expression set. The steps from the template are copied to the first version of the expression set by default. The new expression set appears in the expression sets list view. |

# Create An Expression Set

Creating an expression set is a multistep process. When you create an expression set, the set's first version is created by default. Then, you configure the version and create variables and constants for inputs, outputs, and values that are used by and passed between the steps. Next, you add a logical sequence of steps that perform calculations or evaluate conditions using variables, constants, and decision matrices. After you create the flow, you simulate your expression set version with test values to verify that the set is working as expected, and then activate the expression set. To update an expression set version, save it as a version in a new expression set or as a new version within the existing expression set.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Create and Configure Your Expression Set Version**
When you create an expression set, the first version is created automatically and suffixed with V1. An expression set version is a blank and usable version of your expression set. Configure variables and constants in your expression set and then create steps that call decision matrices or decision tables and perform calculations or evaluate conditions.

**Simulate and Activate Your Expression Set Version**
Before you activate your expression set, run simulations with default and custom test input variables. Different decision matrices or subexpression versions called in an expression set version can have different start date time and end date time values. Test different versions by changing the Effective Date value. If the expression set doesn't work as expected, edit the elements and resimulate. When you're satisfied, activate the expression set version. If the expression set version is related to a context definition, you can simulate it by using the context mappings available in the context definition.

**Build Your Expression Set From a Template**
To reuse or modify an expression set template's components, save the template as an expression set or a version of an expression set. The template's type determines whether you can save it as a new expression set or a version of an existing expression set.

## Create and Configure Your Expression Set Version

When you create an expression set, the first version is created automatically and suffixed with V1. An expression set version is a blank and usable version of your expression set. Configure variables and constants in your expression set and then create steps that call decision matrices or decision tables and perform calculations or evaluate conditions.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete expression sets | Rules Engine Designer |
| To run expression sets in Business Rules Engine | Rules Engine Runtime |

**Important** Create and activate all decision matrices, decision tables, or sub expressions that you

plan to call in your expression set.

Create an expression set named StudentDiscountCalculation. This expression set calculates the fee for students based on the values that they enter in a form and the eligibility for discounted fees defined in our decision matrix.

**Note** Ensure that the name of an expression set and the expression set version starts with a letter and doesn't exceed 75 characters, including spaces. Don't start names with numbers and special characters.

1. From App Launcher ( ), find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Expression Sets**.
3. Click **New**.
4. For name, enter `StudentDiscountCalculation`.
5. Select a usage type, and save the expression set.
   The expression set's first version is created by default.
6. If needed, select **Initiatlize resources with default values**.
   If this checkbox isn't selected, expression set version resources are initialized with null values.
7. To open the version in Expression Set Builder, from the Expression Set Versions area, select **StudentDiscountCalculation V1**.
8. Set the version's properties:
   a. Click , and then enter a start date time and an end date time for the expression set version.
   b. Enter a rank number.

      **Note** When more than one enabled versions match an expression set call, and the date-time periods overlap, the version with the highest rank is chosen. For example, if two enabled versions have rank values set to 1 and 2, the version with rank 2 is chosen.

   c. Close the Expression Set Properties panel.
9. Create resources to use in the version steps:
   a. Click .
   b. Click **Add Resource**, and then create these resources.

      **Note** After adding a resource, to add the next resource, click **Done and New**. After adding all required resources, click **Done**.

Variables

| Variable NAME | Data Type | Decimal |
|---|---|---|
| StudentBirthdate | Date | 0 |
| StudentAge | Number | 0 |
| GPA | Number | 2 |
| QuoteAmount | Currency | 2 |
| QuoteAmountAdjusted | Currency | 2 |

Constants

| Constant Name | Data Type | Decimal | Default Value |
|---|---|---|---|
| Discount | Percent | 0 | 15 |
| HundredPercent | Percent | 0 | 100 |

10. To calculate the age of a student based on their birth date, add a Calculation step to the version.
    a. Click ⊕ in the expression set canvas, and then select **Calculation**.
    b. In the Formula field, search for and select **AGE (Date)**.
    c. Replace Date in Age (Date) with `StudentBirthdate` so that the Formula looks like this: `Age(Birthdate)`.
    d. In the Output Variable field, search for and select **StudentAge**.
11. To look up the discount eligibility of a student based on a combination of age and GPA, add a decision matrix:
    a. Click ⊕ in the expression set canvas, and then add a Decision Matrix element after the Calculation step.
    b. Search for and select **StudentDiscountEligibility**.
    The variables and constants from the decision matrix appear automatically.

12. To calculate the quoted fee, add another Calculation step:
    a. Click ⊕ in the expression set canvas, and then add the Calculation element after the Decision Matrix step.
    b. In the Formula field, search for and select **QuoteAmount**.
    c. In the Output Variable field, search for and select **QuoteAmountAdjusted**.
13. To evaluate whether a student is eligible for a discount, and then to calculate the discounted fee, add a Conditional Group step:
    a. Click ⊕ in the expression set canvas, and then add the Conditional Group element after the Calculation step.
    b. In the Condition element that's part of the Conditional Group, for Resource search for and select **StudentDiscountEligibility__Qualified**.
    c. For operator, select **Equals**.
    d. For Value, select **true**.
    e. Click ⊕ in the Conditional Group, and then add the Calculation element within the conditional group.
    f. In the Formula field, search for and select this series of variables and operators: **QuoteAmount * (HundredPercent – Discount)**.
    g. Select the **Include in Output**checkbox.

    **Important** The Conditional Group step runs only if the condition in the group evaluates to true.
14. Save the version.

See Also

*Salesforce Help*: Supported Data Types in Expression Sets

*Salesforce Help*: Operators and Functions in Expression Sets
*Salesforce Help*: Step Elements to Build Your Expression Sets
*Salesforce Help*: Considerations for Calling Expression Sets

## Simulate and Activate Your Expression Set Version

Before you activate your expression set, run simulations with default and custom test input variables. Different decision matrices or subexpression versions called in an expression set version can have different start date time and end date time values. Test different versions by changing the Effective Date value. If the expression set doesn't work as expected, edit the elements and resimulate. When you're satisfied, activate the expression set version. If the expression set version is related to a context definition, you can simulate it by using the context mappings available in the context definition.

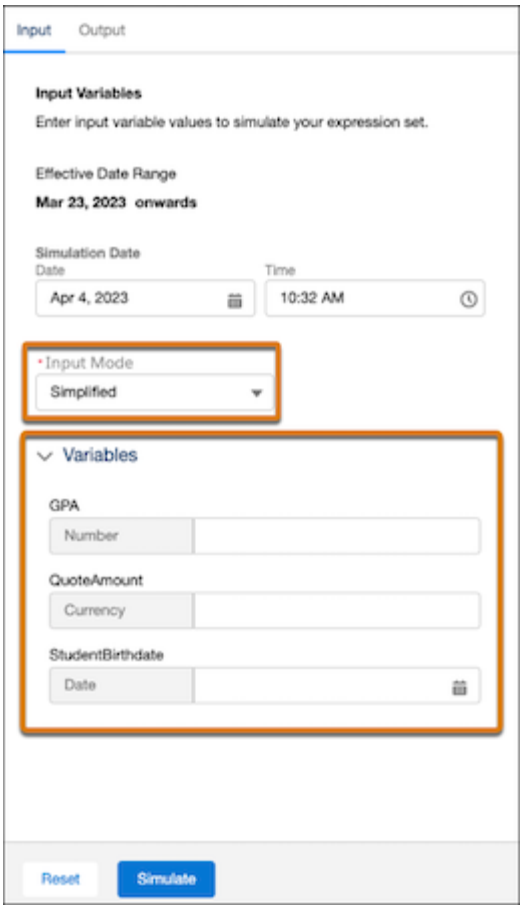
REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

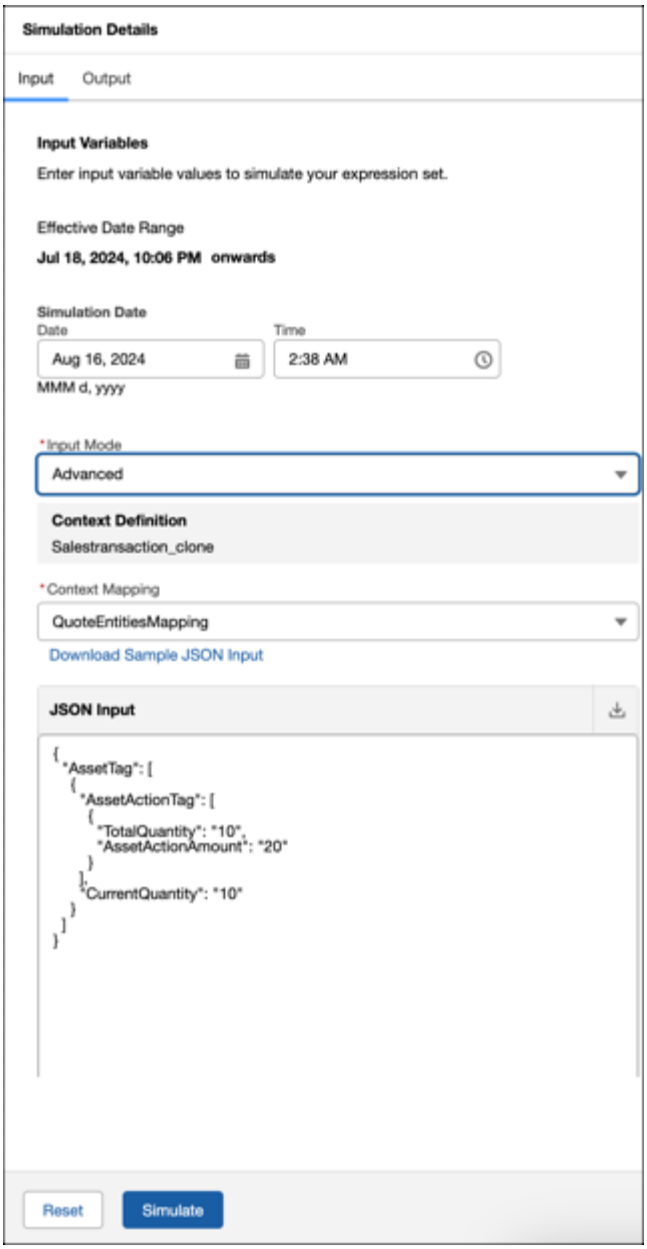
| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete expression sets: | Rule Engine Designer |
| To use expression sets in Business Rules Engine: | Rule Engine Runtime |

Simulate the StudentDiscountCalculation expression set that we created to test if the expression set is working as intended, and then activate the set.

1. Open your expression set version in Expression Set Builder.
2. Click **Simulate**.
   The Input Mode dropdown displays the options for providing sample simulation data to the version.
3. To enter sample values for the expression set version variables in the Variables section, select **Simplified** Input Mode.




4. To enter sample values for the expression set variables in the JSON Input text box, select **Advanced** Input Mode.




a. Change the selection to a mapping of your choice.
   The Context Mappings dropdown lists the available options if the expression set is linked to a context definition. The default context mapping is selected.
b. Enter sample values for the variables in the JSON Input text box.
   Alternatively, download the sample JSON input file, modify the file with your input values, and paste the updated JSON inputs in the textbox.

5. Simulate the expression set.
   Here's how the simulation results look in Detailed View, which is the default simulation view.

**Tip** To see how the results appear to your users, switch to Concise View. Concise View shows the explanations for version steps where you added explainability message templates.

6. For comprehensive testing of the expression set, enter different variable values, change the mapping,

and resimulate.

7. If you're happy with the simulation result, activate the expression set version.

See Also

*Salesforce Help*: Considerations for Calling Expression Sets
*Salesforce Help*: Invoke an Expression Set in a Flow
*Salesforce Help*: Call an Expression Set from an Integration Procedure
*Salesforce Help*: Call an Expression Set from an OmniScript

## Build Your Expression Set From a Template

To reuse or modify an expression set template's components, save the template as an expression set or a version of an expression set. The template's type determines whether you can save it as a new expression set or a version of an existing expression set.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To view an expression set template: | Rule Engine Designer |
| To save an expression set template as an expression set or an expression set version: | Rule Engine Designer |

## Build Your Expression Set from a Read-Only Template

You can't directly use a read-only expression set template because the template is inactive. To activate the template for reuse or to modify the template, save the template as an expression set.

**Note** To use an expression set template as-is for creating an expression set, ensure that all

1. From the App Launcher ( ), find and select **Business Rules Engine**.
2. From the app navigation menu, select **Expression Set Templates**.
3. Open the read-only expression set template that you want to use.

   The names of read-only templates don't have the Ready_to_Use prefix (1), and the templates don't have the Executable checkbox selected (2).

4. Click **Save As**.
   The Save As operation creates an expression set. The expression set's first version is also created by

default and opens in a new browser tab. The version inherits the template's components, such as resources and steps.

To call the expression set from flows or use it as a subexpression, give the version a rank, and activate the version.

## Build Your Expression Set or Version From an Executable Template

An executable template is active, so you can use it as-is in flows or as subexpressions. But you can't make changes or additions to an executable template. To reuse a template's components and then build on top of them, or to modify the components, save the template as an expression set or a version of an expression set.

1. From the App Launcher ( ), find and select **Business Rules Engine**.
2. From the app navigation menu, select **Expression Set Templates**.
3. Open the executable expression set template that you want to use.

   Executable templates have the Ready_to_Use prefix (1) in their names, and have the Executable checkbox selected (2).

4. Click **Save As**.
5. Select how you want to save the template.

| | |
|---|---|
| **Expression Set** | Saves the template as an expression set. |
| **Expression Set Version** | Saves the template as a version of the expression set that you select. |

6. Save your changes.
   Depending upon your selection, a version of the newly created or the selected expression set opens in a new browser tab.

# How to Create an Expression Set That Uses Context Definition

This example shows how to associate a context definition with an expression set, and how to use the definition's tags as list variables in the expression set version. The example also shows how to verify if the expression set version works as expected.

REQUIRED EDITIONS

Consider these recommendations when you design expression sets that use context definitions.

- To break down complex calculations into multiple steps, use transient attributes from the context definition to temporarily store interim calculation results in expression set steps. Transient attributes don't require mapping, and their values aren't saved in the database.

- You can save a version of a context-based expression set without explicitly marking any step as an output step, and manage the output by using the context mapping itself.
- Configure the decimal scale in the expression set version properties to correctly round off the results of the calculations that involve context variables.
- Select a context definition that does not have a reference context definition.

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for Industries clouds that have Business Rules Engine and Context Service enabled |

To encourage writers to publish their blog posts on their platform, a new blogging platform is offering an introductory, short-term income opportunity. Blog writers are entitled to an income based on the number of views, comments, and likes for their blog posts. Each blog post published on the platform is entitled to a flat income of $10. In addition, the writers are entitled to $0.20 per like and comment, and $0.10 per view.

The platform's marketing team asks their rule designer and Salesforce admin to create an automated rule that instantly shows the writers their earnings in real time.

The rule designer and the platform's Salesforce admin divide the requirement into these tasks.

1. **Create a Context Definition to Define the Blog Data Structure**
   The definition has the complete set of information that's required to build the expression set to calculate the income based on the views, likes, and comments on each blog post.
2. **Map the Definition to Data Sources and Activate the Definition**
   After you create the context definition, map the definition's nodes and attributes to sObjects and their fields. The mapping feeds data into the definition.
3. **Create an Expression Set to Calculate Blogging Income**
   In this example, the expression set is associated with the BlogDetails context definition. When you open the expression set version, you can use the context definition's tags as list variables in the version's steps.

**See Also**

*Salesforce Help*: List Variables in Expression Sets

## Create a Context Definition to Define the Blog Data Structure

The definition has the complete set of information that's required to build the expression set to calculate the income based on the views, likes, and comments on each blog post.

REQUIRED EDITIONS

| USER PERMISSIONS NEEDED | |
|---|---|
| To create and activate a context definition: | Context Service Admin |

1. From Setup, in the Quick Find box, enter `Context Definitions`, and then select **Context Definitions**.
2. Click **New**.
3. For Name, enter `BlogDetails`.
4. Enter the effective start and end dates for the definition.
   To use a context definition in an expression set, enter a start date that's before the expression set's start date, and enter an end date that's after the expression set's end date.
5. Click **Next**.
6. To define the relationship between the definition's nodes, create the definition's structure.
   a. Add a top-level node named `Blog_Data`.
      This node represents all attributes related to blogging data, including the child nodes and their attributes.
   b. Add a child node named `Engagement_Stats`.

7. Click **Next**.
8. Add attributes to each node, select whether the attribute is used as input, output, or both, and select the data type for each attribute.
   Here are the attributes in the Blog_Data and Engagement_Stats nodes respectively.

9. Click **Next**, and add tags for each node and the node's attributes.
   Tags appear as list variables in versions of the expression sets that are associated with context definitions.

   Here are the tags for the nodes and their attributes.

10. Save the definition.

## Map the Definition to Data Sources and Activate the Definition

After you create the context definition, map the definition's nodes and attributes to sObjects and their fields. The mapping feeds data into the definition.

REQUIRED EDITIONS

| USER PERMISSIONS NEEDED | |
|---|---|
| To create and activate a context definition: | Context Service Admin |

In this example, we're mapping the BlogDetails context definition's nodes and attributes to the Blog and Blog Engagement custom objects, which we've already created. These objects have fields that are similar to the attributes in the context definition's nodes.

1. From Setup, in the Quick Find box, enter `Context Definitions`, and then select **Context Definitions**.
2. To open the BlogDetails definition to map the data, select the BlogDetails from the definitions list.
3. Select **Map Data**.
4. Click **Add Mapping**.
   The mapping page opens on a new tab.
5. Provide mapping details.
   a. For name, enter `Blog_Readership`.
   b. Enter a description.
   c. For Mapping Type, select **Automatic sObject mapping**.
   d. Select **Mark as Default**.
   e. Click **Map**.
6. Map the context definition's nodes and attributes to sObjects and their fields.
   a. Click **Select Objects**.
   b. Search for the objects.
      In this example, we selected Blog__c and Blog_Engagement__c.

      **Tip** If you can't find a custom object, search with the object's API name.

   c. Click **Done**.
   d. To map the nodes to sObjects, select a node and then select the object. This mapping forms a connection between the node and the object.
   e. To map attributes to sObject fields, select an attribute and then select the field.
      This mapping forms a connection between the attribute and the field.

      **Note** Always map the definition's nodes before mapping the attributes.

   f. Save the mapping.
7. Return to the context definitions list.
8. To activate the BlogDetails context definition, click for the definition, and select **Activate**.

Your context definition is now ready to be used in the expression set.

## Create an Expression Set to Calculate Blogging Income

In this example, the expression set is associated with the BlogDetails context definition. When you open the expression set version, you can use the context definition's tags as list variables in the version's steps.

REQUIRED EDITIONS

| USER PERMISSIONS NEEDED | |
|---|---|
| To create an expression set that uses a context definition: | Rule Engine |

| USER PERMISSIONS NEEDED | |
|---|---|
| To use the definition's tags as list variables in the expression set's version steps: | Rule Engine |

1. From the App Launcher ( ), find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Expression Sets**.
3. Click **New**.
4. Create the expression set.
   a. For name, enter `Calculate_Blogging_Income`.
   b. Select **Default** as the usage type.
   c. For context definition, select **BlogDetails**.
   d. Save the expression set.
5. From the Expression Set Versions area, select **Calculate_Blogging_Income V1**.
   The version opens in Expression Set Builder.
6. To see the resources available for use in the version, click .

   You can use these context definition's tags as list variables in the steps.

7. Create the additional resources to be used in the version.
   a. Click **Add Resource**.
   b. For the first resource, specify these details, and click **Done and New**.
      Resource Type: Constant, Resource Name: Blog_Flat_Rate, Data Type: Currency, Default Value: 10.
   c. For the second resource, specify these details, and click **Done**.
      Resource Type: Variable, Resource Name: Blog_Income, Data Type: Currency
8. Set the properties of the expression set version.
   a. To open the Expression Set Properties panel, click .
   b. Enter a start date time and an end date time for the expression set version.
   c. Enter a rank number.
   d. Enter a decimal scale.
      The decimal scale represents the number of decimal places to round off the calculation step results that involve context variables to the required precision.
   e. Close the Expression Set Properties panel.
9. On the builder canvas, add a step to calculate the number of likes per blog post by the platform's subscribers.
   a. Click , and select **Calculation**.
   b. In the Formula field, search for and select **LISTSUM**.
   c. Replace ListVariable inside the parenthesis with **Like_Count**.
      Like_Count is an attribute tag from the Blog_Data context definition. The tag is used as a list variable in the calculation and is part of the Engagement_Stats node in the BlogDetails context definition.
   d. For Output Variable, select **Likes**.
      Likes is an attribute tag under the Blog_Data node in the BlogDetails context definition.

   Here, Like_Count is lower in the context definition's node hierarchy than Likes. When the LISTSUM

function sums the number of likes across subscribers per blog post, the sum is stored in the Likes variable, which is above Like_Count in the node hierarchy.

