A

Major Project Report

on

***Interview Management System***

Submitted in partial fulfillment for the award of degree of

**Bachelor of Technology**

In

**Computer Science & Engineering**



|  |  |
| --- | --- |
| **Submitted to:**  **Mr. Pradeep Jha**  **HOD-CSE/IT/AI&DS Dept.** | **Submitted By:**  **Arti Acharya**  **19EGJCS048**  **Shrashti Chechani**  **19EGJCS815** |

**Department of Computer Science & Engineering**

Global Institute of Technology

Jaipur (Rajasthan)-302022

Batch: 2019-23

**GLOBAL INSTITUTE OF TECHNOLOGY**

ITS-1 & 2, IT Park, EPIP, Sitapura, Jaipur – 302022, Rajasthan – INDIA

***Certificate***

This is to certify that the work, which is being presented in the project entitled “Interview management system**”** submitted by Ms. Arti Acharya and Ms. Shrashti Chechani, a student of fourth year (VIII Sem), B.Tech. in Computer Science & Engineering, in partial fulfilment for the award of degree of Bachelor of Technology is a record of student’s work carried out and found satisfactory for submission.

Arti Acharya

19EGJCS048

Shrashti Chechani

19EGJCS815

|  |  |
| --- | --- |
| Guide  (Mr. Pankaj Jain) | Project Coordinators  Mr. Amit Bohra  Ms. Manju Mathur |
|  |  |
|  |  |
|  |  |
| Mr. Pradeep Jha  (HOD, CSE/ IT/ AI & DS Dept.) |  |

**Acknowledgement**

We take this opportunity to express my deep sense of gratitude to project coordinators **Ms.** **Manju Mathur**, Assistant Professor, Department of Computer Science and Engineering, **Mr.** **Amit Bohra,** Assistant Professor, Department of Computer Science and Engineering and our project guide **“Mr. Pankaj Jain”,** Assistant Professor Department of Computer Science and Engineering, Global Institute of Technology, Jaipur, for his valuable guidance and cooperation throughout the Project work. He provided constant encouragement and unceasing enthusiasm at every stage of the project work.

We are grateful to our respected **Dr. I. C. Sharma, Principal GIT,** for guiding us. We express our indebtedness to **Mr. Pradeep Jha**, Head of the Department of Computer Science and Engineering, Global Institute of Technology, Jaipur, for providing us with ample support.

Without their support and timely guidance, the completion of our project would have seemed a farfetched dream. We find ourselves lucky to have mentors of such great potential in this respect.

**Place:** GIT, Jaipur

**Arti Acharya**

19EGJCS048

**Shrashti Chechani**

19EGJCS815

B.Tech. VIII Semester, IV Year

Computer Science & Engineering

**Abstract**

Interview Management System in APPIAN is a project for digitally managing the interviews of any individual. The project contains the Interview resource allocation and HR Department.

The Interview Management System (IMS) will serve as automated management and a better way of selection system will be there. IMS will bring, scheduling convenience and reminders to meet the criteria of every fresher and experienced one.

In IMS automated interview, interviewing will be there where the recruiter assesses the candidate without conducting one in real-time. In real-time interviews, the Resources or candidates are required to travel or spend a significant amount of time in setting up and completing the process but by the IMS the automation Process will reduce half of the time and in this section, and resources team will be able to take multiple interviews.

Smart organizations are discerning the growing need for interview scheduling software companies.

Intelligent online automated systems are a way of simplifying the interaction between candidates, the Resources team, and the HR Department with the help of IMS.

With IMS, the entire Resources management process is handled online and offline.