10. Calculate the number of comments per blog post by the platform's subscribers.
    a. Click ⊕, and select **Calculation**.
    b. In the Formula field, search for and select **LISTSUM**.
    c. Replace ListVariable inside the parenthesis with `Comment_Count`.
       Comment_Count is an attribute tag from the Blog_Data node. The tag is used as a list variable in the calculation and is part of the Engagement_Stats node in the BlogDetails context definition.
    d. For Output Variable, select **Comments**.
       Comments is an attribute tag under the Blog_Data node in the BlogDetails context definition.
11. Add the condition logic to filter blogs based on their total views, likes, and comments.
    a. Click ⊕, and select **List Group**.
    b. In the list filter, for Resource, select **Views**, for Operator, select **Greater than or Equal**, and for Value, enter `100`.
       This expression represents the condition, Views are greater than or equal to 100.
    c. To add a condition, click **Add Condition**.
    d. For Resource, select **Likes**, for Operator, select **Greater than or Equal**, and for Value, enter `10`.
       This expression represents the condition, Likes are greater than or equal to 10.
    e. To add another condition, click **Add Condition**.
    f. For Resource, select **Comments**, for Operator, select **Greater than or Equal**, and for Value, enter `5`.
       This expression represents the condition, Comments are greater than or equal to 5.
    g. In the list filter, for Filter Condition Requirements, select **Custom Condition is Met**.
    h. For Custom Condition Logic, enter `1 AND (2 OR 3)`.

       The custom logic indicates that the list filter's conditions evaluate to true when the first condition and either the second or the third condition are met.

12. Calculate the blogging income for the filtered list items.
    a. In the list group, after the list filter, click ⊕.
    b. Select **Calculation**.
    c. In the Formula field, enter `Blog_Flat_Rate + ( 0.10 * Views ) + 0.20 * ( Likes + Comments ).`
       The formula represents how the blogging income is calculated. Each blog post earns the writer a flat blog rate of $10, which is stored in the Blog_Flat_Rate resource. The writer is entitled to $0.10 per view on the blog post. The income from the view is calculated by multiplying the number of views with 0.10. The writer is entitled to an additional income of $0.20 per like and comment on the blog post. The income from the likes and comments is calculated by adding the number of comments and likes per post, and multiplying the number by 0.20. The income from the likes, views, and comments is then added to the blog flat rate.
    d. In the Output Variable field, select **Blog_Income**.
       The Blog_Income variable stores the output of the calculation formula for the total blog income.
13. To see a version step's output when the expression set version runs, select the step element, click ⋮, and select **Include in output**.

14. Save the version.
15. To verify that the steps work as expected, simulate the version.
    a. Click **Simulate**.
    b. For Input Mode, select **Advanced**.
    When you use list variables in an expression set, always use the Advanced input mode to pass sample values for simulation.
    c. Modify the JSON input template with these sample values.

    d. Click **Simulate**.

    The list filter returns one matching result based on the filter criteria.

    The calculation step in the list group returns the blogging income for the blog post that meets the filter conditions.

# Manage Expression Sets

Learn how to clone an expression set, open an expression set version in the builder, and deactivate an expression set version.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, edit, activate, and delete an expression set and its versions: | Rule Engine Designer |

### Clone an Expression Set

To copy an expression set along with all its versions, use the Clone option on the record detail page of an expression set. When you clone an expression set, all the cloned versions are in an inactive state.

### Open an Expression Set Version in Expression Set Builder

Open an expression set version in Expression Set Builder to view the version steps, modify the configuration, or simulate the version to test whether the version steps work as expected. You can modify only inactive expression set versions.

### Save an Expression Set Version in Expression Set Builder

Save an expression set version within a new expression set or as a new version within the existing expression set. This allows you to make updates without cloning the entire set and its versions, and

customize the name and version number for easier identification.

### Deactivate an Expression Set Version

Active expression set versions can't be edited. To edit an expression set version, deactivate the version first.

## Clone an Expression Set

To copy an expression set along with all its versions, use the Clone option on the record detail page of an expression set. When you clone an expression set, all the cloned versions are in an inactive state.

1. Click , and then find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Expression Sets**.
3. Select an expression set from the list.
4. On the record detail page of the selected expression set, click **Clone**.
5. To prevent naming conflict, enter a unique name for the cloned expression set.

   **Warning** To retain the object field aliases or business elements associated with the expression set's usage type, don't change the usage type while cloning the expression set.

6. Save the expression set.

## Open an Expression Set Version in Expression Set Builder

Open an expression set version in Expression Set Builder to view the version steps, modify the configuration, or simulate the version to test whether the version steps work as expected. You can modify only inactive expression set versions.

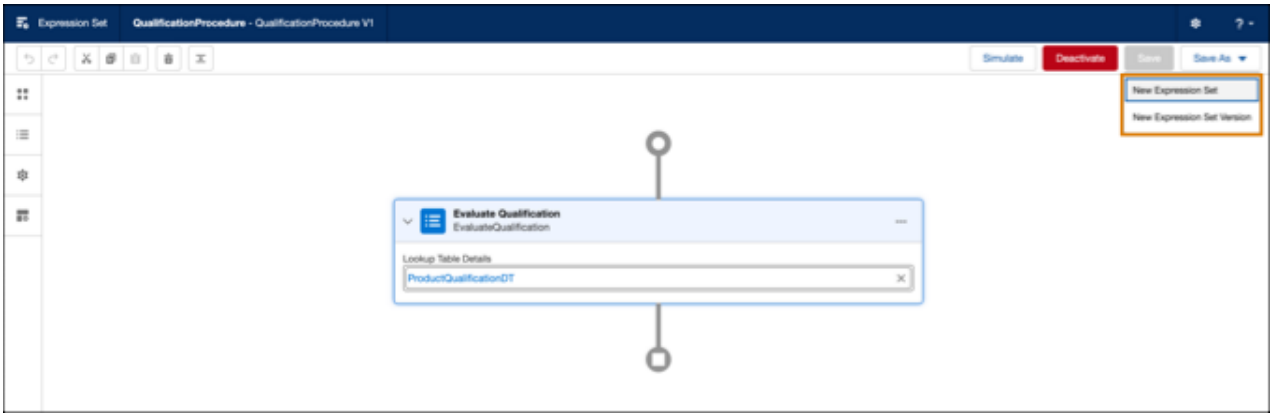
1. Click , and then find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Expression Sets**.
3. Select an expression set from the list.
4. On the Details tab, in the Expression Set Versions area, select the version.
   The version opens in the expression set builder.

## Save an Expression Set Version in Expression Set Builder

Save an expression set version within a new expression set or as a new version within the existing expression set. This allows you to make updates without cloning the entire set and its versions, and customize the name and version number for easier identification.

1. Click , and then find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Expression Sets**.
3. Select an expression set from the list.
4. On the Details tab, in the Expression Set Versions area, select the version.
   The version opens in the expression set builder.
5. Click **Save As**.
   You can either save the version in a new expression set or as a new version in the current expression

set.




6. To save the selected version in a new expression set, click **New Expression Set**.
   a. Enter the expression set name.
   b. Select the context definition.
   c. Enter the rank and select the activate checkbox.
   d. Save the selected expression set version in the new expression set.
      The expression set version is now ready to be invoked.
   e. To make changes, deactivate the expression set version, make the desired changes, and activate it again.
7. To save the selected version as a new version in the same expression set, click **New Expression Set Version**.
   a. Enter the expression set version name.
   b. Enter the expression set version number.
   c. Save the new expression set version in the current expression set.
   d. Make the desired changes in the expression set version and activate it.

## Deactivate an Expression Set Version

Active expression set versions can't be edited. To edit an expression set version, deactivate the version first.

1. From the app navigation menu of Business Rules Engine, select **Expression Sets**.
2. From the expression set list, click the expression set that has the version that you want to deactivate.
3. On the Details tab, in the Expression Set Versions area, select the version.
   The version opens in the expression set builder.
4. Click **Deactivate**.

# Considerations for Migrating Expression Sets

Generally, expression sets are migrated to production orgs after they're tested in the development and staging sandbox orgs, and verified to work as expected. Expression sets can contain references to dependent elements such as decision matrices, decision tables, subexpressions, context definitions, field and object aliases, and explainability message templates.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Here are some considerations to keep in mind when you plan the migration of expression set and expression set versions by using Salesforce CLI (SF CLI), Workbench, Change Sets, or any other migration tool.

**Note** Although SF CLI supports migrating complete expression sets and decision matrices, it doesn't support migrating individual versions. To deploy a single expression set version, first retrieve it via a package.xml manifest file, and then deploy it using another tool apart from SF CLI. See Sample package.xml Manifest Files.

## Scenario 1: Migrate a new expression set

If the target org doesn't contain the expression set that you want to migrate, deploy the expression set along with its versions from the source org to the target org. If you specified ranks on the expression set versions in the source org, they operate at the same ranks after the migration. If the expression set contains dependencies, follow the recommendations in Scenario 3: Migrate expression set versions with dependencies.

## Scenario 2: Migrate updates to an expression set

To update an expression set that exists in the target org, migrate a new expression set version with the updates; don't update the existing active versions.

You can create an expression set version in the source org with the updates. Specify a rank and activate the new version in the source org. Then, migrate the new version or the entire expression set to the target org. The migrated expression set version is invoked in the target org as per the specified rank. Migration fails if the target org already contains an expression set version with the specified rank.

If the expression set contains dependencies, follow the recommendations in Scenario 3: Migrate expression set versions with dependencies.

Consider these points when you migrate an expression set or an expression set version.

- The migration result depends on whether the expression set and its versions exist in the target org, and on the status of such versions.
- Expression set versions are migrated to the target org in the same state as in their source org.
- Expression set versions in the target org aren't deleted irrespective of their state. Manually delete expression set versions that you don't require in the target org.
- When you migrate an expression set version that isn't in the target org, it's deployed from the source

org to the target org.
- When you migrate an expression set version that's inactive in the target org, it's replaced by the version in the source org. The migrated expression set version from the source org overwrites the inactive expression set version in the target org.
- When you migrate a modified expression set version that's active in the target org, it fails and the version in the source org doesn't overwrite the active version in the target org.
- If both the source and target orgs have a version with the same developer name, regardless of its active status, make sure that the version number in the target org remains unchanged and matches the version number in the source org.

Let's review these considerations with an example of an expression set named FeeCalculator used by an educational institution. FeeCalculator version, v1 is used to calculate the fees of a student based on the selected course, entrance examination rank, and application category in the target org. Now, the institution wants to include discounts based on the calculated fees. Accordingly, the rule designer creates another expression set version, v2 in the source org to incorporate the new criteria, and tests the version with sample data. The rule designer either migrates the entire expression set or only version v2 to the target org. The migration behavior depends on the status of the FeeCalculator version in the target org.

| Target Org Before Migration | Source Org Before Migration | Migration status | Target Org After Migration |
|---|---|---|---|
| Active v1 | • Active v1 (Same as in target org)<br>• Inactive v2 | Successful<br>• v1 in the target org is retained.<br>• v2 gets deployed from the source org to the target org. | • Active v1<br>• Inactive v2 |
| Active v1 | • Active v1 (Same as in target org)<br>• Active v2 | Successful<br>• v1 in the target org is retained.<br>• v2 gets deployed from the source org to the target org. | • Active v1<br>• Active v2 |
| • Active v1<br>• Inactive v2 | • Active v1 (Same as in target org)<br>• Active v2 | Successful<br>• v1 in the target org is retained.<br>• v2 from the source org overwrites the inactive v2 in the target org. | • Active v1<br>• Active v2 |
| Active v2 | Active v2 (Different from v2 in target org) | Failure<br>• v2 from the source org doesn't | Active v2 (As was in the target org before migration) |

| Target Org Before Migration | Source Org Before Migration | Migration status | Target Org After Migration |
|---|---|---|---|
| | | overwrite the active v2 in the target org. v2 in the target org is retained. | |
| Active v1 | • Active v1 (Different from v1 in the target org)<br>• Active v2 | Failure<br>• v1 in the target org is retained.<br>• v2 isn't deployed from the source org to the target org. | Active v1 (As was in the target org before migration) |
| | | Recommended approach: Deactivate v1 in the target org and migrate v1 and v2.<br>• v1 is migrated from the source org to the target org.<br>• v2 in the target org is retained. | • Active v1 (As was in the source org)<br>• Active v2 |

**Note** When input or output variables of an expression set change, it results in a contract-breaking change. To migrate this expression set, clone the expression set heirarchy as needed, apply the necessary updates, and deploy the expression set to the target org. Finally, update the invocation to reference the new expression set.

## Scenario 3: Migrate expression set version with dependencies

Expression set versions can reference dependent elements such as decision matrices, decision tables, subexpressions, context definitions, field and object aliases, and explainability message templates. In such cases, migrate the dependent elements independently before you migrate the expression set version itself. You can deploy an expression set along with its immediate dependencies simultaneously. If those dependencies contain further nested dependencies, migrate the lowest-level dependencies before you deploy the parent expression set. For more recommendations and examples of expression set versions with dependencies and contarct-breaking changes, see How to Migrate Expression Sets with Dependencies.

### Expression Set Migration By Using Salesforce CLI

Use Salesforce CLI (SF CLI) to migrate the Business Rules Engine components - such as expression sets, decision matrices, decision tables, and their dependencies - from one org to another.

### Migration of Expression Sets with Dependencies

Expression set versions can reference elements such as decision matrices, decision tables,

subexpressions, context definitions, field and object aliases, and explainability message templates. Migrate the dependent elements independently before you migrate the expression set version.

## Expression Set Migration By Using Salesforce CLI

Use Salesforce CLI (SF CLI) to migrate the Business Rules Engine components - such as expression sets, decision matrices, decision tables, and their dependencies - from one org to another.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Note** Although SF CLI supports migrating complete expression sets and decision matrices, it doesn't support migrating individual versions. To deploy a single expression set version, first retrieve it via a package.xml manifest file, and then deploy it using another tool apart from SF CLI. See Sample package.xml Manifest Files.

Let's consider an insurance business rule implemented in an expression set with the API name, `InsuranceES1`. This expression set calls a decision table ( `InsuranceDT1` ), and a decision matrix ( `InsuranceDM1` ) to handle parts of the business logic. After you test the components in a scratch org, they're ready to be migrated to the production org or the target org.

Here are sample commands to query the object tables for the component API names from a scratch org named ScratchOrgQ1.

```
sf data query --query "SELECT DeveloperName FROM ExpressionSetDefinition where
MasterLabel = 'Expression Set for Insurance'"  --target-org ScratchOrgQ1
```

```
sf data query --query "SELECT DeveloperName FROM DecisionMatrixDefinition wher
e MasterLabel = 'Decision Matrix for Insurance'"  --target-org ScratchOrgQ1
```

```
sf data query --query "SELECT DeveloperName FROM DecisionTable where SetupName
= 'Decision Table for Insurance'"  --target-org ScratchOrgQ1
```

Ensure that you've installed and configured your SF CLI and connected to your orgs as described in the Salesforce DX Developer Guide. To migrate the components, first retrieve them from the scratch org and then deploy them to the target org.

## Retrieve Components from Scratch Org

To retrieve the metadata of the decision table, decision matrix, and expression set from the scratch org, run these commands:

```
sf project retrieve start --metadata ExpressionSetDefinition:<Expression Set Entity API Name> --target-org <Scratch Org Name>
```

```
sf project retrieve start --metadata DecisionMatrixDefinition:<Decision Matrix Entity API Name> --target-org <Scratch Org Name>
```

```
sf project retrieve start --metadata DecisionTableDefinition:<Decision Table Entity API Name> --target-org <Scratch Org Name>
```

For the latest version of the SF CLI commands listed here, see Salesforce CLI Command Reference.

Here are the commands to retrieve the components for the insurance business rule example from the scratch org we considered:

```
sf project retrieve start --metadata ExpressionSetDefinition:InsuranceES1 --target-org ScratchOrgQ1
```

```
sf project retrieve start --metadata DecisionMatrixDefinition:InsuranceDM1 --target-org ScratchOrgQ1
```

```
sf project retrieve start --metadata DecisionTableDefinition:InsuranceDT1 --target-org ScratchOrgQ1
```

Folders InsuranceES1, InsuranceDM1, and InsuranceDT1 that contain the metadata.xml of the expression set, decision matrix and decision table respectively are created under the force-app/main/default path.

## Deploy Components to Target Org

Deploy the dependent elements independently before you deploy the expression set version itself. Run these commands to deploy the metadata of the decision matrix and decision table to the target org.

```
sf project deploy start --metadata DecisionTableDefinition:<Decision Table API Name> --target-org <Target Org Name>
```

```
sf project deploy start --metadata DecisionMatrixDefinition:<Decision Matrix API Name> --target-org <Target Org Name>
```

To deploy the expression set, run this command:

```
sf project deploy start --metadata ExpressionSetDefinition:<Expression Set API Name> --target-org <Target Org Name>
```

Here are the commands to deploy the components for the example we considered to a target org named TargetOrgQ1:

```
sf project deploy start --metadata DecisionTableDefinition:InsuranceDT1 --target-org TargetOrgQ1
```

```
sf project deploy start --metadata DecisionMatrixDefinition:InsuranceDM1 --target-org TargetOrgQ1
```

```
sf project deploy start --metadata ExpressionSetDefinition:InsuranceES1 --target-org TargetOrgQ1
```

After you run this command, you receive a response with the details of the deployment. Now the decision matrix, decision table, and expression set are deployed to the target org.

Migrating a decision matrix or decision table deploys the metadata of the component and not the data. Import the data into the component after the migration is complete and then activate the component.

- After you migrate a decision matrix without row data, open the decision matrix version in the target org, and then upload the row data by using the Upload CSV File option.
- After you migrate the decision table without row data, refresh the decision table with the latest data in the object that it refers to.

For more information on various migration scenarios and how to handle them, see Considerations for Migrating Expression Sets.

## Migration of Expression Sets with Dependencies

Expression set versions can reference elements such as decision matrices, decision tables, subexpressions, context definitions, field and object aliases, and explainability message templates. Migrate the dependent elements independently before you migrate the expression set version.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

This image depicts the dependencies among components typically used in expression sets. To migrate these components, start from the independent components in Level 1 of the Deployment Order and move up along the order.

Consider these recommendations.

- You can deploy an expression set along with its immediate dependencies simultaneously. If those dependencies contain further nested dependencies, then migrate the lowest-level dependencies before you deploy the parent expression set.
- Migrate the lookup tables before you migrate the expression set and subexpression versions.
- If the expression set version contains a context definition, then migrate the context mapping, and context definition independently. Activate the definition, and then migrate the expression set.
- If the expression set version uses field aliases for the fields on custom objects or the custom fields on standard objects, then make sure that you migrate the dependencies before you migrate the expression set version.
- If the expression set contains a decision table that uses the fields on a custom object or the custom fields on a standard object, then make sure that you migrate the dependencies before you migrate the decision table, and then migrate the expression set that contains the decision table.

- Deactivate lookup tables, and then migrate them. The migration of a lookup table deploys the metadata of the lookup table and not the data. Import data into the lookup tables after the migration is complete and then activate the lookup tables.
  - After you migrate a decision matrix without row data, open the decision matrix version in the target org, and then upload the row data by using the Upload CSV File option.
  - After you migrate a decision table without row data, refresh the decision table with the latest data in the object that it refers to.

Here are a few dependency scenarios in expression sets and the recommended migration steps.

| Use case | Components | Scenario | Migration Steps |
|---|---|---|---|
| Expression set with two levels | • Parent expression set - PES<br>– Expression set - ES | PES calls ES. | Deploy PES and ES together. |
| Expression set with more than two levels | • Parent expression set - PES<br>– Expression set - ES<br>– Child expression set - CES | • PES calls ES.<br>• ES calls CES. | • Deploy CES.<br>• Deploy ES and PES together. |
| Multi-level expression set with decision table | • Parent expression set - PES<br>– Expression set - ES<br>– Child expression set - CES<br>– Decision table - DT<br>– Custom Object - CO | • PES calls ES.<br>• ES calls CES.<br>• CES imports DT.<br>• DT is based on CO. | • Deploy and populate CO.<br>• Deploy and activate DT.<br>• Deploy CES.<br>• Deploy ES and PES together. |
| Multi-level expression set with context definition | • Parent expression set - PES<br>– Expression set - ES | • PES calls ES.<br>• ES calls CES. | • Deploy CD and CM, activate CD. |

| Use case | Components | Scenario | Migration Steps |
|---|---|---|---|
| | - Child expression set - CES<br>  - Context Definition - CD<br>  - Context Mapping - CM | • CES uses CD.<br>• CD is based on CM. | • Deploy CES.<br>• Deploy ES and PES together. |

When the input or output variables in a new expression set version are changed or deleted, it constitutes a contract-breaking change. You can't migrate an expression set version with a contract-breaking change directly. However, you can save it as a new expression set, and migrate the new expression set. These examples illustrate this concept.

| Use case | Components | Changes to components | Migration Steps |
|---|---|---|---|
| Expression set version without contract-breaking change | • Parent expression set - PES<br>  - Expression set - ES1 with version EV1<br><br>EV1 variables:<br><br>• Input - IV1, IV2<br>• Output - OV1, OV2<br><br>PES calls EV1 in the source org.<br><br>Migrate PES and ES1 with EV1 to the target org. | In the source org, create EV2 in ES1 with the required changes.<br><br>EV2 has the same variables as EV1.<br><br>• Input - IV1, IV2<br>• Output - OV1, OV2 | Activate, rank, and deploy EV2 to target org.<br><br>PES calls EV2 of ES1. |
| Expression set version with contract-breaking change | • Parent expression set - PES<br>  - Expression set - ES1 with version EV1<br><br>EV1 variables:<br><br>• Input - IV1, IV2<br>• Output - OV1, OV2<br><br>PES calls EV1 in the source org. Migrate PES and ES1 with EV1 to the target org. | In the source org, create EV2 in ES1 with the required changes.<br><br>EV2 variables are different from EV1 variables.<br><br>• Input - IV1, IV2, IV3<br>• Output - OV1 | Migration of EV2 to the target org fails.<br><br>Here are the recommended migration steps.<br><br>• In the source org, save EV2 as a new expression set ES2. ES2 has version EV2 with the updated logic. |

| Use case | Components | Changes to components | Migration Steps |
| --- | --- | --- | --- |
| | | | • Migrate ES2 to the target org.<br>• Update PES to call ES2 instead of ES1 in the target org. |

## Expression Sets for Developers

Developers in your Salesforce org can employ expression set developer resources to use and extend expression sets.