Catalog

[Chapter-1 3](#_Toc5610)

[INTRODUCTION 3](#_Toc3785)

[1.1.1. Project Description : 4](#_Toc315)

[1.1.2. Purpose : 4](#_Toc31848)

[1.1.3. Assumptions : 4](#_Toc5702)

[1.1.4. Constraints 5](#_Toc23983)

[Technical constraints: 5](#_Toc25305)

[1.2.1. Users: 5](#_Toc12766)

[1.2.2. Functional Requirements : 5](#_Toc5943)

[Chapter-2 6](#_Toc26149)

[Appian Data Types 6](#_Toc23008)

[Overview 6](#_Toc1227)

[Any Type 6](#_Toc15010)

[Primitive system data types 7](#_Toc23174)

[Boolean 7](#_Toc18191)

[Date 7](#_Toc11145)

[Date and Time 7](#_Toc5069)

[Number (Decimal) 8](#_Toc7703)

[Number (Integer) 8](#_Toc26984)

[Text 8](#_Toc28315)

[Time 8](#_Toc30845)

[Complex System Data Types 9](#_Toc24983)

[Application Test Result 9](#_Toc22652)

[DataSubset 9](#_Toc18915)

[EntityData 9](#_Toc27995)

[Chapter-3 10](#_Toc28183)

[Data Store Object 10](#_Toc29672)

[Overview: 10](#_Toc11797)

[Create : 11](#_Toc12700)

[Edit: 11](#_Toc4620)

[Delete : 12](#_Toc27680)

[Security: 12](#_Toc30316)

[Retrieving External Data 12](#_Toc11745)

[Perform Bulk Operations as Much as Possible: 12](#_Toc25763)

[Retrieve Data in Smaller Batches: 12](#_Toc18539)

[Chapter-4 14](#_Toc26268)

[Decisions 14](#_Toc19036)

[Overview: 14](#_Toc21332)

[Create : 14](#_Toc3631)

[Decision basics: 14](#_Toc17671)

[Inputs: 15](#_Toc28725)

[Outputs: 15](#_Toc11706)

[Compound output: 16](#_Toc28015)

[Define decision logic 16](#_Toc25907)

[Design guidance 16](#_Toc14892)

[Versions 17](#_Toc9122)

[Move 17](#_Toc5778)

[Duplicate 17](#_Toc13948)

[Security 18](#_Toc6271)

[Chapter-5 19](#_Toc26978)

[Expression Rules 19](#_Toc7526)

[Overview: 19](#_Toc11014)

[Reserved names: 19](#_Toc27662)

[Create 20](#_Toc3428)

[Add rule inputs: 20](#_Toc2806)

[Editing rule inputs: 20](#_Toc23741)

[Define the rule: 21](#_Toc4642)

[Test the rule: 21](#_Toc4918)

[Ad hoc test: 21](#_Toc2018)

[Test cases 22](#_Toc28442)

[Create a test case 22](#_Toc8940)

[Save ad hoc test values as a test case 22](#_Toc24438)

[Assertion examples and recommendations 22](#_Toc9349)

[Assertion recommendations: 23](#_Toc7488)

[Expression rules should have enough test cases to accurately describe the behavior of the rule. This increases confidence that the rule works properly for runtime scenarios and helps with maintenance of the rule as it changes over time. 23](#_Toc29576)

[Run test cases 23](#_Toc9913)

[Edit: 24](#_Toc17592)

[1. In your application, go to the Build view. 24](#_Toc30459)

[Design guidance 24](#_Toc25250)

[Move 25](#_Toc25292)

[Duplicate 25](#_Toc31134)

[Delete 25](#_Toc3372)

[Security 26](#_Toc12453)

[Chapter-6 27](#_Toc14653)

[Interface Object 27](#_Toc24094)

[Overview 27](#_Toc11375)

[Create from data type 31](#_Toc24528)

[Duplicate an interface 32](#_Toc11037)

[Editing rule inputs 34](#_Toc6038)

[Edit an interface 36](#_Toc11004)

[Edit an interface 37](#_Toc23180)

[Chapter-7 41](#_Toc29189)

[Process Model Object 41](#_Toc20173)

[Overview: 41](#_Toc19915)

[Creating a process model: 42](#_Toc21501)

[Process analyst view 42](#_Toc11863)

[Process start form tab 43](#_Toc13610)

[Canvas annotations 44](#_Toc2300)

[Process model recommendation dismissal 45](#_Toc5847)

[Generating process model documentation 45](#_Toc19142)

[Open a process model. 45](#_Toc27172)

[Process model diagram 46](#_Toc8053)

[Node list 47](#_Toc13939)

[Event Nodes 48](#_Toc14617)

# Chapter-1

## INTRODUCTION

•The Functional Requirements Document (FRD) is a formal statement of the application’s functional and operational requirements. It serves as a contract between the developer and the customer for whom the system is being developed.