REQUIRED EDITIONS

| |
| --- |
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

### Connect REST API

- Create an expression set
- Update an expression set
- Retrieve or delete expression sets
- Invoke an expression set
- Fetch component guradrails

### Metadata API

- Create an expression set
- Create an expression set object field aliases to be used as variables in expression sets

### Invocable Actions

- Invoke an expression set in a flow using expression set action

# Decision Explainer

Decision Explainer is a service that provides insights into why a business rule generated a specific result.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Professional**, **Unlimited**, and **Developer** Editions where Decision Explainer is enabled |

Decision Explainer has setup objects that capture logs for how a rule or an expression set ran, making it easy for reviewers to explain the outcome of the rule to users. For example, a student wants to know why they received a lesser discount on their course fee than what they expected. With Decision Explainer, the student application reviewer can check the application's history and the expression set that's used to evaluate the application, and provide the student the reason for receiving a lesser discount.

**Create Application Subtype Definitions**
Create application subtype definitions to define the types of applications used in your Decision Explainer entities.

**Create Business Process Type Definitions**
Define the types of business processes that are applied to a rule.

**Create Explainability Action Definitions**
Define where the metadata of your business rules is stored.

**Create Explainability Action Versions**
Define and store versions of the explainability actions used by business rules that use Decision Explainer.

**Decision Explainer Default Limits**
To ensure optimal performance, consider these default limits while using Decision Explainer to retrieve action logs. If your business requires any default limits to be raised, contact Salesforce customer support for further guidance.

**Decision Explainer for Developers**
Developers in your Salesforce org can employ Decision Explainer developer resources to use and extend the Decision Explainer functionality.

**Decision Explainer for Expression Sets**
Use Decision Explainer with Business Rules Engine to show your users explanations for the results of steps in an expression set, and to capture action logs for expression set runs. Store explanations for expression steps in explainability message templates. And then add the templates to your expression set steps for which you want to show explanations. When you simulate the expression set version, switch to the Concise view option to preview how your users see the explanations.

# Create Application Subtype Definitions

Create application subtype definitions to define the types of applications used in your Decision Explainer entities.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Professional**, **Unlimited**, and **Developer** Editions where Decision Explainer is enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create application subtype definitions: | Decision Explainer Service Access |

1. From Setup, in the Quick Find box, enter `Decision Explainer`, and then select **Application Subtype Definition**.
2. Click **New Application Subtype Definition**.
3. Enter a label for the new definition and press tab.
   The developer name autopopulates. If the developer name exceeds 10 characters because your label is too long, you can either shorten the label or modify the autogenerated developer name. Try to keep the label and the developer name similar.
4. Optionally, enter a description.
5. Select an application usage type.
6. Save your changes.

## Create Business Process Type Definitions

Define the types of business processes that are applied to a rule.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Professional**, **Unlimited**, and **Developer** Editions where Decision Explainer is enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create business process type definitions: | Decision Explainer Service Access |

1. From Setup, in the Quick Find box, enter `Decision Explainer`, and then select **Business Process Type Definition**.
2. Click **Business Process Type Definition**.
3. Enter a label for the new definition and press tab.
   The developer name autopopulates. If the developer name exceeds 10 characters because your label is too long, you can either shorten the label or modify the autogenerated developer name. Try to keep the label and the developer name similar.
4. Optionally, enter a description.
5. Select an application usage type.

6. Save your changes.

## Create Explainability Action Definitions

Define where the metadata of your business rules is stored.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Professional**, **Unlimited**, and **Developer** Editions where Decision Explainer is enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create explainability action definitions: | Decision Explainer Service Access |

1. From Setup, in the Quick Find box, enter `Decision Explainer`, and then select **Explainability Action Definition**.
2. Click **New Explainability Action Definition**.
3. Enter a label for the new definition and press tab.
   The developer name autopopulates. If the developer name exceeds 10 characters because your label is too long, you can either shorten the label or modify the autogenerated developer name. Try to keep the label and the developer name similar.
4. For Business Process Type, click , and select the business process type for the new explainability action definition.
5. Select an application type for which you want to generate an explainability log.
6. For Action Log Schema Type, select the type of action log schema you want to use for the application. For example, select Expression Set. The default value is Other.
7. Optionally, enter a description.
8. For Application Subtype, click , and select an application subtype.
9. Save your changes.

## Create Explainability Action Versions

Define and store versions of the explainability actions used by business rules that use Decision Explainer.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Professional**, **Unlimited**, and **Developer** Editions where Decision Explainer is enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create explainability action versions: | Decision Explainer Service Access |

1. From Setup, in the Quick Find box, enter `Decision Explainer`, and then select **Explainability Action Version**.
2. Click **New Explainability Action Version**.
3. Enter a label for the version.
4. To make the version available for use, select **Active**.
5. For Explainability Action Definition, click , and select the explainability action definition for which you're creating a version.
6. Optionally, enter a description.
7. Save your changes.

# Decision Explainer Default Limits

To ensure optimal performance, consider these default limits while using Decision Explainer to retrieve action logs. If your business requires any default limits to be raised, contact Salesforce customer support for further guidance.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Professional**, **Unlimited**, and **Developer** Editions where Decision Explainer is enabled |

What is a Context Definition?

- You can invoke up to 500,000 action logs per hour.
- One GET call to Decision Explainer can read up to 500 logs.

# Decision Explainer for Developers

Developers in your Salesforce org can employ Decision Explainer developer resources to use and extend the Decision Explainer functionality.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions where Decision Explainer is enabled |

## Connect REST API

- Create and retrieve explainability action logs using Connect REST API

## Metadata API

- Create an application subtype definition
- Create a business process type definition
- Create an explainability action definition
- Create an explainability action version

# Decision Explainer for Expression Sets

Use Decision Explainer with Business Rules Engine to show your users explanations for the results of steps in an expression set, and to capture action logs for expression set runs. Store explanations for expression steps in explainability message templates. And then add the templates to your expression set steps for which you want to show explanations. When you simulate the expression set version, switch to the Concise view option to preview how your users see the explanations.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for Industries clouds that have Business Rules Engine and Decision Explainer enabled |

Decision Explainer is a service that provides insight into why a business rule generated a specific result. For example, a student wants to know why they received a lesser discount on their course fee than what they expected. With Decision Explainer, the student application reviewer can check the application's history and the expression set that's used to evaluate the application, and provide the student the reason for receiving a lesser discount.

**Create Expression Set Message Tokens**
Expression set message tokens serve as placeholders for specific values from an expression set version within an explainability message template. Without tokens, the explanation messages would be static and unable to reflect the specific data involved in a particular execution of the rule. Create a token, and then map it to actual resources to dynamically display relevant data in explanation messages.

**Create an Explainability Message Template**
Explainability message templates store explanation messages, and can be referenced in an expression set version step to explain the step's result. You can use expression set message tokens in your message templates as placeholders for expression set version resources. Rule designers can then map the tokens to the resources from the expression set version where the template is used. When the expression set runs, the explanations dynamically show the values of the mapped resources.

**Show Rule Explanations to Users**

When Decision Explainer is available in your Salesforce org, you can show your users the explanations for expression set version step results. The explanations are stored in explainability message templates that your admin creates and maintains.

**Disable Decision Explanations for Expression Set Version Steps**
To remove the explainability message templates assigned to an expression set version step, open the version in Expression Set Builder and deselect Show decision explanation for the step. You can disable decision explanations for expression set version steps only if the version is inactive.

**Show Rule Explanations to Users (Example)**
This example shows how you can configure an expression set version to show step-result explanations to external users. The example also shows how to preview the explanations that your users see.

**Compare Expression Set Runs with Decision Explainer**
To analyze why an expression set generated different outputs for each run, enable Decision Explainer for expression set versions, and then use the Compare option on the Decision Explainer Log History component. Decision Explainer logs are generated in the context of object records that an expression set is run for. To use the Compare option, the Decision Explainer Log History component must be enabled on the record page of the object within whose context the expression set is run.

**Explainability Message Templates for Developers**
Developers in your Salesforce org can use developer resources to use and extend explainability message templates.

## Create Expression Set Message Tokens

Expression set message tokens serve as placeholders for specific values from an expression set version within an explainability message template. Without tokens, the explanation messages would be static and unable to reflect the specific data involved in a particular execution of the rule. Create a token, and then map it to actual resources to dynamically display relevant data in explanation messages.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for Industries clouds that have Business Rules Engine and Decision Explainer enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create tokens: | Decision Explainer Service Access |
| To use tokens in explainability message templates: | Decision Explainer Service Access |
| To map tokens from an explainability message template to expression set version resources: | Rule Engine Designer |

1. From Setup, in the Quick Find box, enter `Expression Set Message Token`, and then select **Expression Set Message Token**.

2. Click **New Expression Set Message Token.**
3. Enter a name for the token.

   **Tip** Enter a unique name that begins with a letter and doesn't contain spaces, special characters, or consecutive underscores.

4. Enter a label.
5. Enter a description.
6. Save the expression set message token.

See Also

*Salesforce Help*: Create an Explainability Message Template
*Salesforce Help*: Show Rule Explanations to Users

## Create an Explainability Message Template

Explainability message templates store explanation messages, and can be referenced in an expression set version step to explain the step's result. You can use expression set message tokens in your message templates as placeholders for expression set version resources. Rule designers can then map the tokens to the resources from the expression set version where the template is used. When the expression set runs, the explanations dynamically show the values of the mapped resources.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for Industries clouds that have Business Rules Engine and Decision Explainer enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create and modify explainability message templates: | Decision Explainer Service Access |
| To use explainability message templates in expression set steps: | Rule Engine Designer |
| To run expression sets that have Decision Explainer enabled: | Rule Engine Runtime |

1. From Setup, in the Quick Find box, enter `Explainability Message Template`, and then select **Explainability Message Template**.
2. Click **New Explainability Message Template**.
3. Enter a name for the template.
   Enter a unique name that begins with a letter and doesn't contain spaces, special characters, or consecutive underscores.
4. Enter a label for the template.
5. In the Message field, enter a message for a combination of the expression set step type and the result

type.
To create a token-based message, enter a token name within {} as a placeholder for an expression set version resource.

**Note** A message can include up to five unique tokens.

Here, Resource1 and Resource2 are tokens. When the template is attached to a Calculation step with the Passed result type in an expression set version, rule designers can map the tokens to resources from the version.

6. Select the result type.
7. Select the expression set step type.
   The No Result result type applies only to the Decision Table Lookup and Decision Matrix Lookup step types.
8. Select the usage type.

   **Note** When rule designers search for the explainability message templates to be added to an expression set version step, they see only the templates that share the expression set's usage type and the templates that don't have a usage type.
9. To set the message as default for the expression set step type and result type, select **Default**.

   If a default template is available for an expression set version's step type and result type, the template is automatically selected for the step.

   **Note** One Salesforce instance can have only one default message for a combination of expression set step type and result type.
10. Save your explainability message template.

See Also
*Salesforce Help*: Show Rule Explanations to Users
*Salesforce Help*: Show Rule Explanations to Users (Example)

## Show Rule Explanations to Users

When Decision Explainer is available in your Salesforce org, you can show your users the explanations for expression set version step results. The explanations are stored in explainability message templates that your admin creates and maintains.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for Industries clouds that have Business Rules Engine and Decision Explainer enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To modify the expression set version configuration to show explanations to external users: | Rule Engine Designer |
| To select an explainability message template for an expression set version step: | Rule Engine Designer |
| To run expression set versions where explanations are shown to external users: | Rule Engine Designer<br>OR<br>Rule Engine Runtime |
| To access the Decision Explainer Log History component | Manage Flows<br>OR<br>View Developer Name |

1. From the App Launcher ( ), find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Expression Sets**.
3. Select the expression set.
4. On the Related tab of the selected expression set, select the version that you want to show explanations for, to external users, and then click **Open in Expression Set Builder**.
5. To open the Properties panel for the version, click .
6. Select **Show decision explanation**, and then save your changes.

   **Tip** When you configure an expression set version to show decision explanations, the action logs for the expression set run show the explainability messages added to the version steps by default. This behavior applies even when you deselect the Show decision explanation option after adding the messages to expression set version steps. To specifically exclude decision explanations from action logs, exclude the `actionContextCode` and the `explainabilitySpecName` properties from the API request for running expression sets.

7. On the Expression Set Builder canvas, select the step element that you want to show the explanation for.
8. To open the Element Details panel for the selected element, click .
9. Select **Show decision explanation**.
10. To include the explanation of a subexpression step in the overall rule explanation, select **Explain Subexpression**

    **Note** Enabling this setting may impact performance.

11. Select explainability message templates:
    - For the When Step Returns Output field, click , and then select an explainability message template for the combination of the selected step element and the Passed result type.
    - For the When Step Errors field, click , and then select an explainability message template for the

combination of the selected step element and the Failed result type.

- For the When Step Returns No Output field, click , and then select an explainability message template for the combination of the selected step element and the No Result result type.
  The No Result result type applies only to Lookup Table steps.

**Warning** Your external users see the explanation messages that you add to the selected steps. So, ensure that your explanations don't include any confidential information.

12. If any selected explainability message templates include tokens, map the tokens to the resources from the expression set version.
    The icon next to the selected explainability message template indicates that the template has tokens that require mapping.
    a. Click next to the template.
    b. On the Customize Decision Explanation window, in the search bar next to each token, search for and select a resource.

    **Note** You can map the tokens to local resources or resources that are created using Resource Manager, object field aliases, and variables that are imported from the lookup tables in the expression set version. You can't map the tokens to list variables from context definitions.

    The message preview is updated to show the mapped resources.

    c. In the Preview section, verify the customized message.
    d. Click **Done**.
13. Save your changes.
14. To preview how the explanations appear to your users, simulate the version, and select **Concise View**.

    **Tip** To enable service reps to see explanations in the object, you must add the Decision Explainer Log History component on the record page of the object within whose context the expression set is run. Go to the record page, then from Setup, click **Edit Page** and drag Decision Explainer Log History from Components to the page.

See Also

*Salesforce Help*: Create Expression Set Message Tokens
*Salesforce Help*: Create an Explainability Message Template

## Disable Decision Explanations for Expression Set Version Steps

To remove the explainability message templates assigned to an expression set version step, open the version in Expression Set Builder and deselect Show decision explanation for the step. You can disable decision explanations for expression set version steps only if the version is inactive.

REQUIRED EDITIONS

Available in: Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for Industries clouds that have Business Rules Engine and Decision Explainer enabled

| USER PERMISSIONS NEEDED | |
|---|---|
| To prevent showing decision explanations to external users for specific expression set version steps: | Rule Engine Designer |

1. Open the expression set version for which you want to disable showing decision explanations for specific steps.
   For more information, see Open an Expression Set Version in Expression Set Builder.
2. Select the step on the Expression Set Builder canvas.
3. To open the Element Details panel for the step, click .
4. Deselect **Show decision explanation**.

5. Save your changes.

## Show Rule Explanations to Users (Example)

This example shows how you can configure an expression set version to show step-result explanations to external users. The example also shows how to preview the explanations that your users see.

REQUIRED EDITIONS

Available in: Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for Industries clouds that have Business Rules Engine and Decision Explainer enabled

To encourage higher education among adults, a university offers a 20% discount on base fees to people over 45 who have a minimum grade point average (GPA) of 3.

You create an expression set that checks the eligibility of the student based on the age and GPA. The expression set returns the discounted fees if the student is eligible. You select the explainability message templates for the steps for which you want to show explanations to users. You then simulate the expression set and preview how your external users see the explanations.

1. Create an expression set version and define these variables and constants:

Variables and Constants

| Resource Name | Resource Type | Data Type | Initial Value |
|---|---|---|---|
| Birthdate | Variable | Date | – |
| Age | Variable | Number | – |

| Resource Name | Resource Type | Data Type | Initial Value |
|---|---|---|---|
| GPA | Variable | Number | – |
| DiscountedFees | Variable | Currency | – |
| BaseFees | Constant | Currency | 40,000 |
| EligibleDiscount | Constant | Percent | 20 |

2. Add these steps to the expression set version:
   a. Calculation: To calculate the age of the student based on the birth date that the student enters.

   b. Conditional Group: A condition that checks whether the student is above 45 and has a minimum GPA of 3.
   c. Calculation: A calculation inside the conditional group. The calculation runs only if the condition evaluates to true. The calculation provides the final discounted fees.

3. Select the condition step, and then select the relevant explainability message templates that provide the explanations for when the condition is met (Passed) and isn't met (Failed).

Explainability Message Templates Selected

| Template Name | Template Message | Template Usage Scenario |
|---|---|---|
| DiscountEligible | Based on your age and GPA, you're eligible for a 20% discount on the base fees. | Condition result is Passed |
| DiscountNotEligible | You don't meet the eligibility criteria for a fee discount. To be eligible for the discount, your age must be at least 45, and have a minimum GPA of 3. | Condition result is Failed |

4. Simulate the version to test whether everything works as expected by using these sample input values:
   - Age: `50`
   - GPA: `3.5`

5. As expected, your simulation results show that the input values that you entered qualify for a discounted fee.

6. To preview how your viewers see the explanation that you added to the condition step, click ▼, and

then select **Concise View**.

## Compare Expression Set Runs with Decision Explainer

To analyze why an expression set generated different outputs for each run, enable Decision Explainer for expression set versions, and then use the Compare option on the Decision Explainer Log History component. Decision Explainer logs are generated in the context of object records that an expression set is run for. To use the Compare option, the Decision Explainer Log History component must be enabled on the record page of the object within whose context the expression set is run.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for Industries clouds that have Business Rules Engine and Decision Explainer enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create an expression set: | Rule Engine Designer |
| To create Decision Explainer object records: | Decision Explainer Service Access |
| To create a flow to run the expression set: | Manage Flows<br>AND<br>Rule Engine Runtime |

1. To run the expression set using Flows, use the runExpressionSet action in a flow.
2. Debug or run the flow.
3. Open the record detail page of the object in whose context the expression set was run.
4. Locate the Decision Explainer Log History component on the page.

   Here, the Decision Explainer Log History component is added to the expression set object's record detail page.

5. To compare the results of the expression set's runs, select at least two runs of the expression set, and click **Compare**.

Here, you see the reasoning behind why the same expression set generated two different outputs based on the different inputs.

**Example** You created an expression set named Maintenance Charge Calculation to calculate the maintenance charges for an apartment based on the apartment's size in square feet. You enabled Decision Explainer for the version. Then, you created these resources to store the values required to calculate the maintenance charges.

| Resource Name | Type | Data Type |
|---|---|---|
| Base_Charge | Constant (with the value $100) | Currency |
| Apt_Size_Sq_Feet | Variable | Number |
| Maintenance_Charges | Variable | Currency |

You configured a calculation step in the version that takes the apartment's size as input, and returns the maintenance charges. To make the message in the template more meaningful to the users, you mapped the message tokens to resources from the version. You then activated the version. You created a flow and added the expression set action to run the expression set.

- For __actionContextCode, you entered the record ID of the expression set.This is the context that you used to capture the Decision Explainer logs. You had also added the Decision Explainer Log History component to the expression set record detail page.
- For __explainabilitySpecName, you entered the name of the explainability action definition.

  **Note** Ensure that the explainability action definition you enter has an active explainability action version associated with it.

- For Apt_Size_Sq_Feet, which is the expression set version's resource, you created a local flow variable to pass the apartment size to the expression set from the flow.

You then saved the flow and tested it with a few sample apartment size values. You opened the record detail page of the expression set, and compared the results of the expression set runs with different input values from Decision Explainer Log History.

See Also

*Salesforce Help*: Show Rule Explanations to Users
*Salesforce Help*: Show Rule Explanations to Users (Example)

## Explainability Message Templates for Developers

Developers in your Salesforce org can use developer resources to use and extend explainability message templates.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for Industries clouds that have Business Rules Engine and Decision Explainer enabled |

## Connect REST API

- Retrieve a list of explainability message templates based on the search criteria
- Retrieve details of a specific explainability message template
- Retrieve explainability logs

## Metadata API

- Create an explainability message template

# Business Rules Engine Integrations

Expression sets and lookup tables–the Business Rules Engine components–can be called from Flow and Omnistudio. You can also use Business Rules Engine Business APIs to call these components from workflows that are created by using third-party tools.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Omnistudio**
When your Salesforce instance has Business Rules Engine and Omnistudio enabled, you can call expression sets and decision matrices from Omniscripts and Omnistudio Integration Procedures.

**Flow**
Call expression sets and decision matrices from Flow.

**Invocable Actions**
You can invoke decision matrices and expression sets–the Business Rules Engine components–by using Apex or REST API.

**Third-Party Workflows**
Business Rules Engine business APIs give you the flexibility of calling expression sets and lookup tables in workflows created using third-party tools of your choice.

**Considerations for Calling Expression Sets**
Before you call expression sets from Flows, Omnistudio, or third-party workflows, review these considerations.

**Monitor Guardrail Limits with Apex**
Fetch and monitor guardrail limits for your org using an Apex class. You can also invoke the Apex class in a flow.

**How to Bring External Data into Business Rules Engine**
Review the methods and processes to bring data from your external systems into Business Rules

Engine, and choose the method that best suits your business requirements.

## Omnistudio

When your Salesforce instance has Business Rules Engine and Omnistudio enabled, you can call expression sets and decision matrices from Omniscripts and Omnistudio Integration Procedures.

**Call an Expression Set from an Integration Procedure**
To call an expression set from an Integration Procedure, and return the specified expression set's results to the Integration Procedure, use the Expression Set Action.

**Call a Decision Matrix from an Integration Procedure**
To call a decision matrix from an Integration Procedure, and return the specified matrix's results to the Integration Procedure, use the Decision Matrix Action.

**Call an Expression Set from an Omniscript**
To call an expression set that has at least one active version from an Omniscript, and to return the specified expression set's results to the Omniscript, use the Expression Set Action.

**Call a Decision Matrix from an Omniscript**
To call a decision matrix that has at least one active version from an Omniscript, and to return the specified matrix's results to the Omniscript, use the Decision Matrix Action.

### Call an Expression Set from an Integration Procedure

To call an expression set from an Integration Procedure, and return the specified expression set's results to the Integration Procedure, use the Expression Set Action.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To access Omnistudio to create and modify an Integration Procedure: | Omnistudio Admin |
| To access Business Rules Engine to create and activate an expression set version: | Rules Engine Designer |
| To call an expression set from an Integration Procedure: | Omnistudio Admin |

You can call only expression sets that have at least one active version from an Integration Procedure.

To call expression sets that have a context definition, first hydrate the context definition and then use

Context IDs that you use to hydrate context data in the payload. See Create a Context Definition.

To call expression sets that use Decision Explainer, define **actionContextCode** and **explainabilitySpecName** as keys in remote properties in the Integration Procedure.

1. From the App Launcher ( ), find and select **Omnistudio**.
2. From the app navigation menu, select **Omnistudio Integration Procedures**.
3. Select an Integration Procedure, and then select the version that you want to modify.
4. From the Actions panel, drag **Expression Set Action** onto the Structure panel.
5. In the Remote Properties section, for Configuration Name, enter the name of the expression set that you want to call.

   **Note** Don't edit any other properties.
6. Enter these properties:
   - Send JSON Path: Specify a node for the data JSON that's sent in the outgoing request. For example, *input*.
   - Send JSON Node: Specify a new name for the parent node in the outgoing request. For example, *my_input*.
7. From the Actions panel, drag **Response Action** after the Expression Set Action section in the Structure panel.
8. Enter these properties:
   - Send JSON Path: Enter the element name of the Expression Set Action. This path is used as the JSON node to store the data.
   - Send JSON Node: Specify a new name for the parent node in the outgoing request. For example, *my_output*.
9. Click **Preview**.
10. In the Input Parameters section, click **Edit as JSON**, and then provide the input values in this format:

```
{
  Name of the Send JSON Path specified in the Expression Set Action: {
    "input variable 1": "value1",
    "input variable 2": "value2",
    "input variable n": "valueN",
   }
}
```

For example:

```
{
    "input": {
        "GPA": "3.5",
        "Fees": "20000",
        "Birthdate": "2005-09-01"
    }
}
```

Here:

- `input` is the name of the Send JSON Path specified in the Expression Set Action.
- `GPA`, `Fees`, and `Birthdate` are the input variables in the expression set that's invoked by using the Expression Set Action.

11. Click **Execute**.
    The expression set output for the input values that you entered appears.

## Call a Decision Matrix from an Integration Procedure

To call a decision matrix from an Integration Procedure, and return the specified matrix's results to the Integration Procedure, use the Decision Matrix Action.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To access Omnistudio to create and modify an Integration Procedure: | Omnistudio Admin |
| To access Business Rules Engine to create and activate a decision matrix version: | Rules Engine Designer |
| To call a decision matrix from an Integration Procedure: | Omnistudio Admin |

You can call only active decision matrices from an Integration Procedure.

1. From the App Launcher ( ), find and select **Omnistudio**.
2. From the app navigation menu, select **Omnistudio Integration Procedures**.
3. Select an Integration Procedure, and then select the version that you want to modify.
4. From the Actions panel, drag **Decision Matrix Action** onto the Structure panel.
5. In the Decision Matrix Properties section, specify these decision matrix action properties:

| Field | Description |
|---|---|
| Matrix Input Parameters: Data Source | The name of a data JSON node in the Integration Procedure that contains a value to pass to the decision matrix. |
| Matrix Input Parameters: Filter Value | The name of the corresponding decision matrix input variable that accepts the value. |

| Field | Description |
| --- | --- |
| Matrix Name | The name of the decision matrix that you want to call. |
| Remote Options | Additional options to pass to the Decision Matrix. These options are predefined, but you can also pass options specific to the matrix.<br>• `executionDateTime` – Sets the Effective Date of the Decision Matrix. |
| Default Matrix Result | Value to return if the Decision Matrix returns null |

For example, if the name of your decision matrix is StudentDiscountEligibility, and the input variables that the matrix takes are `Age` and `GPA`, then enter these values:

- Matrix Name: *`StudentDiscountEligibility`*
- Matrix Input Parameters:

| Data Source | Filter Value |
| --- | --- |
| *`Age`* | *`Age`* |
| *`GPA`* | *`GPA`* |

6. From the Actions panel, drag the **Response Action** component, and place it below the Decision Matrix Action in the Structure panel.
7. Enter these properties:
   - Send JSON Path: Enter the element name of the Decision Matrix Action. This path is used as the JSON node to store the data.
   - Send JSON Node: Specify a new name for the parent node in the outgoing request. For example, *`my_output`*.
8. Click **Preview**.
9. In the Input Parameters panel, add Key/Value pairs.
   For example:

| Key | VALUE |
| --- | --- |
| Age | *`16`* |
| GPA | *`3.5`* |

10. Click **Execute**.
    Here's what the output looks like:

```
{
  "my_output": {
```

```
    "result": [
      {
        "Qualified": true
      }
    ]
  }
}
```

## Call an Expression Set from an Omniscript

To call an expression set that has at least one active version from an Omniscript, and to return the specified expression set's results to the Omniscript, use the Expression Set Action.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To access Omnistudio to create and modify an Omniscript: | Omnistudio Admin |
| To access Business Rules Engine to create and activate an expression set: | Rules Engine Designer |
| To call an expression set from an Omniscript: | Omnistudio Admin |

To call expression sets that use Decision Explainer, define **actionContextCode** and **explainabilitySpecName** as keys in the Omniscript under Expression Set Action Properties.

1. From the App Launcher ( ), find and select **Omnistudio**.
2. From the app navigation menu, select **Omniscripts**.
3. Select the Omniscript and the version from which you want to call the expression set.
4. Expand the step from which you want to call the expression set.
5. From the Build section, under Inputs, drag the input fields that are relevant to the inputs that your expression set uses.
   For example, if the expression set that you call uses values from the input variables *AnnualFees*, *Age*, and *GPA* to calculate the discounted student fees, add these input fields:

| Input Field Type | FIELD Name |
|---|---|
| Number | Age |

| Input Field Type | FIELD Name |
|---|---|
| Number | GPA |
| Currency | Fees |

**Note** Ensure that the input field name matches the name of the input variable that's used in the expression set. You can specify field labels of your choice.

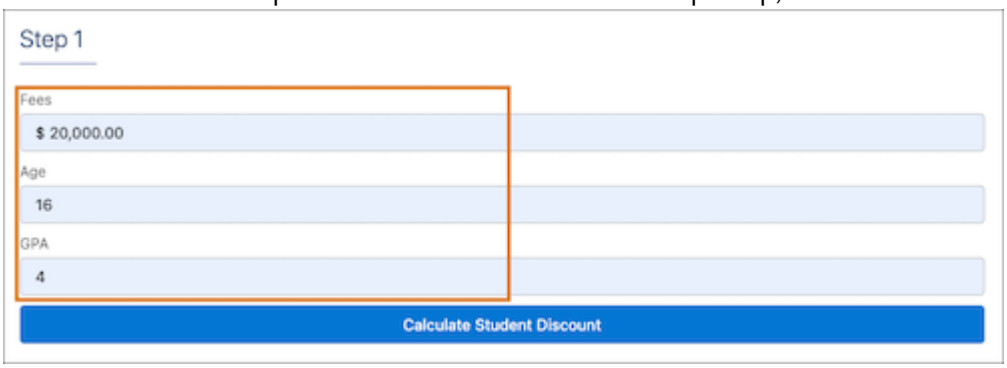
6. From the Build section, under Actions, drag **Expression Set Action** and place it below the input fields.
7. Select the action, and then under the Properties section:
   a. Enter a name, and if necessary, a label for the action.
   b. Expand **Remote Properties**.
   c. For Configuration Name, enter the name of the expression set that you want to call.
   d. Expand **Send/Response Transformations**.
   e. For Send JSON Path, enter the name of the step to which you added the input fields.

      **Tip** Specifying the step name that contains the input field ensures that information from the step is passed to the expression set only, rather than to the entire Omniscript JSON.

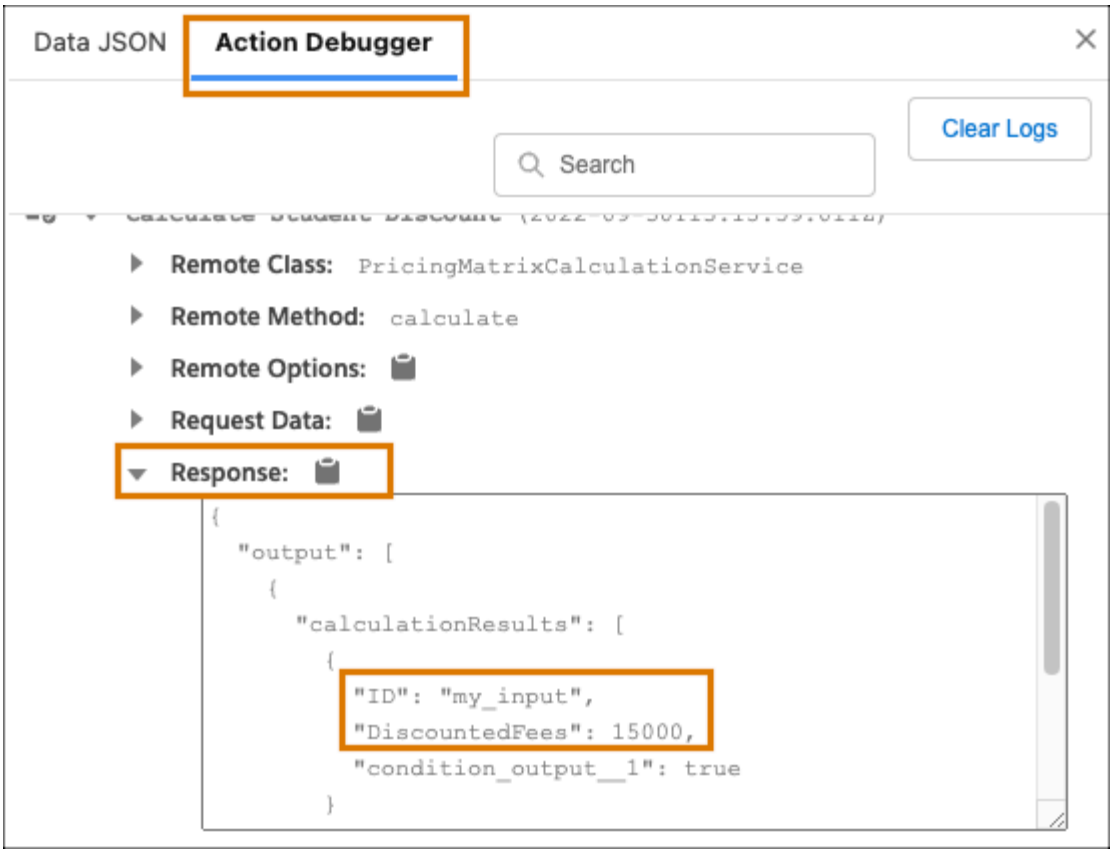
   f. For Send JSON node, enter a name that you want to specify as the root node name in the outgoing response.

      For example, `my_input`.

8. To test whether the Omniscript calls the expression set successfully, click **Preview**.
9. Enter values for the input fields in the relevant Omniscript step, and then run the Omniscript.




10. To view the results of the expression set action, expand the step that contains the input fields and the expression set action, and then under Action Debugger, expand **Response**.




Here, ID represents the Send JSON node that you specified, and DiscountedFees represents the output variable defined in the expression set that the Omniscript called.

## Call a Decision Matrix from an Omniscript

To call a decision matrix that has at least one active version from an Omniscript, and to return the specified matrix's results to the Omniscript, use the Decision Matrix Action.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To access Omnistudio to create and modify an Omniscript: | Omnistudio Admin |
| To access Business Rules Engine to create and activate a decision matrix: | Rules Engine Designer |
| To call a decision matrix from an Omniscript: | Omnistudio Admin |

1. From the App Launcher ( ⠿ ), find and select **Omnistudio**.
2. From the app navigation menu, select **Omniscripts**.
3. Select the Omniscript and the version from which you want to call the decision matrix.
4. Expand the step from which you want to call the decision matrix.
5. From the Build section, under Inputs, drag the input fields that are relevant to the inputs that your decision matrix uses.
   For example, if the decision matrix uses values from the input variables *Age* and *GPA* to determine a student's eligibility for a discount on annual fees, add these input fields:

| Input Field Type | FIELD Name |
|---|---|
| Number | Age |
| Number | GPA |

   **Note** Ensure that the input field name matches the name of the input variable that's used in the decision matrix. You can specify field labels of your choice.

6. From the Build section, under Actions, drag **Decision Matrix Action** and place it below the input fields.
7. Select the action, and then under the Properties section:
   a. Enter a name, and if necessary, a label for the action.
   b. Under Matrix Input Parameters, enter the input variables to pass to the decision matrix.

      For example:

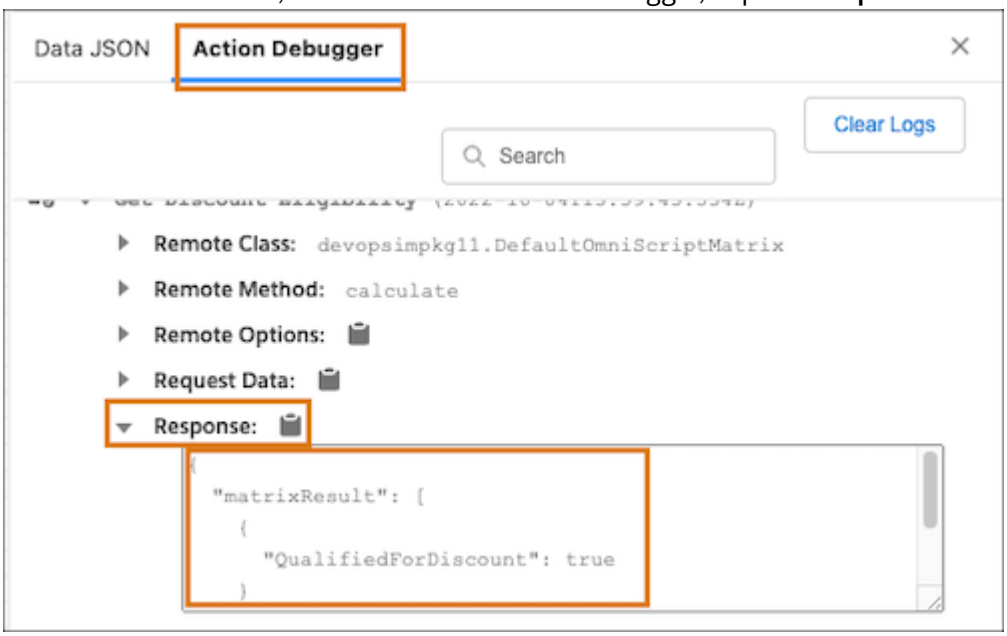
| Data Source | Filter Value |
|---|---|
| Age | Age |
| GPA | GPA |

   c. Expand **Remote Properties**.
   d. For Matrix Name, select the decision matrix that you want to call.
   e. Expand **Send/Response Transformations**.
   f. For Response JSON Path, enter a name for the response node in which you want to capture the results of the matrix.

      For example, `matrixResult`.

   g. For Response JSON node, enter the name of the Omniscript step to which you added the input fields for the matrix and the decision matrix action.
8. To test whether the Omniscript calls the decision matrix successfully, click **Preview**.
9. Enter values for the input fields in the relevant Omniscript step, and then run the Omniscript.

10. To view the results of the decision matrix action, expand the step that contains the input fields and the decision matrix action, and then under Action Debugger, expand **Response**.




Here, matrixResult represents the Response JSON path that you specified. Based on the input values passed to the decision matrix, the result returned is that the student is qualified for a discount (`"QualifiedForDiscount": true`).

## Flow

Call expression sets and decision matrices from Flow.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

To call expression sets with context definitions from a flow, see How to Use Business Rules Engine to Preapprove Health Cloud Care Requests.

To call an expression set with a Decision Explainer component from a flow, see Example: Create a Flow to Run the Expression Set and Update the Record.

**Invoke a Decision Matrix in a Flow**
After a decision matrix is activated, invoke the decision matrix in a flow to provide outcomes and use the outcomes in the flow.

**Invoke a Decision Table in a Flow**
After a decision table is activated, invoke the decision table from a flow to get and use the output in the flow.

**Invoke an Expression Set in a Flow**
After an expression set version is activated, invoke the version in a flow to run the rules configured in the expression set and return the outcomes to the flow.

**Invoke an Expression Set That Uses Decision Explainer from a Flow**
After an expression set version that has decision explanations enabled is activated, call the expression set that the version belongs to from a flow. To capture action logs for the expression set runs, specify values for the autogenerated __actionContextCode and __specName parameters in the flow.

**Example of Invoking a Decision Explainer Enabled Expression Set from Flow**
This example shows how to invoke an expression set that uses Decision Explainer from flow. The example also shows how to view every run of the expression set in the action logs for the context in which the expression set is run.

## Invoke a Decision Matrix in a Flow

After a decision matrix is activated, invoke the decision matrix in a flow to provide outcomes and use the outcomes in the flow.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create a flow: | Manage Flow |
| To invoke a decision matrix: | Rule Engine Runtime |

1. From Setup, in the Quick Find box, enter `Flows`, and then select **Flows**.
2. Click **New Flow**.
3. In the New Flow window, select a flow type, and then click **Create**.
4. Click ⊕, and then select **Action**.
5. In the New Action window, complete these steps:
   a. In the category section, select **Decision Matrices**.

**Note** The Decision Matrices category is available in the New Action window only when your Salesforce org has a decision matrix with at least one active version.

b. In the Action field, look for, and then select the decision matrix you want to run.

**Tip** When you invoke a decision matrix, the matrix version that's active at the invocation time is considered. If multiple matrix versions are active at the same time, the version with the highest rank is considered. For example, when a matrix has active versions with ranks 1, 2, 3, and 4, the version with rank 4 is considered.

c. Enter a label for the decision matrix you selected.
The API name autopopulates from the label you enter.
d. If needed, enter a description.
e. In the Set Input Values section, enter values for the input variables defined in the decision matrix.
f. In the Advanced section, if needed, enable manual assignment for output variables.
With manual assignment, you can map the decision matrix output to a new resource that the flow can use.
g. Click **Done**.

6. Use other elements in the flow to decide how the outcomes returned by the decision matrix are stored or processed later.
7. Save and activate your flow.

## Invoke a Decision Table in a Flow

After a decision table is activated, invoke the decision table from a flow to get and use the output in the flow.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create a flow: | Manage Flows |
| To create and activate a decision table: | Rule Engine Designer |
| To invoke a decision table in a flow: | Rule Engine Runtime<br>OR<br>Run Engine Designer |

1. From Setup, in the Quick Find box, enter `Flows`, and then select **Flows**.
2. Click **New Flow**.

3. In the New Flow window, select a flow type, and then click **Create**.
4. In the flow builder, click ⊕ under the step for invoking the decision table.
5. Select **Action**.
6. In the New Action window, in the Category section, select **Decision Table**.