•The developers agree to deliver a system that meets all functional requirements and provides the capabilities specified in the FRD.

•The client agrees to find the product satisfactory if it satisfies the requirements and provides the capabilities.

•The information in the FRD is solution-independent. It is a statement of what the application is to do, not of how it works. There is no commitment to a particular design.

This FRD does the following:

• Demonstrates, in terms of the business objectives and business processes, that the application

provides value to the customer.

• Contains a complete set of requirements for the application.

The FRD does not do the following:

• Describe how the application provides the required capabilities.

• Describe design considerations such as computer hardware, operating system, and database design.

### 1.1.1. Project Description :

•Interview Management System in APPIAN is a project for digitally managing the interviews of any individual. The project contains the Interview resource allocation and HR Department.

•The Interview Management System (IMS) will serve as automated management and a better way of selection system will be there. IMS will bring, scheduling convenience and reminders to meet the criteria of every fresher and experienced one.

•In IMS automated interview, interviewing will be there where the recruiter assesses the candidate without conducting one in real-time. In real-time interviews, the Resources or candidates are required to travel or spend a significant amount of time in setting up and completing the process but by the IMS the automation Process will reduce half of the time and in this section, and resources team will be able to take multiple interviews.

•Smart organizations are discerning the growing need for interview scheduling software companies.

Intelligent online automated systems are a way of simplifying the interaction between candidates, the Resources team, and the HR Department with the help of IMS.

With IMS, the entire Resources management process is handled online and offline.

### 1.1.2. Purpose :

•This FRD outlines how IMS will meet the goals in the process of Automation in which the resource person and candidate availability will be there.

•The whole process of IMS will be automated for the Candidates, HR team, and resources person.

•The time complexity of time will be reduced after the process of automation in IMS.

### 1.1.3. Assumptions :

•Assumptions, as used in this document, are characteristics of the project environment, outside the control of the developer, whose outcomes influence the success of a project.

•The developer should take these assumptions into account when designing the system.

=Assumptions associated with the development of IMS are listed below:

• All users are located in a Different geographical location.

• Users will need to access MMS both via mobile clients as well as web browsers.

### 1.1.4. Constraints

•Constraints, conditions outside the control of the project that limit the design alternatives, can be broadly categorized as technical and functional.

=The following constraints are applicable.

## Technical constraints:

• Compatibility with the latest version of Appian.

• Compatibility with MySQL.

• Usability on screen with 1024x1280 resolution.

### 1.2.1. Users:

•Candidate

•Resources

•HR

### 1.2.2. Functional Requirements :

The requirements below define the minimum required functionality of the system. In situations where specific details are missing or incomplete, use judgment and common sense to complete the functionality.

# 

# Chapter-2

## Appian Data Types

### Overview

The data (process variables, node inputs, node outputs, rule inputs, data store entities, or constants) used by Appian must conform to certain data types.

Appian data types can be one of the system types or a custom data type built from an XML Schema Definition (XSD), Java object, or imported from a WSDL by the Call Web Service smart service.

### Any Type

The Any Type is a generic data type only available for use as an expression input for rules and certain expression functions, such as the if() function. It accepts data of any data type.

Data is stored in variables of this type by mapping an existing variable, rule, constant, or expression to its value.

### Primitive system data types

The following primitive system data types are available.

### Boolean

Values include True / False.

1 is accepted as a literal value meaning Yes.

0 is accepted as a literal value meaning No.