   **Note** The Decision Table category is available only when your Salesforce org has at least one active decision table.

7. In the Action field, search for and select the decision table that's suffixed with _Default.
8. In the New Action window:
   a. Enter a label for the Decision Table action.
      The API Name autopopulates from the label. For example, if the label you enter is *`Find Student Discount Eligibility`*, the autofilled API Name is Find_Student_Discount_Eligibility. If necessary, you can modify the API Name.
   b. Enter a description.
   c. Include the input fields that you want to provide values for.
   d. For each decision table field that's included in the action, provide values to evaluate.
   e. Click **Done**.
9. Specify how to pass the input values to the action and how to process the decision table's output values further in the flow.
10. Save your flow.
11. To verify that the flow works as expected, run the flow.
12. Activate the flow.

## Invoke an Expression Set in a Flow

After an expression set version is activated, invoke the version in a flow to run the rules configured in the expression set and return the outcomes to the flow.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create a flow: | Manage Flow |
| To invoke an expression set: | Rules Engine Runtime |

To call expression sets with context definitions from a flow, see How to Use Business Rules Engine to Preapprove Health Cloud Care Requests.

To call an expression set with a Decision Explainer component from a flow, see Example: Create a Flow

to Run the Expression Set and Update the Record.

1. From Setup, in the Quick Find box, enter `Flows`, and then select **Flows**.
2. Click **New Flow**.
3. In the New Flow window, select a flow type, and then click **Create**.
4. Click ⊕, and then select **Action**.
5. In the New Action window, complete these steps:
   a. In the category section, select **Expression Sets**.

      **Note** The Expression Sets category is available in the New Action window only when your Salesforce org has an expression set with at least one active version.

   b. In the Action search bar, look for, and then select the expression set you want to run.
   c. Enter a label for the expression set you selected. The API name autopopulates from the label you enter.
   d. If needed, enter a description.
   e. In the Set Input Values section, enter values for the input variables defined in the expression set. Input variables from all the active versions of the expression set are shown. Enter values for all the variables. When the expression set runs, variables from only the active expression set version with the highest rank are considered.

      **Note** If the expression set uses a field alias as a variable, pass the Id field of the source object, to which the field alias belongs, as an input variable to the flow.

   f. In the Advanced section, if needed, enable manual assignment of variables. With manual assignment, you can map the expression set output to a new resource that the flow can use.
   g. Click **Done**.
6. Use other elements in the flow to decide how the outcomes returned by the decision matrix are stored or processed later.
7. Save and activate your flow.

   **Note** When a Flow is invoked using Batch management (for example, for processing 200 records), the calls to expression sets are automatically bulkified, which allows the execution of multiple inputs in a single request.

## Invoke an Expression Set That Uses Decision Explainer from a Flow

After an expression set version that has decision explanations enabled is activated, call the expression set that the version belongs to from a flow. To capture action logs for the expression set runs, specify values for the autogenerated __actionContextCode and __specName parameters in the flow.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create explainability message templates, explainability action definitions, and explainability action versions: | Decision Explainer Service Access |
| To enable Decision Explainer for an expression set version: | Rule Engine Designer |
| To invoke an expression set from Flow | Manage Flows |

To enable Decision Explainer for an expression set and add explainability message templates to a version, your Salesforce instance must have Decision Explainer enabled.

1. From Setup, in the Quick Find box, enter `Flows`, and then select **Flows**.
2. Click **New Flow**.
3. Select a flow type, and then click **Create**.
4. Click ⊕, and then select **Action**.
5. In the Action search bar, search for, and select the expression set you want to run.
6. Enter a label for the expression set. The API name autopopulates from the label you enter.
7. If needed, enter a description.
8. Set input values for the expression set:
   a. Include **__actionContextCode** and select the variable that captures the record ID of the object for which the expression set is run.
   b. Include **__explainabilitySpecName** and enter the name of the explainability action definition that defines the purpose for which the expression set is run.

      **Important** Ensure that the explainability action definition you specify has an associated, active explainability action version.

   c. If needed, include other variables from the expression set and set their values.

      Here, {!contactRecord.Id} is the local flow variable to capture the record ID of the contact for whom this flow is run. Example is the explainability action definition, and {ApplicantBirthdate} is the local flow variable that captures the user's input.

9. Save your changes.
10. To verify that the flow works as expected, run or debug the flow using sample values.

    .

    Here, the flow passes the contact record's birthdate to the expression set, which uses the date to calculate the age. The expression set is configured to have $10,000 as the course fees. Based on the calculated age, the expression set returns the discounted fees of $7,000 to the flow.

11. Go to the record detail page of the object for which you entered the record ID in the **__actionContextCode** field.

**Important** To see the decision explainer action logs, the record detail page of the object used as the actionContextCode must have the Decision Explainer Log History component added to the object's record page

.

Here's the record detail page of the sample contact specified while debugging the flow. The Decision Explainer Log History section captures each run of the expression set.

12. To view step-wise details and output of the expression set version called from the flow, select a specific run, and click **View**.

See Also

Create an Explainability Message Template
Show Rule Explanations to Users
Decision Explainer

## Example of Invoking a Decision Explainer Enabled Expression Set from Flow

This example shows how to invoke an expression set that uses Decision Explainer from flow. The example also shows how to view every run of the expression set in the action logs for the context in which the expression set is run.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for Industries clouds that have Business Rules Engine and Decision Explainer enabled. |

To promote fitness and well-being, a company is offering a discount on its annual gym membership to its contacts. To encourage physical movement, the company launched a week-long step challenge for its contacts. The annual membership price is $1,000. A discount is offered on this price based on a contact's age and the step count from the week-long step challenge.

The company's sales team gave the rule designer and the Salesforce admin the age and step count ranges and the applicable discount percentages. The company also asked that the discounted gym membership fees be updated on a contact's record detail page for quick reference.

The rule designer contacted the company's Salesforce admin and together they divided the requirement into these tasks:

| Task | Description |
|---|---|
| Create a decision matrix to calculate the discount | The matrix uses an input column with the |

| Task | Description |
| --- | --- |
| percent based on the step count from the step challenge. | datatype number range to capture the step count. The matrix captures the discount percent in the output column for these ranges. |
| Create a decision matrix to get age-based discount. | The matrix uses an input column with the data type number range to define the age range. The matrix captures the discount percent in the output column for these ranges. |
| Create an expression set and its version to design the end-to-end rule to calculate the discounted gym membership. | The expression set version:<br>• Takes the step count as an input and looks up the decision matrix to get the step-based discount.<br>• Uses the contact's birth date to calculate age.<br>• Passes the contact's age to the age-based discount matrix to get the additional discount.<br>• Calculates the total discount by adding the age-based and the step-based discounts.<br>• Calculates the discounted premium by subtracting the total discount percent from the base membership fee of $1,000.<br>• Enables Decision Explainer for customer service representatives to understand the logic behind the discounted membership fee for a specific contact. |
| Create a flow to run the expression set and return the output to the flow. | • The flow captures the contact's step count and passes it to the expression set action.<br>• The expression set action is configured to use a contact's record ID as the context in which the flow and the expression set run.<br>• The expression set action returns the discounted membership fees to the flow.<br>• The flow then updates the contact record with the discounted fees. |
| To launch the flow for a contact, add the flow action to the contact record detail page. | • The company's customer service representatives can launch the flow for a contact, and enter the contact's step count for the previous week.<br>• The flow runs in the background and updates |

| Task | Description |
| --- | --- |
| | the Gym Membership Fees field for the contact record. |
| View the action log for the flow. | The customer service associates can see the Decision Explainer Log History on the contact's record detail page to check how the gym membership fees for the contact were calculated. |

**Example: Create a Decision Matrix to Get Step-Based Discount Percent**
Create a decision matrix and configure its version with an input column for number of steps, and an output column with the corresponding discount percent based on a number range of steps.

**Example: Create a Decision Matrix to Get Age-Based Discount**
Create a decision matrix and configure its version with an input column for age, and an output column with corresponding discount percent based on the age range.

**Example: Create an Expression Set to Calculate Discounted Membership Fees**
Configure the expression set version to calculate a contact's age from their birth date, look up the contact's step count to get the steps-based discount, and return the discounted gym membership fee after subtracting the total discount from the base membership fee.

**Example: Create a Flow to Run the Expression Set and Update the Record**
Create an input form to capture a contact's step count and pass it to the expression set action. Configure the flow to update the contact record with the expression set's output.

**Example: Add the Flow to the Contact Record Page and Run the Flow**
To enter the step count for a contact from the contact's record page, and pass the contact's record ID to the flow from the same page, add a flow component to the Contact Record Page.

**Example: View Decision Explainer Log History**
View the action log for the expression set that was run from the flow for a contact record.

**See Also**

Create an Explainability Message Template
Show Rule Explanations to Users
Show Rule Explanations to Users (Example)
Decision Explainer for Expression Sets

## Example: Create a Decision Matrix to Get Step-Based Discount Percent

Create a decision matrix and configure its version with an input column for number of steps, and an output column with the corresponding discount percent based on a number range of steps.

REQUIRED EDITIONS

| USER PERMISSIONS NEEDED | |
|---|---|
| To create decision matrices and activate matrix versions: | Rule Engine Designer |

Create a decision matrix named StepBasedDiscount. This matrix uses step count as input, and returns the matching discount percent.

1. Click ⠿, and then find and select **Business Rules Engine**.
2. From the app navigation menu, select **Lookup Tables**.
3. Click **New**.
4. Select **Decision Matrix** | **Next**.
5. For name, enter `StepBasedDiscount`, and for Type, select **Standard**.
6. Save your changes.
7. On the Related tab on the decision matrix record page, select **StepBasedDiscount V1**.
8. On the Matrix tab, click **Add Column**, and then create two columns with these settings.

| Column Header | Header Type | Data Type |
|---|---|---|
| Steps | Input | Number Range |
| Discount | Output | Percent |

9. Click **Done**.
10. To create a row, click **Add Row**.
11. To add row data, click a table cell, click ✎ in the cell, and then enter the value.
    Add these rows and values.

| Steps | Discount Percent |
|---|---|
| 35000 | 0 |
| 35,001 | 5 |
| 50,001 | 10 |
| 70,001 | 15 |
| 100,000 | 25 |

    The Steps column in this matrix version represents number ranges. The values in the column define these ranges: Below 35000, 35,001–50,000, 50,001–70,000, 70,001–99,999, and 1000,000 or above.
12. Save your changes.
13. Activate the decision matrix version.
    Expression sets can call only active matrix versions.

## Example: Create a Decision Matrix to Get Age-Based Discount

Create a decision matrix and configure its version with an input column for age, and an output column with corresponding discount percent based on the age range.

REQUIRED EDITIONS

| USER PERMISSIONS NEEDED | |
|---|---|
| To create decision matrices and activate matrix versions: | Rule Engine Designer |

Create a decision matrix named AgeBasedDiscount. This matrix uses the contact's age count as input, and returns the matching discount percent.

1. From app navigation menu of Business Rules Engine, select **Lookup Tables**.
2. Click **New**.
3. Select **Decision Matrix** | **Next**.
4. For name, enter `AgeBasedDiscount`, and for Type, select **Standard**.
5. Save your changes.
6. On the Related tab on the decision matrix record page, select **AgeBasedDiscount V1**.
7. On the Matrix tab, click **Add Column**, and create columns with these settings.

| Column Header | Header Type | Data Type |
|---|---|---|
| ContactAge | Input | Number Range |
| Discount | Output | Percent |

8. Click **Done**.
9. To create a row, click **Add Row**.
10. To add row data, click a table cell, click in the cell, and then enter the value
    Add these rows and values.

| Steps | Discount Percent |
|---|---|
| 17 | 0 |
| 18 | 5 |
| 31 | 10 |
| 46 | 15 |
| 60 | 25 |

    The steps column in this matrix version represents number ranges. The values in the column define these ranges: Below 18, 18–30, 31–45, 46–59, and 60 or above.
11. Save your changes.

12. Activate your decision matrix version.
    Expression sets can call only active matrix versions.

## Example: Create an Expression Set to Calculate Discounted Membership Fees

Configure the expression set version to calculate a contact's age from their birth date, look up the contact's step count to get the steps-based discount, and return the discounted gym membership fee after subtracting the total discount from the base membership fee.

REQUIRED EDITIONS

| USER PERMISSIONS NEEDED | |
|---|---|
| To create a field alias for the Birthdate field in the Contact object: | System Administrator for a Salesforce org with Business Rules Engine |
| To create the expression set, and design and activate its version | Rule Engine Designer |

The field alias (Applicant.Birthdate) for using the value of the Birthdate field from the Contact object must be created in Setup, and available to use in the expression set. See Create Source Field Aliases.

Create an expression set named AnnualGymMembership. In Version 1 of this expression set, add steps to use the contact's inputs and return the discounted gym membership fees.

1. From the app navigation menu of Business Rules Engine, select **Expression Sets**.
2. Click **New**.
3. Enter `AnnualGymMembership` as the name, and save your work.
   The expression set and its first version are created.
4. On the record page for the new expression set, on the Versions tab, select **AnnualGymMembership V1**, and then click **Open in Expression Set Builder**.
5. Set the properties for the expression set version:
   a. Click , and enter a start date time for the expression set version.
   b. Enter a rank.

      The default is 1.

   c. Select **Show decision explanation**.
   d. Close the Expression Set Properties panel.
6. Create resources for the expression set:
   a. Click .
   b. Click **Add Resource**, and then create these variables and constants.

      **Note** After adding a resource, to add the next resource, click **Done and New**. After adding all required resources, click **Done**.

Resources

| Resource Type | Variable NAME | Data Type | Decimal | Default Value |
|---|---|---|---|---|
| Constant | GymFees | Currency | 0 | 1000 |
| Variable | ContactAge | Number | 0 | – |
| Variable | TotalDiscount | Percent | 0 | – |
| Variable | DiscountedGym Fees | Currency | 2 | – |

**Tip** Save the version after making any change.

7. To get the discount percent based on the step count, add a Lookup Table step.
   a. Click ⊕ on the builder canvas, and select **Lookup Table**.
   b. Search for and select **StepsBasedDiscount**.
   The matrix is added and the input and output columns from the matrix appear as local variables in the resources list.
8. To calculate the contact's age from their birth date, add a Calculation step.
   a. Click ⊕ on the builder canvas, and select **Calculation**.
   b. In the Formula field, search for and select **AGE (Date)**.
   c. To replace Date in Age (Date) with the value of the Birthdate field on the Contact object, search for and select **Applicant.Birthdate** so that the formula looks like this: *`Age(Applicant.Birthdate)`*.
   Here. Applicant is the object alias for Contact, and Birthdate is the field alias for the Birthdate field in Contact.
   d. In the Output Variable field, search for and select **ContactAge**.
9. To use the contact's age to a get the applicable age-based discount, add a Lookup Table step.
   a. Click ⊕ on the builder canvas, and select **Lookup Table**.
   b. Search for and select **AgeBasedDiscount**.
   The matrix is added and the input and output columns from the matrix appear as local variables in the resources list.
10. To calculate the total discount based on the contact's age and step count, add a Calculation step.
   a. Click ⊕ on the builder canvas, and select **Calculation**.
   b. In the Formula field, enter this formula: `StepBasedDiscount_Discount + AgeBasedDiscount_Discount`.
   c. In the Output Variable field, search for and select **TotalDiscount**.
11. To calculate the discounted gym membership fees, add a Calculation step.
   a. Click ⊕ on the builder canvas, and select **Calculation**.
   b. In the Formula field, enter this formula: `GymFees - ( TotalDiscount * GymFees ).`
   c. In the Output Variable field, search for and select **DiscountedGymFees**.
12. Select each step individually, click ⁘, and select Include in output.

   **Tip** To save the version, this action is required for at least one step. However, to include step-wise details in the Decision Explainer logs, complete this action for every step.

13. Click **Simulate** and test the version by entering sample inputs.

14. Activate the version.

## Example: Create a Flow to Run the Expression Set and Update the Record

Create an input form to capture a contact's step count and pass it to the expression set action. Configure the flow to update the contact record with the expression set's output.

REQUIRED EDITIONS

| USER PERMISSIONS NEEDED | |
|---|---|
| To create a flow: | Manage Flows |

An Explainability Action Definition named Example must be available in your Salesforce org. The definition must also have an associated Explainability Action Version.

1. From Setup, in the Quick Find box, enter Flows, and then select **Flows**.
2. Click **New Flow**.
3. Select **Screen Flow** and click **Create**.
4. On the Flow Builder canvas, click ⊕, and select **Screen**.
5. Configure the screen.
   a. In the Screen Properties, for Label, enter `Gym Membership Offer`.
      The API Name autopopulates.
   b. Expand Configure Header and deselect **Show Header**.
   c. Expand Configure Footer, and for Next or Finish Button, select **Use a custom label** and enter `Calculate`.
      Hide the Previous and Pause buttons if you don't need them.
   d. From the Components panel, select **Display Text**, and enter an API name for the component.
   e. To set the context to launch the flow, enter some explanatory text. .
      Here's an example.

   f. From the Components panel, select **Number**, and for Label, enter `Step Count`.
   g. Set the Number component as required.
   h. Save your changes.
6. To invoke the expression set, click ⊕, and select **Action**.
7. In the Action field, search for Gym, and select the AnnualGymMembership expression set.

8. For label, enter `Calculate Gym Membership`.
9. To capture Decision Explainer logs for each run of the expression set, include **_actionContextCode**.
   a. To capture action logs on the Contact object, in the _actionContextCode field, select **New Resource**.
   b. For Resource Type, select **Variable**.

c. For API name, enter `ContactRecord`.
d. For Data Type, select **Record**.
e. For Object, select **Contact**.
f. Select **Available for Input** and **Available for Output**.
g. Click **Done**.
   The variable appears in the _actionContextCode field.
h. Click inside {} and enter `.Id` after !ContactRecord.

10. For _explainabilitySpecName, enter `Example`.
11. For ApplicantId, from the Record (SINGLE) Variables section, select **ContactRecord**, and then select **Id**.

12. To pass the step count to the expression set when the flow is triggered, for Steps, select **Step_Count**.
13. Click **Done**.
14. To update the discounted gym membership fees on a contact's record detail page, click ⊕, and select **Action**, and from the Data section, select **Update Records**.
    a. For Label, enter `Update Gym Membership`.
    b. Select **Specify conditions to identify records, and set fields individually.**.
    c. For Object, select **Contact**.
    d. For Field, select **Id**.
    e. To map the record ID to the contact for whom the flow is run, for Value, select **ContactRecord**, and then select **Id**.
    f. To set field values for the Contact records, for the Field, select **Gym_Fees_c**.
    g. For Value, from the Actions section, select the expression set, and then select the **DiscountedGymFees** output variable.

    h. Click **Done**.
15. Save the flow.
    For Flow Label, enter `Calculate Discounted Gym Fees`.
16. To verify that the flow works as expected, click **Debug**.
    a. For ContactRecord, select a contact from your Salesforce org, and click **Run**.
    b. Enter a step count, and click **Calculate**.
       The step-by-step flow results appear in the Debug Details section. Here are the results of the sample data for the expression set action and update records steps.

17. Activate the flow.

## Example: Add the Flow to the Contact Record Page and Run the Flow

To enter the step count for a contact from the contact's record page, and pass the contact's record ID to the flow from the same page, add a flow component to the Contact Record Page.

REQUIRED EDITIONS

| USER PERMISSIONS NEEDED | |
|---|---|
| To modify a record page using the Lightning App Builder: | Customize Application |

1. Open the record page of a contact, and select **Related**.
2. From Setup, select **Edit Page**.

   The record page opens in Lightning App Builder.
3. From Components, drag Flow to an empty component on the page.
4. Configure the component.
   a. For Flow, search for and select **Calculate Discounted Gym Fees**.
5. Save and activate your changes.
6. To verify that the flow is added to the contact page, return to the Related tab of a contact record.

7. To verify that the flow is triggered and updates the Gym Fees field on the record, in the flow, for Step Count, enter `51000`.
   The Gym Fees field is populated with the discounted fees from the expression set's output that's returned to the flow.

## Example: View Decision Explainer Log History

View the action log for the expression set that was run from the flow for a contact record.

The contact record page must have Decision Explainer Log History standard component added.

1. From the record page of a contact, go to the Discounted Gym Membership Offer section.
2. For Step Count, enter **35,000**.

3. Click **Calculate**.
   The Gym Fees field is populated with the output of the expression set that calculates the discounted gym fees based on the contact's age and step count.
4. To see how the discounted fee amount was calculated, go to the Decision Explainer Log History component and, select the expression set run that returned the discounted fees.

5. Click **View**.

Step-wise details of how the discounted gym fees were calculated for the contact record's inputs are shown.

# Invocable Actions

You can invoke decision matrices and expression sets–the Business Rules Engine components–by using Apex or REST API.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

## REST API

- Invoke decision tables by using REST API
- Invoke decision matrices by using REST API
- Invoke expression sets by using REST API

## Apex Classes

To invoke expression set and decision matrix actions using Apex, use the Invocable namespace, which contains the required classes and methods. See Invocable Namespace.

To call expression sets that have a context definition, you must first hydrate the context definition and then use Context IDs that you use to hydrate context data in the payload. See Create a Context Definition

**Important** Apex runs in system context, which overrides the user's current permissions. Ensure that you manage user permissions to prevent users without the necessary permission sets from accessing expression sets and decision matrices.

**Example** Invoking an expression set with Apex:

```
Invocable.Action action = Invocable.Action.createCustomAction('runExpressi
onSet', 'ESTest');
action.setInvocationParameter('v1', 20);
action.setInvocationParameter('v2', 2);
List<Invocable.Action.Result> results = action.invoke();
```

Invoking an expression set with context definitions with Apex:

```
Invocable.Action action = Invocable.Action.createCustomAction('runExpressi
onSet', 'ESTest');
        action.setInvocationParameter('v1', 20);
        action.setInvocationParameter('v2', 2);
        action.setInvocationParameter('PricingContextId', <HydratedContext
Id>);
        List<Invocable.Action.Result> results = action.invoke();
```

Invoking a decision matrix with Apex:

```
Invocable.Action action = Invocable.Action.createCustomAction('runDecision
Matrix', 'SampleMatrix);
action.setInvocationParameter('v1', 20);
action.setInvocationParameter('v2', 2);
List<Invocable.Action.Result> results = action.invoke();
```

Invoking a decision table with Apex:

```
Invocable.Action action = Invocable.Action.createCustomAction('decisionTab
leAction', 'TestDTAsset_Default');
List<Map<String, Object>> bulkInput = new List<Map<String, Object>>();
        for(Integer i = 0; i<2; i++)
        {
            Integer newInput1 = 101;
            Integer newInput2 = 101;
            Map<String, Object> input = new Map<String, Object>();
            input.put('Price', newInput1);
            input.put('Quantity', newInput2);
            bulkInput.add(input);
        }
action.setInvocations(bulkInput);
List<Invocable.Action.Result> results = action.invoke();
system.debug(results);
```

# Third-Party Workflows

Business Rules Engine business APIs give you the flexibility of calling expression sets and lookup tables in workflows created using third-party tools of your choice.

REQUIRED EDITIONS

Available in: Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled

- Call an expression set
- Call a decision matrix
- Call a decision table

## Considerations for Calling Expression Sets

Before you call expression sets from Flows, Omnistudio, or third-party workflows, review these considerations.

REQUIRED EDITIONS

Available in: Lightning Experience

Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled

| Expression Set Component | Considerations When the Expression Set is Called |
|---|---|
| Input Variables | • If the value of the input variable is specified as null, the null value is considered even if the variable has a default value.<br>• If you don't want to specify a value for an input variable, exclude the variable from the input passed to the expression set. When you exclude a variable from the input, the default value of the variable that's configured in the expression set builder is considered.<br>**Note** If a default value isn't configured for the variable, then these system default values are considered:<br>- For variables of type Number, Currency, or Percent: 0<br>- For variable of type Boolean: False<br>- For variable of type Text: An empty string or "" |
| Field Aliases | If the expression set uses a field alias as a variable, pass the Id field of the source object, to which the field alias belongs, as an input to the calling |

| Expression Set Component | Considerations When the Expression Set is Called |
|---|---|
| | workflow. |
| Decision Matrix | The active decision matrix version with the highest rank is considered. |
| Sub expression | The active sub expression with the highest rank is considered. |

# Monitor Guardrail Limits with Apex

Fetch and monitor guardrail limits for your org using an Apex class. You can also invoke the Apex class in a flow.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Create three Apex classes to fetch and list guardrail limits for your org.

| APEX Class | Description |
|---|---|
| GuardrailOutput | Represents a single guardrail with various properties such as type, name, notification support, and value limits. It is designed to be used in Salesforce processes, flows, or Lightning components. |
| GuardrailListOutput | Represents a list of GuardrailOutput objects. It is used to group multiple guardrails together, making it easier to handle collections of guardrails in processes, flows, or Lightning components. |
| GuardrailAPI | Provides the invoke method to retrieve guardrail information for a list of components. The invoke method calls the getBREGuardrails method from the ConnectApi.BreGuardrailFamily class for each component, processes the results, and returns a list of GuardrailListOutput objects. |

You can use the Apex classes in a scheuled flow to routinely check guardrail

1. Click the Setup icon, and then click **Developer Console**.
2. To create a new Apex class, click **File | New | Apex Class**.
3. Create an Apex class with the name *GuardrailOutput*.

   Here's a sample Apex code for the GuardrailOutput apex class.

```
public class GuardrailOutput {
    @InvocableVariable
    @AuraEnabled
    global ConnectApi.GuardrailTypeEnumRepresentation guardrailType;
    @InvocableVariable
    @AuraEnabled
    global String name;
    @InvocableVariable
    @AuraEnabled
    global Boolean notificationSupported;
    @InvocableVariable
    @AuraEnabled
    global String minValue;
    @InvocableVariable
    @AuraEnabled
    global String maxValue;
    @InvocableVariable
    @AuraEnabled
    global String limitValue;
    @InvocableVariable
    @AuraEnabled
    global String currentValue;
}
```

4. Create another Apex class with the name **GuardrailListOutput**.
   Here's a sample Apex code for the GuardrailOutput Apex class.

```
public class GuardrailListOutput {
    @InvocableVariable
    @AuraEnabled
    global List<GuardrailOutput> guardrails;
}
```

5. Create another Apex class with the name **GuardrailAPI**.
   Here's a sample Apex code for the GuardrailOutput Apex class.

```
public class GuardrailApi {
```

```
    @InvocableMethod()
    global static List<GuardrailListOutput> invoke(List<String> components)
{
        List<GuardrailListOutput> allRes = new List<GuardrailListOutput>();
        for(String component : components) {
            ConnectApi.BreGuardrails result = ConnectApi.BreGuardrailFamil
y.getBREGuardrails(component, true);
            GuardrailListOutput res = new GuardrailListOutput();
            res.guardrails = new List<GuardrailOutput>();
            for(ConnectApi.Guardrails guardrailIt : result.result[0].guardra
ils) {
                GuardrailOutput go = new GuardrailOutput();
                go.guardrailType = guardrailIt.guardrailType;
                go.name = guardrailIt.name;
                go.notificationSupported = guardrailIt.notificationSupporte
d;
                go.minValue = guardrailIt.minValue;
                go.maxValue = guardrailIt.maxValue;
                go.limitValue = guardrailIt.limitValue;
                go.currentValue = guardrailIt.currentValue;
                res.guardrails.add(go);
                System.debug(go);
            }
            allRes.add(res);
        }
        return allRes;
    }
      }
```

6. Save your changes.
7. To send email to users in your org who have reached their guardrail limits, follow these steps:
   a. From Setup, find and select **Process Automation Settings**.
   b. Set up the workflow user and the organization-wide email addresses for your org.
   c. From Setup, go to **Flows**.
   d. To create a Schedule-Triggered Flow, click **New Flow**, then click **Scheduled**, then click **Schedule-Triggered Flow**.
   e. Set the **Start Date**, **Start Time**, and the **Frequency** of the flow.
   f. Click **Add** element, and select the **Action** element.
   g. In Search Actions, find and select the invocable method created.
   h. Add a loop element to iterate over all the guardrails.
      In the loop section, include the GuardrailListOutput Apex class, and select guardrails that have isNotificationEnabled set to True.
   i. Add a decision element to check if a guardrail has reached its limit and generate a text.
   j. After the loop ends, add a **Send Email** element to send an email.

# How to Bring External Data into Business Rules Engine

Review the methods and processes to bring data from your external systems into Business Rules Engine, and choose the method that best suits your business requirements.

## Role of Business Rules Engine

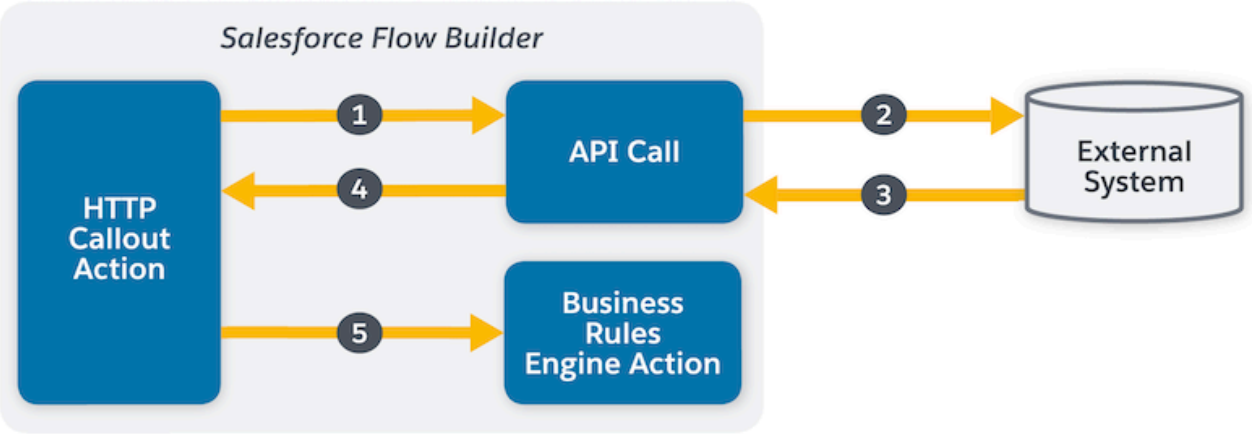
Business Rules Engine doesn't inherently provide processes or methods to fetch external data. Instead, it offers components such as lookup tables and expression sets that you can use to perform complex lookups and calculations, and create business rules. Before you invoke Business Rules Engine components, use other Salesforce products and services to extract data from external sources.

**Note** All customer data coming from external sources into Salesforce adheres to the Salesforce standard data compliance and security protocols. See Salesforce Security Guide.

Based on the architecture that you've designed for your digital processes, choose one of these methods to bring external data into Salesforce and run rules by using Business Rules Engine.

## Method 1: Use Salesforce Flow

If you want to use Salesforce Flow to automate your business processes and need external data to run rules by using Business Rule Engine, create an HTTP callout action that calls a web-based service API. You can do this in Flow Builder without code. After you add the API details, Flow Builder generates a reusable callout action that you can use right away in Flow Builder. You can also edit an HTTP callout action declaratively and configure the inferred data structure. The API call fetches the data from your external system and makes it available in flow. Then, use an action element in flow to invoke Business Rules Engine components such as expression sets and lookup tables, and pass the data to these components. See HTTP Callout.



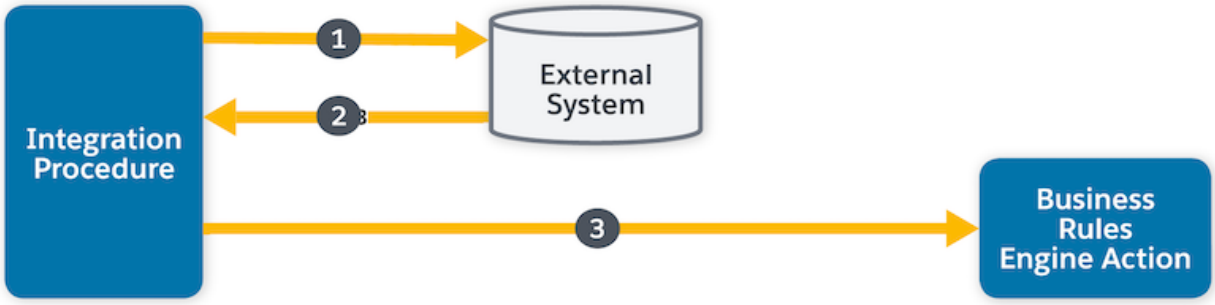
## Method 2: Use Salesforce Custom Object

Create a Salesforce custom object and fields, and upload your external data as CSV files. You can also use any of the other data import methods available to sync data between your external systems and Salesforce custom object. Then, use the data from the custom object to run rules in Business Rules Engine. See Salesforce Data Import.

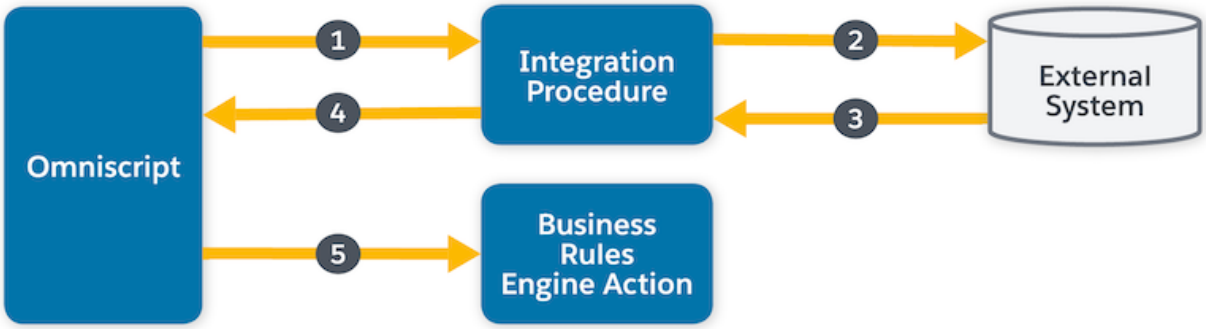
## Method 3: Use Omnistudio Integration Procedures

Omnistudio Integration Procedures can read and write data from external systems via REST calls. Both Omniscripts and Integration Procedures can invoke Business Rules Engine components and pass external data to them, but the process differs slightly for each.

- Integration Procedures: Fetch external data via REST calls, then invoke Business Rules Engine components and pass the data to these components.




- Omniscripts: Call an Integration Procedure to fetch external data, then invoke the Business Rules Engine components and pass the data to these components.

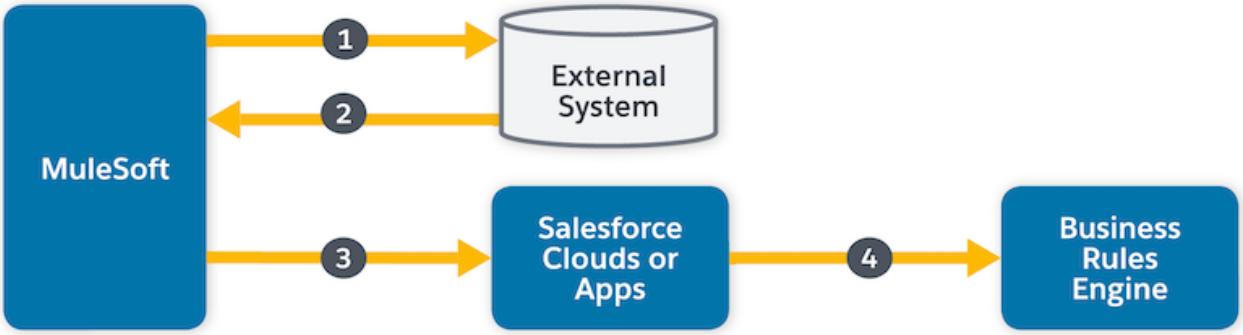



There are prebuilt integrations with Business Rules Engine components that you can call from an Omniscript or Integration Procedure with invocable actions. See Omnistudio Integration Procedures.

## Method 4: Use MuleSoft Integration

You can use MuleSoft Integration to bring data from outside Salesforce into the Business Rules Engine. Create a flow in your MuleSoft application that connects to the external system and retrieves the necessary data. Then, use any of the Salesforce Cloud processes or apps to call the external data from

MuleSoft and pass it to Business Rules Engine. With MuleSoft, you can automate data flow from external systems into Business Rules Engine and create business rules to process the data according to your business logic. See MuleSoft Documentation.

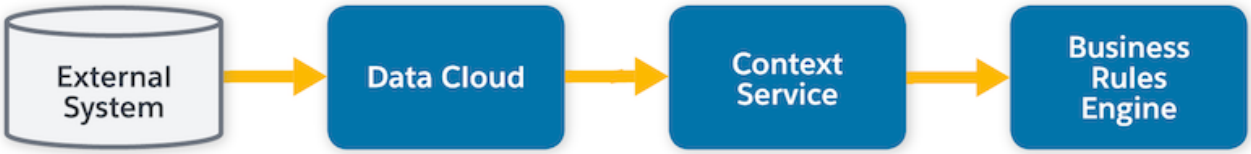



## Method 5: Use Apex

If you have a custom integration scenario that requires precise control over how you fetch, transform, and insert data into Salesforce, use Apex to create custom integrations with external systems. You can use Apex to call external APIs, retrieve data, and insert or update records in Salesforce. In Apex, you can use the Invocable namespace feature to invoke expression set and lookup table actions. Invocable Namespace contains the required classes and methods to invoke actions from Apex. To know more about invoking Business Rules Engine components in Apex, see Invocable Actions.

## Method 6: Use Data Cloud and Context Service

Data Cloud is an integrated platform to bring together data from disparate sources. Use connectors to ingest external data into Data Cloud. Then, use Context Service to map the xml data from Data Cloud to Business Rules Engine at run time and run business rules. To explore how external data is ingested into Data Cloud, see Connect an External Data Source to Salesforce Data Cloud.




# Calculation Matrix and Procedure Migration to Business Rules Engine

To use calculation procedures and calculation matrices from Omnistudio as Business Rules Engine components, migrate these Omnistudio components as expression sets and decision matrices.

REQUIRED EDITIONS

To migrate a calculation procedure as an expression set, and a calculation matrix as a decision matrix, use the CalculationMatrixMigrationService and CalculationProcedureMigrationService Apex classes.

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

**Considerations for Migrating Calculation Matrices and Procedures**
Before you migrate your calculation matrices and calculation procedures as decision matrices and expression sets respectively, review these migration recommendations and the post-migration behavior of these components.

**Migration Scenarios and Workflow**
Let's look at a few scenarios for migrating calculation matrices as decision matrices and calculation procedures as expression sets. Let's also review the migration workflow for various states of the calculation matrices and calculation procedures.

**Migration Limitations**
Before you migrate calculation matrices and procedures as decision matrices and expression sets, consider these migration limitations.

# Considerations for Migrating Calculation Matrices and Procedures

Before you migrate your calculation matrices and calculation procedures as decision matrices and expression sets respectively, review these migration recommendations and the post-migration behavior of these components.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

## Migration Recommendations

- If a calculation matrix is used in a calculation procedure, avoid changing the matrix name and the matrix column names. Modifying a matrix that's referenced in a calculation procedure results in a variable name mismatch between the matrix and the procedure.
- Ensure that you don't change the matrix column data types when the matrix is referenced in a calculation procedure.
- Ensure that the matrix's column names don't have spaces and special characters.
- Ensure that the variable names in the calculation procedure don't have spaces and special characters.

- Migrate calculation procedures and matrices as expression sets and decision matrices respectively to a sandbox environment first. If everything works as expected, deploy the expression sets and decision matrices from the sandbox environment to the production environment.
- If a calculation procedure references a calculation matrix, first migrate the calculation matrix. For more information, see Migration Scenarios and Workflow.

## Post-Migration Behavior

- The migrated expression set and decision matrix versions are inactive.
- When a calculation procedure from a managed package is migrated as an expression set, the expression set is an unmanaged component.
- When a calculation matrix from a managed package is migrated as a decision matrix, the decision matrix is an unmanaged component.
- The calculation procedures and calculation matrices in the managed packages remain unchanged.

# Migration Scenarios and Workflow

Let's look at a few scenarios for migrating calculation matrices as decision matrices and calculation procedures as expression sets. Let's also review the migration workflow for various states of the calculation matrices and calculation procedures.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

Migration Scenarios and Workflow

| Scenario | Workflow |
|---|---|
| Migrate a calculation procedure as an expression set. The calculation procedure is standalone. | Migrate the calculation procedure as an expression set. |
| Migrate a calculation matrix as a decision matrix. The calculation matrix is standalone and isn't used in a calculation procedure. | 1. Identify the versions of the calculation matrix that have data.<br>2. Download the data from the identified calculation matrix versions as CSV files.<br>3. Migrate the calculation matrix as a decision matrix.<br>**Note** Only column headers are migrated.<br>4. To add row data to the migrated decision |

| Scenario | Workflow |
| --- | --- |
| | matrix, upload the CSV file that contains the calculation matrix data to the relevant decision matrix version.<br>5. Activate the decision matrix version that you want to use. |
| Migrate a calculation procedure as an expression set. The calculation procedure references a calculation matrix. | 1. Identify the active version of the calculation matrix that's used in the calculation procedure.<br>2. Download the data from the active calculation matrix version as a CSV file.<br>3. Migrate the calculation matrix as a decision matrix.<br>**Note** Only column headers are migrated.<br>4. To add row data to the migrated decision matrix, upload the CSV file that contains the calculation matrix data to the relevant decision matrix version.<br>5. Activate the decision matrix version that you want to use or activate at least one version of the migrated decision matrix so that the calculation procedure can be migrated as an expression set.<br>6. Migrate the calculation procedure as an expression set. |

## Migration Limitations

Before you migrate calculation matrices and procedures as decision matrices and expression sets, consider these migration limitations.