The default value is null, which appears as [Empty Value] in a process variable. The output of an empty boolean value in an expression depends on the function used.

You can populate the value using results from the true() or false() functions.

The to boolean() function can be used to convert true/false and 0/1 values into a Boolean data type value.

### Date

Data in a date format can be created using the [date(year, month, date) function](https://docs.appian.com/suite/help/23.1/fnc_date_and_time_date.html). Variables that have a date data type do not accept text string input.

The default value for a date is [Empty Value]. The minimum value is 1/1/1000, and maximum value is 12/31/9999. Dates values are not adjusted to a different time zone when saved or displayed.

Numerical and text values can be converted to a date value using the todate() function.

### Date and Time

Variables that hold a Date and Time data type refer to a point in time that is the same for all users. Date and time variables do not accept text strings as input.

A Date and Time value is saved in Greenwich Mean Time (GMT) then converted to the end-user's time zone (accounting for daylight saving time) when displayed.

GMT is the default time zone used when evaluating expressions that include Date and Time values. You can specify a different time zone in your process model properties. The time zone used for display can be the user's preferred time zone, a globally specified time zone, or the time zone context.

If a function expects a Date and Time value, and you pass it a date, the date is automatically converted to a Date and Time value, using 12:00 am for the time component.

The Date and Time value is only converted to the end user's time zone when it is displayed in the following manner. (Use separate date values and time values if you do not want this conversion to take place.)

### Number (Decimal)

Holds numeric data stored as double precision floating-point decimal numbers. The default value is 0.0.

Decimal numbers can be created from text strings using the todecimal() function.

For forms, use one of the following Text Functions to display a decimal number as a default value to avoid rounding.

fixed()

text()

Also for forms, use the a! currency() function to localize a currency value based on a given ISO currency code.

### Number (Integer)

Integer numbers can range from -2,147,483,647 to 2,147,483,647 (or from -231+1 to 231-1 in scientific notation).

The default value is 0 and the null value is -2\^31.

Integer numbers can be created from text strings using the tointeger() function.

When an arithmetic operation (such as an expression) creates a Number (Integer) value that exceeds the type's limits, the value wraps.

---> 2147483647 + 10 = -2,147,483,639

---> 2147483647 + 1 = -2147483648 - interpreted as null. When looking in the user interface at a process variable changed to this value, an [Empty Value] result is displayed.

When values that exceed the Number(Integer) range are passed through a user interface, the excessive value is changed.

In most cases, this is changed to a null value.

When a value that exceeds the Number(Integer) range is converted to Number(Integer) from a string in an engine server (such as when you convert text to an integer using the to integer() function), the excessive value is changed to the maximum value.

On task forms, the Number Form Component prevents entry of values outside of the valid range.

If an expression is entered for a value and the Number Form Component is not mapped to a node input, any values that exceed 7 digits display in standard form

### Text

This type is used to store any UTF-8 text string. Numerical values can be entered into the text data type; however, data manipulation cannot be performed on the text data type (except for report aggregations). The default value is [Empty Value].

To display text, enclose it within double quotation marks ("").

### Time

Time data can be created using the time(hour,minute,second) function. Variables that have a time data type do not accept text string input.

Time values are not adjusted to a different time zone when saved or displayed.

## Complex System Data Types

The following complex system types are made available in the system to support smart services.

These types cannot be edited or deleted. Their XML structure is not guaranteed to remain the same from release to release.

### Application Test Result

The Application Test Result data type is designed to hold test result information for all expression rules in an application. This type also includes execution statistics for an individual application.

### DataSubset

The DataSubset data type is designed to hold the data returned by a query configured with a paging parameter.

It contains the following fields:

* **startIndex** - This field holds a single Number(Integer) record.
* **batchSize** - This field holds a single Number(Integer) record.
* **sort**- This field holds multiple SortInfo records.
* **totalCount**- This field holds a single Number(Integer) record.
* **data** - This field holds multiple Any Type records.
* **identifiers** - This field holds multiple Any Type records.

### EntityData

The EntityData data type lets you define a target data store entity and the values to store in the target entity as an input value for the Write to Multiple Data Store Entities Smart Service.

# 

# Chapter-3

## Data Store Object

### Overview:

Data stores connect Appian to external relational databases in order to store and retrieve application data. This page covers how to create data stores, run DDL scripts, and retrieve external data in your application.

Data stores allow you to insert, update, query, and delete data in the format needed by your applications without writing structured queries (SQL). You must have a business data source configured to use this feature.

Custom data types are used by your processes to store data in an external relational database. The relationships between these custom data types are used to create tables and keys in the database that are associated with through the data store.

Once you define a custom data type and map it to a data store entity, you can update data in the database using the Write to Data Store Entity Smart Service. You can also use a!queryEntity() to retrieve information through the data store, without knowing the underlying table schemas of the database or SQL.

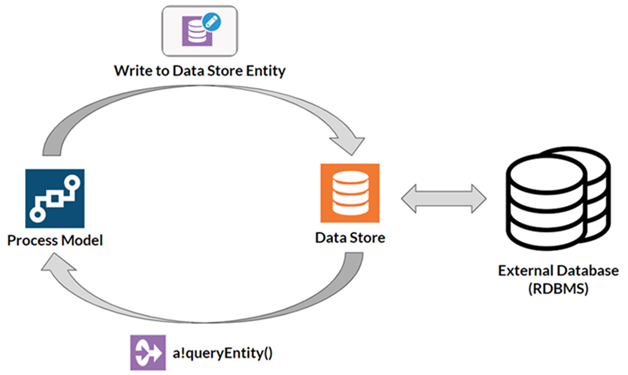


Fig 3.1

### Create :

Once you've created the data store, you need to map your custom data types to the external database by adding entities to the data store.

To add an entity to the data store:

1 Click Add Entity. An entity dialog displays.

2 In the first box of the dialog, enter a name for the new entity.

3 In the second box of the dialog, browse to and select a custom data type from the list of available CDTs.

4 Click Save.

### Edit:

All data store attributes can be edited by users with Editor or Administrator permissions. The following list details how to edit each attribute:

1. Click on the name or description to edit them in place.
2. Update the data source by selecting one from the list of available data sources.
3. Check the Automatically update database schema checkbox to allow Appian to execute DDL SQL statements when dependent data types are updated or when this data store is imported. Changes to this attribute do not take effect in the environment if the auto-update feature has been disabled from the Admin Console.
4. Click an entity to modify its name or its data type.
5. Click Add Entity to configure a new data store entity.

### Delete :

Go to an application that contains the data store.

1. Select it in the grid and then click the Delete button in the grid toolbar.
2. System administrators have the ability to delete data stores (and other objects) in bulk by selecting them and clicking Delete in the toolbar.

### Security:

The security role map of a data store controls which users can see or modify it and its properties. By default, only the data store creator and system administrators have access to the data store. See Editing Object Security to modify a data store's security.

1. The following table outlines the actions that can be completed for each permission level in a data store's security role map.

2.Users must have at least Viewer permissions to the data store in order to read or write data using the Write to Data Store smart service.

## Retrieving External Data

Use the following guidelines when loading data into a data store from an external RDBMS.

### Perform Bulk Operations as Much as Possible:

Minimize the number of individual queries to the database.

You can write arrays of data to your data store using the Write to Data Store Entity Smart Service, so there is no need to run multiple instances to write data to a single entity. To write multiple data types, you can use the Write to Multiple Data Store Entities Smart Service.

### Retrieve Data in Smaller Batches:

The amount of data that can be returned by a query is limited to 1MB by default. If you expect to return more than 1MB of data, you can design your process models to retrieve data in smaller batches and loop the process flow to retrieve the desired data.

When using a query to start processes, you may quickly reach the default limit of 1000 activated instances by repeatedly activating your Send Message Event and gateways (which do not provide the option to delete previously completed instances in the same manner as process flow activities).