REQUIRED EDITIONS

| |
| --- |
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

A calculation procedure can't be migrated as an expression set if:

- The calculation procedure uses looping.
- The calculation procedure uses custom functions.
- The calculation procedure references a decision matrix that isn't migrated as a decision matrix.
- The calculation procedure references a calculation matrix that has no active versions.
- The number of input variables and output variables and the data types that are used in the calculation procedure have changed because of changes to the calculation matrix that the calculation procedure references.

## Business Rules Engine Use Cases

Explore various use cases that shows how Business Rules Engine is used across different clouds.

| Product | Usage | Help LINK |
|---|---|---|
| Public Sector Solutions | Uses Business Rules Engine to automatically calculate processing fees associated with business license applications | Automatically Calculate Application Fees |
| Public Sector Solutions | Uses Business Rules Engine to build automation that creates a regulatory transaction fee from an inspection visit or a violation enforcement action in Public Sector Solutions. | Automatically Create a Fee for an Inspection or Enforcement Action |
| Financial Services - Digital Lending | Uses prebuilt expression set to automate decision-making processes, such as categorize risk profiles, determine maximum eligible loan amounts for secured and unsecured loans, and present relevant offers. | Decision Matrices and Expression Sets for Digital Lending–India |
| Manufacturing | Uses Flow Builder, Business Rules Engine, and Context Service to set up an automated claim adjudication process. | Automated Adjudication of Warranty Claims |

**How to Use Business Rules Engine to Preapprove Health Cloud Care Requests**
Configure and test the Business Rules Engine features by using a Health Cloud Prior Authorization of Care Requests scenario.

# How to Use Business Rules Engine to Preapprove Health Cloud Care Requests

Configure and test the Business Rules Engine features by using a Health Cloud Prior Authorization of Care Requests scenario.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise** and **Unlimited** Editions for clouds that have Business Rules Engine enabled |

A Health Cloud customer is looking for an efficient end-to-end authorization process for patients to deliver timely care services. To achieve this, the customer's authorization coordinator has to:

- Retrieve patient information from Electronic Health Records (EHR)
- Check the need for authorization for the requested services or diagnosis from the payer
- Create and submit authorization requests, with necessary details and documents, to the payer based on the defined requirements.
- Submit and follow up on the approval process with the payer
- Complete the process in a timely manner so that patient care is delivered as planned

Health Cloud's Utilization Management feature provides the capabilities of centralized and guided workflows with multiple Omnistudio components and Business Rules Engine. It empowers users to easily manage healthcare prior authorization request submissions and reviews.

Before you begin, make sure that you have these settings and product licenses in your org:

- Health Cloud: Sign up for a Health Cloud Learning Org.
- Business Rules Engine
- Context Service
  - From Setup, in the Quick Find box, enter `Context Service Settings`, and then select **Context Service Settings**.
  - Enable the **Context Definitions** toggle.

**Note** This guide is based on the Salesforce Summer '24 release.

### Create Health Cloud Accounts

Before you set up Business Rules Engine features with Health Cloud, create a few accounts with sample data that you can use when you configure the scenario.

### Create Plans and Benefits

Create Member Plans, Purchaser Plans, Coverage Benefits, and Coverage Benefit Items to ensure that the Health Cloud prior authorization workflow accurately reflects the rules and coverage details

required to assess user eligibility and calculate benefits. These objects are the data sources for context definition and Business Rules Engine.

**Create a Context Definition for Health Cloud Objects**
Use Context Definition nodes and attributes to create a context definition for the Health Cloud objects and fields. Then, map the nodes with the corresponding objects, and map the attributes with the corresponding fields.

**Create a Decision Matrix for Benefit Service Codes**
Create a decision matrix to assign the prior authorization required status for the service codes defined in the Health Cloud objects and also for a few undefined ones.

**Create an Expression Set for Prior Authorization Determination**
Use an expression set to create a rule to determine the prior authorization and member plan status of a service. Add a List Group element in the expression set to fetch the Health Cloud object fields from the context definition that you created earlier. Based on the prior authorization and member plan status, calculate the coverage benefit associated with the service and its approval status.

**Add Decision Explainer Messages to Prior Authorization Requests**
Create expression set message templates where you add the success and failure explanation messages for the elements you want to show the explanations. Then, assign these templates to the respective elements in the Expression Set Builder.

**Create a Flow for Prior Authorization Check**
A Salesforce Flow brings the Context Service and Business Rules Engine components on a single platform and helps you run the scenario to experience how these components work together to give the expected results. Create a flow to test the Health Cloud scenario and verify if you're getting the expected results. In this flow, you first add a screen component to capture the user inputs. Then, use the Get Records elements to fetch the Context IDs that you use to hydrate context data. Add a loop element to find the prior authorization and benefit details for each service code. Use a Build Context action element inside the loop to hydrate data from the Member Plan object and its child nodes. Then, add an action element to call the expression set that uses the hydrated data to get the prior authorization check status for each service code and the corresponding coverage benefit details and approval status.

**Run Health Care Prior Authorization Check Scenario**
Run the Health Care Prior Authorization Check flow to execute the Prior Authorization scenario and experience how Business Rules Engine performs calculations and provides the results.

## Create Health Cloud Accounts

Before you set up Business Rules Engine features with Health Cloud, create a few accounts with sample data that you can use when you configure the scenario.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise** and **Unlimited** Editions for clouds that have Business Rules Engine enabled |

Create a healthcare provider Payer account with these details. The Payer account type represents the healthcare providers.

1. From the App Launcher (⠿), find and select **Accounts**.
2. Click **New**.
3. Select the **Payer** account type.
4. Add these details.
   - Account Name: `Cigna International`
   - Phone: `+1 868 7899 9090`
   - Fax: `+1 868 7899 9999`
   - Website: `https://www.cigna.com`
   - Employees: `5000`
   - Industry: `Healthcare`
   - Annual Revenue: `300000000000`
   - Billing City: `New Jersey`
   - Billing Zip/Postal Code: `70854`
   - Billing State/Province: `New Jersey`
   - Billing Country: `United States`
   - Shipping City: `New Jersey`
   - Shipping Zip/Postal Code: `78656`
   - Shipping State/Province: `New Jersey`
   - Shipping Country: `United States`
5. Save your changes.
6. Similarly, create two Person accounts for Victor Kent and Steve Smith with these details. A person account record combines an account and a contact to represent a person without the need to relate that person to an organization.

| Fields | Victor Kent | Steve Smith |
|---|---|---|
| Salutation | Mr. | Mr. |
| First Name | Victor | Steve |
| Last Name | Kent | Smith |
| Phone | +1 868 7899 9090 | +1 868 7899 7676 |
| Fax | +1 868 7899 9094 | +1 868 7899 7777 |
| Billing City | New York | New York |
| Billing State/Province Code | New York | New York |
| Shipping City | - | New Jersey |
| Shipping State/Province Code | - | New Jersey |

## Create Plans and Benefits

Create Member Plans, Purchaser Plans, Coverage Benefits, and Coverage Benefit Items to ensure that the Health Cloud prior authorization workflow accurately reflects the rules and coverage details required to assess user eligibility and calculate benefits. These objects are the data sources for context definition and Business Rules Engine.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise** and **Unlimited** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create member plan records: | Health Cloud Foundation (for Health Cloud) permission set |
| To create purchaser plan records: | Health Cloud Foundation (for Health Cloud) permission set |
| To create a coverage benefit: | Health Cloud Starter OR Health Cloud Foundation |
| To create a coverage benefit item: | Health Cloud Starter OR Health Cloud Foundation |

## Purchaser Plan

1. From the App Launcher (⠿), find and select **Purchaser Plans**.
2. Click **New**.
3. Add these details.
   - Name: Cigna Medicare Purchaser Plan
   - Payer: Cigna International
   - Plan Number: P1234567
   - Plan Type: Medicare
   - Plan Status: Active
   - Effective From: [Current Date]
   - Effective To: [Future Date]
4. Save your changes.

## Member Plan

1. From the App Launcher (⠿), find and select **Member Plans**.
2. Click **New**.
3. Add these details.
   - Name: *Cigna Medicare Advantage Plan*
   - Member: *Steve Smith*

- Member Number: `675858989`
- Group Number: `7878787878`
- Plan: `Cigna Medicare Purchaser Plan`
- Payer: `Cigna International`
- Subscriber: `Victor Kent`
- Effective From: `[Current Date]`
- Effective To: `[Future Date]`
- Status: `Active`
- Verification Status: `Active - Verified`
- Relationship to Subscriber: `Spouse`

4. Save your changes.

## Coverage Benefit

1. From the App Launcher (⠿), find and select **Coverage Benefit**.
2. Click **New**.
3. Add these details.
   - Name: `Medicare Coverage Benefit`
   - Member: `Steve Smith`
   - Member Plan: `Cigna Medicare Advantage Plan`
4. Save your changes.

## Coverage Benefit Item

1. From the App Launcher (⠿), find and select **Coverage Benefit Item**.
2. Click **New**.
3. Add these details.
   - Name: `Cardiovascular Monitoring Services`
   - Member: `Steve Smith`
   - Coverage Benefit: `Medicare Coverage Benefit`
   - Service Type Code: `CPT23456`
   - Save your changes.
4. Similarly, create coverage benefit items with these details.

| Name | Member | Coverage benefit | Service Type Code |
|---|---|---|---|
| Echocardiography Procedures | Steve Smith | Medicare Coverage Benefit | CPT23457 |
| Non-invasive Physiologic Studies and Procedures | Steve Smith | Medicare Coverage Benefit | CPT23458 |
| TMT Test | Steve Smith | Medicare Coverage Benefit | CPT23459 |

## Create a Context Definition for Health Cloud Objects

Use Context Definition nodes and attributes to create a context definition for the Health Cloud objects and fields. Then, map the nodes with the corresponding objects, and map the attributes with the corresponding fields.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise** and **Unlimited** Editions for clouds that have Context Service enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To load, view, edit, and delete context definitions: | Context Service Admin |

1. From Setup, in the Quick Find box, enter `Context Service`, and then select **Context Definitions**.
2. Click **New**.
3. Add these details.
   a. Enter `Get_MemberPlan_Details` as the name.
   b. In Effective From, enter `[Current Date and Time]`.
   c. In Time to Live, enter `10` as the duration, and select **Minute** as the unit.
   d. Click **Next**.
4. Configure a structure to define nodes and the relationship between the nodes.
   a. In Add Node, enter `MemberPlan`.
   b. Click beside the MemberPlan node to add a child node. Enter `CoverageBenefit`.
   c. Click beside the CoverageBenefit node to add a child node. Enter `CoverageBenefitItem`.

5. Click **Next**.

## Add Attributes

1. Click **Add Attributes**, and enter these attribute details for each node.

Node: MemberPlan

| Attribute Name | Type | Data Type |
|---|---|---|
| Status | INPUT OUTPUT | STRING |
| EffectiveFrom | INPUT OUTPUT | DATE |
| EffectiveTo | INPUT OUTPUT | DATE |
| isActive | INPUT OUTPUT | BOOLEAN |
| PlanId | INPUT | STRING |
| Name | INPUT | STRING |
| MemberNumber | INPUT | STRING |

Child Node: CoverageBenefit

| Attribute Name | Type | Data Type |
|---|---|---|
| IsActive | INPUT OUTPUT | BOOLEAN |
| CoverageType | INPUT OUTPUT | STRING |
| BenefitPeriodEndDate | INPUT OUTPUT | DATE |
| BenefitPeriodStartDate | INPUT OUTPUT | DATE |
| Name | INPUT OUTPUT | STRING |
| MemberPlanId | INPUT OUTPUT | STRING |
| MemberId | INPUT OUTPUT | STRING |

Child Node: CoverageBenefitItem

| Attribute Name | Type | Data Type |
|---|---|---|
| ServiceTypeCode | INPUT OUTPUT | STRING |
| Name | INPUT OUTPUT | STRING |
| ServiceType | INPUT OUTPUT | STRING |
| IsPreauthorizationRequired | INPUT OUTPUT | BOOLEAN |
| Cost | INPUT | STRING |

2. Click **Next**.

## Add Tags

1. Click **Add Tags** beside the MemberPlan node. Enter `MemberPlan`, and click **Done**.
2. Similarly, click **Add Tags** beside each of these attributes and enter the corresponding tag names. Click **Done** after you add each tag.

   Node: MemberPlan

| Attribute Name | Tag name |
|---|---|
| Status | PlanStatus |
| EffectiveFrom | EffectiveDateFrom |
| EffectiveTo | EffectiveDateTo |
| isActive | IsMemberPlanActive |
| PlanId | PlanId |
| Name | MemberName |
| MemberNumber | MemberNumber |

   Child Node: CoverageBenefit

| Attribute Name | Tag name |
|---|---|
| CoverageBenefit | CoverageBenefit |
| IsActive | IsCoverageBenefitActive |
| CoverageType | CoverageType |
| BenefitPeriodEndDate | BenefitPeriodEndDate |
| BenefitPeriodStartDate | BenefitPeriodStartDate |
| Name | CoverageBenefitName |
| MemberPlanId | MemberPlanId |
| MemberId | MemberId |
| ParentReference | Plan |

   Child Node: CoverageBenefitItem

| Attribute Name | Tag name |
|---|---|
| CoverageBenefitItem | CoverageBenefitItem |
| ParentReference | CB |

| Attribute Name | Tag name |
|---|---|
| ServiceTypeCode | ServiceTypeCode |
| Name | CoverageBenefitItemName |
| ServiceType | ServiceType |
| IsPreauthorizationRequired | IsPreauthorizationRequired |
| Cost | Cost |

3. Save your changes.

## Add Context Mapping

1. Click the **Get_MemberPlan_Details** context definition to open it.
2. Click the **Map Data** tab.
3. Click **Add Mapping**.
4. Enter `Map1` as the name of the mapping.
5. Select **Mark as Default**.
6. Click **Next**.
7. In Mapping Intent, select **Association**, **Hydration**, **Persistence**, and **Translation**.
8. Click **Map**.
9. On the Context Mapping builder page, click **Select Data Source**.
10. In the Select an object search field, enter `Member Plan`, and select the **Member Plan** object from the dropdown.
11. Similarly, find and select the **Coverage Benefit** and **Coverage Benefit Item** objects.
12. To select each object and add them to the map, click .
13. Click **Done**.
14. In the Structure: Get_MemberPlan_Details section, click beside the MemberPlan node. Then, click beside the MemberPlan object under the Connect Objects section and connect the node and object.

    **Note** Map the Nodes first before you map the attributes under each node.

15. Click beside Unmapped under MemberPlan in both the Structure and Connect Objects sections to show all the attributes and fields, respectively.
16. Click beside the MemberNumber attribute in the Structure section, and then click beside the MemberNumber field in the Connect Objects section to connect them.
17. Similarly, connect all the nodes and attributes in the Structure section with the corresponding objects and fields in the Connect Objects section, respectively.

18. Save your changes.
19. Open the Get_MemberPlan_Details context definition.
20. Click the **Map Data** tab.
21. Click the dropdown for the Map1 mapping record, and select **Input Mapping**.

22. In the MemberPlan node, for isActive, enter `isActive` in the Input Mapping field.
23. Save your changes.
24. On the Custom Definitions tab, click the dropdown for the Get_MemberPlan_Details context definition item, and click **Activate**.

## Create a Decision Matrix for Benefit Service Codes

Create a decision matrix to assign the prior authorization required status for the service codes defined in the Health Cloud objects and also for a few undefined ones.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise** and **Unlimited** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete decision matrices and matrix versions: | Rule Engine Designer |
| To use decision matrices in Business Rules Engine: | Rule Engine Runtime |

1. From the App Launcher (⠿), find and select **Business Rules Engine**.
2. Click the app navigation menu, and then select **Lookup Tables**.
3. Click **New**.
4. Select **Decision Matrix**, and then click **Next**.
5. Enter these details.
   - Name: Benefit Service Code
   - Type: Standard
6. Save your changes.

## Add Columns and Rows

1. On the Benefit Service Code decision matrix record page, click the **Related** tab.
2. On the Decision Matrix Versions related list, click the **Benefit Service Code V1** decision matrix version.
3. On the Matrix tab, click **Add Column**.
4. Enter these details.
   - Column Header: ServiceCode

- Header Type: Input
- Data Type: Text

5. Click **Add Column**.
6. Enter these details for the second column.
   - Column Header: PriorAuthorizationRequired
   - Header Type: Output
   - Data Type: Boolean

7. Click **Done**.
8. Click **Add Row**.
9. In the ServiceCode column, hover over the first row, and click .
10. Enter `CPT23456` as the service code.
11. In the PriorAuthorizationRequired column, hover over the first row, and click .
12. To ensure prior authorization for the corresponding service code, select the checkbox.
13. Similarly, add the remaining service codes and their prior authorization required status.

| ServiceCode | PriorAuthorizationRequired |
|---|---|
| CPT23457 | Yes |
| CPT23458 | Yes |
| CPT23459 | No |
| CPT45678 | Yes |
| CPT67890 | No |

14. Save your changes.
15. Click  on the top right, and select **Activate**.

## Create an Expression Set for Prior Authorization Determination

Use an expression set to create a rule to determine the prior authorization and member plan status of a service. Add a List Group element in the expression set to fetch the Health Cloud object fields from the context definition that you created earlier. Based on the prior authorization and member plan status, calculate the coverage benefit associated with the service and its approval status.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise** and **Unlimited** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create, update, and delete expression sets | Rule Engine Designer |
| To run expression sets in Business Rules Engine | Rule Engine Runtime |

1. From the App Launcher , find and select **Expression Sets**.
2. Click **New**.
3. Enter `Member Plan Status Check` as the name of the new expression set.
4. Select **Default** as the usage type.
5. Select **Get_MemberPlan_Details** as the context definition.
6. Save your changes.

## Create Resources

1. On the Member Plan Status Check expression set record page, click the **Member Plan Status Check V1** expression set version to open it in Expression Set Builder.
2. In the Expression Set Builder, open Resource Manager by clicking .
3. Click **Add Resource**.
4. To create the Date of Service variable, enter these details.
   - Resource Type: Variable
   - Resource Name: DateOfService
   - Data Type: Date
5. Click **Done**.
6. Similarly, create these variables.

| Resource Type | Resource Name | Data Type |
|---|---|---|
| Variable | MemberNumberInput | Text |
| Variable | MemberPlanStatus | Boolean |
| Variable | PriorAuthDetermination | Text |

7. Save your changes.

## Add Elements

1. In the Expression Set Builder canvas, click .
2. Find and select **Lookup Table**.
3. In the Lookup Table Details field, find and select the **Benefit Service Code** decision matrix.

4. Save your changes.

5. Click after the Benefit Service Code lookup table element.
6. Find and select **List Group**.
7. Add these details in the List Filter element.
   a. Rename the List Filter element as `Check Member Plan Status`.
   b. In Filter Condition Requirements, select **All Conditions Are Met (AND)**.
   c. Select **MemberNumber** as the resource.
   d. Select **Equals** as the operator. Select **MemberNumberInput** as the value.
   e. Click **Add Condition**.
   f. Select **PlanStatus** as the resource.
   g. Select **Equals** as the operator.
   h. Enter `Active` as the value.
   i. Click **Add Condition**.
   j. Select **EffectiveDateFrom** as the resource.
   k. Select **Less Than** as the operator.
   l. Select **DateOfService** as the value.
   m. Click **Add Condition**.
   n. Select **EffectiveDateTo** as the resource.
   o. Select **Greater Than** as the operator.
   p. Select **DateOfService** as the value.
8. Click inside the List Group box.
9. Find and select **Calculation**.
10. Add these details in the calculation element.
    a. Rename the Calculation element as `Member Plan Active`.
    b. Enter `true` as the formula.
    c. Select **isMemberPlanActive** as the output variable.
11. Click inside the List Group box.
12. Find and select **Calculation**.
13. Add these details in the calculation element.
    a. Rename the Calculation element as `Member Plan Status`.
    b. Select **isMemberPlanActive** as the formula.
    c. Select **MemberPlanStatus** as the output variable.
14. Save your changes.