You may want to configure your Send Message Event in a subprocess and configure the node properties to delete the completed instances when they are no longer needed.

You can also configure the other process activities to automatically delete previously completed instances to avoid reaching the 1000 node limit. This setting also reduces memory (RAM and HD) usage of your processes.

# 

# Chapter-4

## Decisions

### Overview:

A decision is a grouping of business rules that determines output values based on a set of inputs.

Unlike expression rules, which primarily calculate or manipulate data, decisions are best used to encapsulate complex, business-specific logic. Decisions are configured in the decision designer, which enables fast, intuitive implementation of business logic without using expressions by way of decision tables.

Decisions can be called from any expression, so they can be reused across multiple objects throughout the system.

Each time you modify and save a decision, a new version is created. All expressions that use the decision use the latest version. All versions are accessible to designers who can view the decision, and a decision can be reverted back to a previous version at any time.

For step-by-step guidance on how to build your first decision, see the Create a Decision page.

For examples of different decision use cases, see the Appian Decisions page. These examples will help you understand how you can leverage decision objects in your applications.

### Create :

1. In your application, go to the Build view.
2. Click NEW > Decision.
3. In the Create Decision dialog, configure.
4. Click CREATE.

The decision object opens in a new tab or window.

### Decision basics:

The logic of a decision is configured using a decision table.

In a decision table, each column is either an input to the decision or one of its outputs. Each row in the table represents a decision rule, in which each cell is a logical operation. When all the cells' conditions for a particular row are true, the decision will return the output values specified for that row.

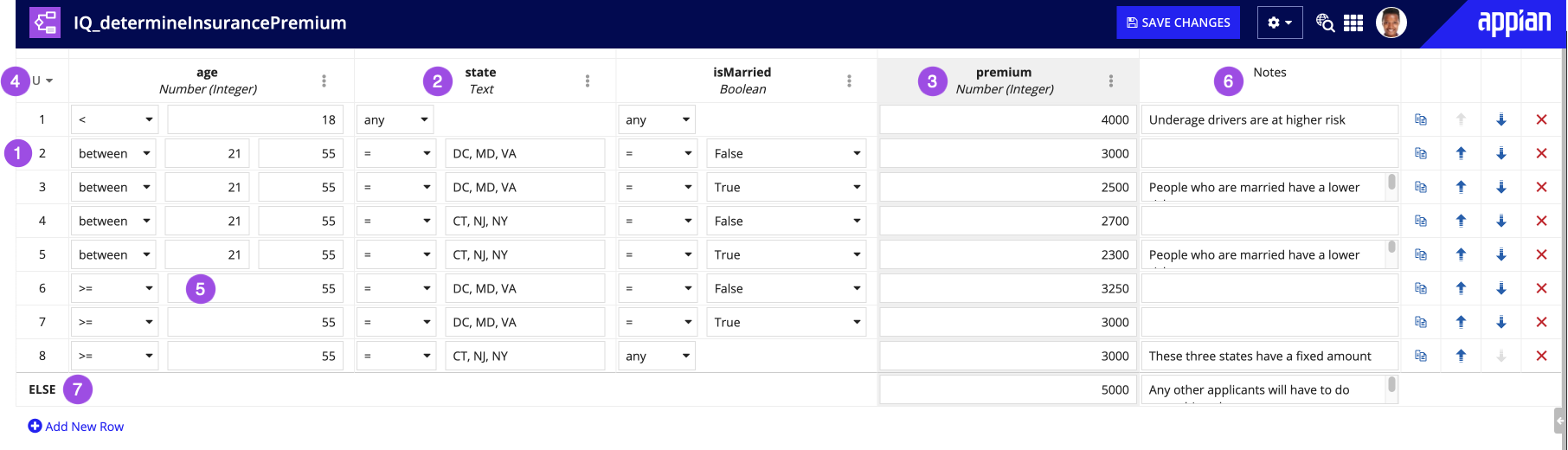


Fig:4.1

### Inputs:

The input description can be seen by hovering over the question mark icon by the input name. If there is no question mark icon, it means no description was added to that input.

The following types are available for decision inputs:

* Boolean
* Date
* Number (Decimal)
* Number (Integer)
* Text

A decision is invalid if not all of its input values are defined. When a decision with undefined values is evaluated, it will fail to return a value. The decision designer helps you identify if there are any missing or invalid values when you test and save your decision.

### Outputs:

The decision output is used to specify what the decision returns if a particular row's logic matches the passed in input values. Decision output columns are indicated by a gray background.

The following types are available for decision outputs:

* Boolean
* Date
* Document
* Folder
* Group
* Number (Decimal)
* Number (Integer)
* Process Model
* Text
* User
* User or Group

### Compound output:

Like inputs, a decision can have multiple outputs. This is considered a compound output. When a decision has a compound output, it will return a dictionary with fields that correspond to each output column of the decision. The field names in the dictionary are the output names specified in the decision.

Decisions have an explicit return type, this helps designers understand what to expect when integrating the decision into their applications. The return type is displayed in the inline help within expression editors along side the decision's description and inputs. A decision's return type depends on the type of output, whether it has a compound output, and its hit policy.

When a decision is evaluated but there are no matching rows and no default output is defined, the decision will return either a null of its output type or an empty list of its output type if it is an array type (which is determined by the scenarios outlined in the table above).

### Define decision logic

Decisions are defined in the decision designer by specifying logic in the decision table located in the main pane of the designer.

Each input cell in the decision table corresponds to a logical operation that is evaluated with the value passed into the input at execution time. When called, the decision evaluates each row in order. If any cell in the row evaluates to false then the decision stops evaluating that row.

If all the logical conditions for a row evaluate to true, then that row is said to be a hit, or a match, and the decision will result in the defined output value for that row. The decision's hit policy determines whether it will continue to evaluate rows or finish once it has found a match.

### Design guidance

While editing your decisions you may encounter guidance. Appian design guidance reinforces best practice design patterns that should be implemented in your objects. Guidance is calculated while editing expressions within the decision or when a precedent of the decision is updated.

* In addition to supporting general expression guidance, the decision designer helps you create smarter decisions with the following decision specific functionality:
* The designer immediately validates the soundness of your logic by identifying when values are invalid.
* Upon testing and saving the decision, decision designer pinpoints which cells are missing values.
* Overlapping rule validations during testing the decision identify when rules won't ever match.
* Completeness check discovers gaps in your logic as you test your decision
* Overlapping rules validation
* When using the Unique hit policy, the designer will highlight which of your rows overlap with each other to identify when rules won't ever be matched.
* When a Unique hit policy Decision has overlapping rules, it will fail to evaluate. Overlapping rules are highlighted red on the decision table and when you hover over the error icon for that row, a tooltip will appear identifying which rows it overlaps with.

### Versions

Each time you modify and save a decision, a new version is created. All objects that use the decision will use the latest version. All versions are accessible to designers who can view the decision, and a decision can be reverted back to a previous version at any time.

For information on how to manage object versions, see Managing Object Versions.

### Move

Users with Administrator permission to this object or rule folder can move it to another folder:

1. Navigate to the parent folder that the object or folder is in.
2. Select the object or folder. You can select more than one.
3. Click the Move button on the toolbar. The Move Objects dialog appears.
4. Use the folder picker or browser to select the new parent folder.
5. Click Move.

### Duplicate

Developers with Viewer permissions to this object can duplicate it. There are two ways to duplicate an object:

* From any view in an application, you can select the object you wish to duplicate and use the toolbar option to launch the duplication dialog. The duplicated object will be added to the application you're working in. This capability is only available for single object selections from the grid.
* If you are in an interface, expression rule, integration or decision object, you can select Duplicate from the object's settings menu . From there, you can specify the target application for the new object.

### Security

Any user can invoke any decision that is used by an interface or process they can access.