15. Click outside the List Group box.
16. Find and select **List Group**.
17. Add these details in the List Filter element.
    a. Rename the List Filter element as `Check Requested Code with Plan Condition 1`.
    b. In Filter Condition Requirements, select **Any Condition Is Met (OR)**.
    c. Select **IsPreauthorizationRequired** as the resource.
    d. Select **Equals** as the operator.
    e. Enter `false` as the value.

f. Click **Add Condition**, and add two more conditions with these details.

| Resource | Operator | Value |
|---|---|---|
| Benefit_Service_Code__PriorAuthorizationRequired | Equals | false |
| ServiceTypeCode | Not Equals | ServiceCode |

18. Click inside the List Group box.
19. Find and select **Calculation**.
20. Add these details in the calculation element.
    a. Rename the Calculation element as `Prior Authorization Not Required`.
    b. Enter `ServiceCode + " " + CoverageBenefitItemName + " - Administrative Not Approved, PriorAuth Not Required or Service Code is not covered for Benefit."` as the formula.
    c. Select **PriorAuthDetermination** as the output variable.
21. Save your changes.
22. Click outside the List Group box.
23. Find and select **List Group**.
24. Add these details in the List Filter element.
    a. Rename the List Filter element as `Check Requested Code with Plan Condition 2`.
    b. In Filter Condition Requirements, select **Custom Condition Logic is Met**. For Custom Condition Logic, enter `(1 OR 2) AND 3`.
    c. Select **IsPreauthorizationRequired** as the resource.
    d. Select **Equals** as the operator.
    e. Enter `true` as the value.
    f. Click **Add Condition**, and add two more conditions with these details.

| Resource | Operator | Value |
|---|---|---|
| Benefit_Service_Code__PriorAuthorizationRequired | Equals | true |
| ServiceTypeCode | Equals | ServiceCode |

25. Click inside the List Group box.
26. Find and select **Calculation**.
27. Add these details in the calculation element.
    a. Rename the Calculation element as `Prior Authorization Approval`.
    b. Enter `ServiceCode + " " + CoverageBenefitItemName + " - Administrative Approved"` as the formula.
    c. Select **PriorAuthDetermination** as the output variable.
28. Save your changes.

29. Click in the Expression Set Builder page.
30. Enter *1* as the rank.
31. Select the **Prior Authorization Approval** element.
32. Click in the Expression Set Builder page.
33. Select **Include in Output**.
34. Similarly, select **Include in Output** for all the calculation elements and lookup table element in the expression set.
35. Save your changes.
36. Click **Activate**.

## Add Decision Explainer Messages to Prior Authorization Requests

Create expression set message templates where you add the success and failure explanation messages for the elements you want to show the explanations. Then, assign these templates to the respective elements in the Expression Set Builder.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise** and **Unlimited** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create and modify explainability message templates: | Decision Explainer Service Access |
| To use explainability message templates in expression set steps: | Rule Engine Designer |
| To run expression sets that have Decision Explainer enabled: | Rule Engine Runtime |

## Create Expression Set Message Tokens

1. From Setup, in the Quick Find box, enter `Decision Explainer`, and then select **Expression Set Message Token**.
2. Click **New Expression Set Message Token**.
3. Enter the expression set message token details.
   - Name: BenefitServiceCode.
   - Label: BenefitServiceCode.
4. Save your changes.

## Create Explainability Message Templates

1. From Setup, in the Quick Find box, enter `Decision Explainer`, and then select **Explainability Message Template**.
2. Click **New Explainability Message Template**.
3. Enter the explainability message template details.
   - Name: BenefitServiceCode_Success.
   - Message: Prior authorization requirement for your service {BenefitServiceCode} is determined.
   - Expression Set Step Type: Decision Matrix Lookup.
   - Usage Type: Default. Label: BenefitServiceCode_Success.
   - Result Type: Passed.
4. Save your changes.
5. Similarly, create these explainability message templates.

| Name | Message | Expression Set Step Type | Usage Type | Label | Result Type |
|---|---|---|---|---|---|
| BenefitServiceCode_Failure | Sorry! Prior authorization requirement for your service {BenefitServiceCode} could not be determined. | Decision Matrix Lookup | Default | BenefitServiceCode_Failure | Failed |
| MemberPlanStatus_Success | Your Member Plan activation status is Active. | Calculation | Default | MemberPlanStatus_Success | Passed |
| MemberPlanStatus_Failure | Your Member Plan activation status is Inactive. | Calculation | Default | MemberPlanStatus_Failure | Failed |
| Approve_Success | This Member is Approved and PreAuthorised. | Calculation | Default | Approve_Success | Passed |
| Approve_Fail | PreAuthorisation could not be determined for this member. | Calculation | Default | Approve_Fail | Failed |

## Add Decision Explanations to Expression Set Elements

1. From the App Launcher , find and select **Expression Sets**.
2. Click **Member Plan Status Check**.
3. On the Member Plan Status Check expression set record page, click the **Member Plan Status Check V1** expression set version to open it in Expression Set Builder.
4. Click **Deactivate**.
   You must deactivate the expression set to make any changes.
5. Select the **Benefit Service Code** lookup table element.
6. Click on the left panel in the Expression Set Buildercanvas.
7. Select **Show decision explanation**.
8. Save your changes.
9. Click on the left panel in the Expression Set Builder canvas.
10. Select **Show decision explanation**.
11. In the Decision Explainer section, add these details.
    a. In When Step Returns Output, select **BenefitServiceCode_Success**.
    b. In When Step Errors, select **BenefitServiceCode_Failure**.
12. Click beside the When Step Returns Output field. Under Map Tokens, in BenefitServiceCode, select **ServiceCode**.
13. Click **Done**.
14. Click beside the When Step Errors field. Under Map Tokens, in BenefitServiceCode, select **ServiceCode**.
15. Click **Done**.
16. Save your changes.
17. Select the **Member Plan Active** calculation element, and click on the left panel.
18. Select **Show decision explanation**.
19. In the Decision Explainer section, add these details.
    a. In When Step Returns Output, select **MemberPlanStatus_Success**.
    b. In When Step Errors, select **MemberPlanStatus_Failure**.
20. Select the **Prior Authorization Approval** calculation element, and click on the left panel.
21. Select **Show decision explanation**.
22. In the Decision Explainer section, add these details.
    a. In When Step Returns Output, select **Approve_Success**.
    b. In When Step Errors, select **Approve_Fail**.
23. Save your changes.
24. Click **Activate**.

## Simulate the Expression Set and View the Decision Explanation

Now, simulate the expression set and verify if the decision explanations you added earlier are appearing as expected. To simulate the expression set with context data, manually add the context input data as JSON inputs. Context data doesn't hydrate automatically when you simulate an expression set.

1. From the App Launcher , find and select **Expression Sets**.
2. Click **Member Plan Status Check**.
3. On the Member Plan Status Check expression set record page, click the **Member Plan Status Check V1** expression set version to open it in Expression Set Builder.
4. Click **Simulate**.
5. Select **Advanced** as the input mode.
6. In the JSON Input box, remove any existing data and add this sample test input data manually. You can copy and paste this code in the JSON Input box.

```
{
  "ServiceCode": "CPT23456",
  "MemberNumberInput": "675858989",
  "DateOfService": "2024-09-26",
  "MemberPlan": [
    {
      "EffectiveDateTo": "2025-08-22",
      "EffectiveDateFrom": "2024-08-22",
      "PlanStatus": "Active",
      "CoverageBenefit": [
        {
          "CoverageBenefitItem": [
            {
              "IsPreauthorizationRequired": "false",
              "CoverageBenefitItemName": "XYZ",
              "ServiceTypeCode": "CPT23456"
            }
          ]
        }
      ],
      "MemberNumber": "675858989"
    }
  ]
}
```

7. Click **Simulate**.
8. From the **Detailed View** dropdown, select **Concise View**.

You can now view the decision explanations for the Benefit Service Code lookup table element, and the Member Plan Active and Prior Authorization Approval calculation elements. The decision explanations are determined for the service code CPT23456 that was provided as input.

## Create a Flow for Prior Authorization Check

A Salesforce Flow brings the Context Service and Business Rules Engine components on a single platform and helps you run the scenario to experience how these components work together to give the expected results. Create a flow to test the Health Cloud scenario and verify if you're getting the expected results. In this flow, you first add a screen component to capture the user inputs. Then, use the Get Records elements to fetch the Context IDs that you use to hydrate context data. Add a loop element to find the prior authorization and benefit details for each service code. Use a Build Context action element inside the loop to hydrate data from the Member Plan object and its child nodes. Then, add an action element to call the expression set that uses the hydrated data to get the prior authorization check status for each service code and the corresponding coverage benefit details and approval status.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise** and **Unlimited** Editions for clouds that have Business Rules Engine enabled |

| USER PERMISSIONS NEEDED | |
|---|---|
| To create a flow: | Manage Flow |
| To invoke an expression set: | Rules Engine Runtime |

1. From Setup, in the Quick Find box, enter `Flows`, and then select **Flows**.
2. Click **New Flow**.
3. Select **Start From Scratch**, and then click **Next**.
4. Select **Screen Flow**, and then click **Create**.

## Create a User Input Screen

1. Click .

2. Select the **Screen** interaction element.

3. In the Screen Properties section, enter `Enter Details` as the label.
4. Drag the Text component from the Components list to the Header component.

5. Add the Text details.
   - Label: Member Number
   - API Name: MemberNumberInput
6. Select **Require**.

7. Drag the Date component from the Components list to the Member Number text component.
8. Under Date, enter `Date of Service` as the label.

9. Select **Require**.
10. Drag the Text component to the Date of Service date component.
11. Under Text, enter `Member Plan Type` as the label.
12. Similarly, add these service code details in sequence.

| Component | Label |
|---|---|
| Text | Service Code 1 |
| Text | Service Code 2 |
| Text | Service Code 3 |
| Text | Service Code 4 |

13. Click **Done**.
14. Save your changes.
15. Enter `Health Care Prior Authorization Check` as the flow label.
16. Save your changes.

## Assign Variables to Service Codes

1. Click after the Enter Details element.
2. Select the **Assignment** element.
3. In Label and API Name, enter `ServiceCodeList`.
4. Enter the variable values.
   a. Click in the Variable field.
   b. Select **New Resource** from the dropdown.

   c. Select **Variable** as the resource type.
   d. Enter `c_ServiceCode` as the API name.
   e. Select **Text** as the data type.
   f. Select **Allow multiple values (collection)**.
   g. Click **Done**.
   h. Select **Add** as the operator.
   i. Click in the Value field, and select the **Service Code 1** screen component.
   j. Click **Add Assignment**.
   k. Similarly, add these variable values.

| Variable | Operator | Value |
|---|---|---|
| c_ServiceCode | Add | Service Code 2 |
| c_ServiceCode | Add | Service Code 3 |
| c_ServiceCode | Add | Service Code 4 |

## Add Context IDs

1. Click after the ServiceCodeList element.
2. Select the **Get Records** element.
3. Enter `Get Context Definition Id` as the label.
4. Select **Context Definition** as the object.
5. Under Filter Context Definition Records, add these details.
   a. Select **All Conditions Are Met (AND)** as the condition requirement.
   b. Select **DeveloperName** as the field.
   c. Select **Equals** as the operator.
   d. Enter `Get_MemberPlan_Details` as the value.
6. Save your changes.
7. Click after the Get Context Definition Id element.
8. Select the **Get Records** element.
9. Enter `Get Context Version Id` as the label.
10. Select **Context Definition Version** as the object.
11. Under Filter Context Definition Version Records, add these details.
    a. Select **All Conditions Are Met (AND)** as the condition requirement.
    b. Select **ContextDefinitionId** as the field.
    c. Select **Equals** as the operator.
    d. Enter `Context Definition from Get_Context_Definition_Id > Context Definition Id` as the value.
12. Save your changes.
13. Click after the Get Context Version Id element.
14. Select the **Get Records** element.
15. Enter `Get Mapping Id` as the label.
16. Select **Context Mapping** as the object.
17. Under Filter Context Mapping Records, add these details.
    a. Select **All Conditions Are Met (AND)** as the condition requirement.
    b. Select **ContextDefinitionVersionId** as the field.
    c. Select **Equals** as the operator.
    d. Enter `Context Definition Version from Get_Context_Version_Id > Id (Context Definition Version Id)` as the value.
18. Save your changes.

## Add Loop

1. Click after the Get Mapping Id element.
2. Select the **Loop** element.
3. Enter `Loop through Service Codes` as the label.
4. Select **c_ServiceCode** as the collection variable.
5. Select **First item to last item** as the direction.

## Hydrate Member Plan Object and Child Nodes Data

1. Click after For Each.
2. Select the **Action** element.
3. Find and select the **Build Context** action.
4. Enter `Hydrate data from Member Plan Object and its child nodes` as the label.
5. Under Set Input Values for the Selected Action, add these details.
   a. Select **Context Definition from Get_Context_Definition_Id > Context Definition Id** as the context definition.
   b. Turn on Context Data and enter `{"MemberPlan":[{"id":"<record id>","businessObjectType":"MemberPlan"}]}`. To get the record ID, open the Cigna Medicare Advantage Plan member plan record that you created earlier and copy the record ID from the URL. Paste the record ID in the *<record id>* placeholder. For example, 0SqdN0000000ExRSAU is the record ID shown in the screenshot.

   c. Turn on Context Mapping and select **Context Mapping from Get Mapping Id > Context Mapping Id** as the context mapping.
6. In the Show advanced options section, select **Let the flow decide (Recommended)** as the transaction control.
7. Save your changes.

## Assign Service Code

1. Click after the Hydrate data from Member Plan Object and its child nodes element.
2. Select the **Assignment** element.
3. In Label and API Name, enter `Assign Service Code`.
4. Under Set Variable Values, add these details.
   a. Click in the Variable field.
   b. Select **New Resource** from the dropdown.
   c. Select **Variable** as the resource type.
   d. Enter `ServCode` as the API name.
   e. Select **Text** as the data type.
   f. Click **Done**.
   g. Select **Equals** as the operator.
   h. Click in the Value field, and select **Current Item from Loop Loop_through_Service_Codes**.
5. Save your changes.

## Fetch Expression Set Details

1. Click after the Assign Service Code element.
2. Select the **Action** element.
3. Find and select the **Member Plan Status Check-Bre** action.
4. Enter `Member Plan Details` as the label.

5. Under Set Input Values for the Selected Action, add these details.
   a. Turn on DateOfService and select **Enter Details > Date of Service** as the date of service.
   b. Turn on MemberNumberInput and select **Enter Details > Member Number**.
   c. Turn on Get_MemberPlan_DetailsId select **Outputs from Hydrate data from Member Plan Object and its child nodes > Context Id**.
   d. Turn on ServiceCode and select **ServCode**.
6. Save your changes.

## Assign Output Values

1. Click after the Member Plan Details element.
2. Select the **Assignment** element.
3. Enter `Output Values from Expression Set` as the label.
4. Enter `OutputValues` as the API name.
5. Under Set Variable Values, add these details.
   a. Click in the Variable field.
   b. Select **New Resource** from the dropdown.
   c. Select **Variable** as the resource type.
   d. Enter `c_PriorAuthDetermination` as the API name.
   e. Select **Text** as the data type.
   f. Select **Allow multiple values (collection)**.
   g. Click **Done**.
   h. Select **Add** as the operator.
   i. Click in the Value field, and select **Outputs from Member_Plan_Details > PriorAuthDetermination**.
6. Save your changes.

## Show Results

1. Click after the Output Values from Expression Set element.
2. Select the **Screen** element.
3. Enter `Result Screen` as the label.
4. Drag a Display Text component to the Header component.
5. Enter `Results` as the API name.
6. In the text input field, enter `Prior Authorization Determination –` and click in the Insert a resource field.
7. Find and select **Outputs from Member_Plan_Details > PriorAuthDetermination**.
8. Click **Done**.
9. Save your changes.
10. Click **Activate**.

## Run Health Care Prior Authorization Check Scenario

Run the Health Care Prior Authorization Check flow to execute the Prior Authorization scenario and

experience how Business Rules Engine performs calculations and provides the results.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise** and **Unlimited** Editions for clouds that have Business Rules Engine enabled |

1. From Setup, in the Quick Find box, enter `Flows`, and then select `Flows`.
2. Click **Health Care Prior Authorization Check**.
3. Click **Run**.
4. Enter these healthcare details.
   a. Enter `675858989` as the member number.
   b. Select **[Current Date]** as the date of service.
   c. Enter `Active` as the member plan type.
   d. Enter these service codes.

| Service Code Number | Service Code |
|---|---|
| Service Code 1 | CPT23456 |
| Service Code 2 | CPT23457 |
| Service Code 3 | CPT23458 |
| Service Code 4 | CPT23459 |

5. Click **Next**.
6. For CPT23456, the health care administrative request is approved. Click **Next**.
7. Similarly for CPT23457 and CPT23458 the health care administrative request is approved, and for CPT23459, the health care administrative approval isn't required.

# Business Rules Engine Default Limits

Review the default limits for Business Rules Engine components and their usage. To raise any default limits, contact Salesforce Customer Support.

REQUIRED EDITIONS

| |
|---|
| Available in: Lightning Experience |
| Available in: **Enterprise**, **Unlimited**, and **Developer** Editions for clouds that have Business Rules Engine enabled |

To ensure optimal performance, consider these default limits when you use Business Rules Engine.

**Note** The values of default limits are copied from your Salesforce production org to your sandbox.

## Expression Sets

| Value | Default Limit | Extensible |
|---|---|---|
| The number of local resources that can be used in a version | 1000 | Yes |
| The number of object aliases whose fields can be referenced in a version | 5 | No |
| The number of object field aliases that can be used in a version | 100 | No |
| The number of context tags that can be used as list variables in a version | 100 | Yes |
| The number of steps that a version can have | 200 | Yes |
| The number of business elements that can be used in a version | 50 | Yes |
| The number of conditions that can be used in a conditional group step in a version | 200 | No |
| The maximum expression set to subexpression depth | 2 levels<br><br>For example, an expression set A that doesn't reference any subexpressions is considered to have a depth of 0. If A references B, whose depth is 0, A has a depth of 1. If B references another subexpression C, then B has a depth of 1, and A has a depth of 2. However, if C further references D, then the depth of A becomes 3 levels, which isn’t supported. | No |

| Value | Default Limit | Extensible |
| --- | --- | --- |
| The number of expression sets executed per hour | 5,000,000 | Yes |

## Decision Matrices

| Value | Default Limit | Extensible |
| --- | --- | --- |
| The maximum size of a CSV file that can be uploaded to a decision matrix version | 100 MB | No |
| The number of input sets that can be simultaneously looked up in a decision matrix | 200 | No |
| The maximum number of decision matrices invoked per hour | 400,000 | No |

## Standard Decision Tables

| Value | Default Limit | Extensible |
| --- | --- | --- |
| The maximum number of standard decision tables invoked per hour | 2,000,000 | Yes |
| The number of rows that a standard decision table can process | 100,000 | Yes |
| The number of output results a standard decision table can have | 10 | No |
| The number of standard decision tables that can simultaneously be active in an org | 20 | Yes |
| The number of input fields that can be used to group standard decision table rows | 2 | No |
| The number of groups that a decision table's rows can be split | 200 | Yes |

| Value | Default Limit | Extensible |
|---|---|---|
| into | | |
| The number of rows that each group in a decision table whose rows are grouped by input fields can have | 25,000 | Yes |
| The number of output rows that a decision table can return | 50 (When a single set of inputs is looked up)<br><br>200 (When multiple sets of inputs are simultaneously looked up)<br><br>This limit applies irrespective of the number of matching output rows. | No |
| The number of input sets that can be simultaneously looked up in a standard decision table | 200 | Yes |
| Number of full refreshes that can be made per hour for all standard decision tables in an org | 40 | No |
| **For each combination of Equals, Required, and All conditions are met (AND) columns** | | |
| Number of columns in a standard decision table | 15 | Yes |
| The number of input conditions a standard decision table can have | 30<br><br>The input field limit of 30 is subject to how the input fields in the decision table are configured. The fields must be configured such that a maximum of 10 input fields are considered. To reduce the number of input fields to 10, use the AND operator in the table's input condition logic to join up to 21 required columns that use | Yes |

| Value | Default Limit | Extensible |
|---|---|---|
| | the Equals operator. | |

## Advanced Decision Tables

| Value | Default Limit | Extensible |
|---|---|---|
| The maximum number of advanced decision tables invoked per hour | 10,800,000 | Yes |
| The number of rows that an advanced decision table can process | 20,000,000 | Yes |
| The number of advanced decision tables that can simultaneously be active in an org | 100 | Yes |
| The number of input sets that can be simultaneously looked up in an advanced decision table | 200 | Yes |
| The maximum number of rows in an advanced decision table for each combination of Equals, Required, and All conditions are met (AND) columns. | 200 | No |
| Number of incremental refreshes in an org per hour | 60 | Yes |
| Number of rows updated in an incremental refresh of an advanced decision table | 100,000 | Yes |
| Number of output columns in an advanced decision table | 10 | No |
| Number of input sets that can be simultaneously looked up in an advanced decision table | 200 | Yes |
| Number of output rows that can be returned when multiple sets of inputs are simultaneously | 40,000 | Yes |

| Value | Default Limit | Extensible |
|---|---|---|
| looked up in an advanced decision table | | |
| Number of full refreshes that can be made per hour for all advanced decision tables in an org | 60 | Yes |
| Number of objects in a decision table | 5 | No |
| **For each combination of Equals, Required, and All conditions are met (AND) columns** | | |
| The maximum number of matched columns in an advanced decision table | 31 | No |
| The maximum number of columns in an advanced decision table that are not matched | 10 | Yes |
| **CSV-Based Decision Tables**<br><br>All default limits for advanced decision tables are also applicable to CSV-based decision tables. | | |
| The number of rows that an advanced CSV-based decision table can process | 1,000,000 | No |
| The number of rows that you can upload in a CSV-based decision table | 50,000 | Yes |
| Number of CSV files being uploaded simultaneously | 5 | Yes |
| Number of CSV files being downloaded simultaneously | 5 | Yes |