The security role map of a decision controls which developers can see or modify it and its properties. By default, decisions inherit the security of the folder that they are saved in. However, after creating the decision, you can disable that inheritance and modify the decision's security. See Editing Object Security to modify a decision's security.

# 

# Chapter-5

## Expression Rules

## Overview:

An expression rule is a stored expression that can be called from other expressions. Within your expression rules, you can add rule inputs so you can dynamically change the data returned by your expression.

This page explains how to create, edit, and delete expression rules.

For more information on expression rules, see:

* Testing an Expression Rule
* User Contexts for Expressions

### Reserved names:

The following names are reserved for process report metrics, and must not be used.

* Completion
* is\_process\_ontime
* is\_process\_overdue
* is\_task\_favorite
* is\_task\_ontime
* is\_task\_overdue
* Lag
* NetCompletion
* NetLag
* NetWork
* process\_completion\_time
* process\_deadline
* process\_ee\_id
* process\_start\_time
* process\_status
* pm\_avg\_completion
* pm\_avg\_lag
* pm\_avg\_work
* task\_assignee\_owner
* task\_assignment\_time
* task\_attributed\_to
* task\_avg\_completion
* task\_avg\_lag
* task\_avg\_work
* task\_completion\_time
* task\_deadline
* task\_ee\_id
* task\_status
* total\_completion\_time
* total\_lag\_time
* total\_work\_time
* Work

### Create

To create an expression rule:

1. In your application, go to the Build view.
2. Click NEW > Expression Rule.
3. In the Create Expression Rule dialog, configure the expression rule properties.
4. Click CREATE.

The expression rule opens in a new dialog or window, where you can:

* Add rule inputs.
* Enter an expression to define the rule.
* Test the rule input.

### Add rule inputs:

Rule inputs allow you to dynamically pass values into your expression.

For example, if you want to return information about a single customer, you could create an expression rule that queries the Customer record type, and use a rule input to filter the query so it only returns data for a specific customer ID.

To create a rule input:

1. In the expression rule, click New Rule Input.

If you don't see this option, expand the right-hand pane of the expression rule.

1. Configure the following properties for your rule input:

### Editing rule inputs:

As naming conventions and requirements evolve, you may need to update the names of your inputs to certain objects. Over time, rules and interfaces can gather many dependents. This can make manual changes to update any keyword references to rules and interfaces difficult and prone to human error.

If you update the name of an input while editing an expression rule or interface, Appian will update any keyword references to that rule input in dependent objects.

There are certain cases when Appian cannot update the dependent object(s):

1. The design object is locked.
2. The design object is not an expression rule or interface.
3. The design object does not use keyword syntax.
4. There are more than 20 direct dependents.

If there is a blue banner in the rule inputs pane, Appian is going to attempt an automatic update of the keyword references.

After you save your changes, a dialog will appear with a summary of what dependent objects were updated and any that you may need to update manually.

If you accidentally exit the rename dialog before you've completed all manual updates, use design guidance to find the remaining objects to update. Any object using the wrong keyword reference will be flagged with the invalid keyword syntax design guidance on the health dashboard.

All objects that are updated automatically as a result of the rename will show the developer who renamed the input as the last modifier for the dependent objects.

### Define the rule:

To define your expression rule:

1. In the left-hand pane of the expression rule, use the expression editor to enter an expression, or use the query editor to populate a query using a guided experience.

For example, the following expression rule uses an a!queryRecordType() expression to return all critical cases for a specific customer.

1. Test the expression rule. If you have rule inputs configured, you can add test inputs to test the values returned by your expression.
2. Click SAVE CHANGES.

### Test the rule:

Testing your expression rule allows you to see what the expression rule outputs based on a given set of inputs.

There are two ways to test your expression rule: using ad hoc testing and running test cases.

See Expression Rule Testing for more information.

### Ad hoc test:

By default, the Ad Hoc Test view displays whenever you open an expression rule.

The Ad Hoc Test view allows you to quickly test your expression by providing sample values or expressions for your rule inputs.

### Test cases

Test cases view can be created and saved with an expression rule. This allows you to run test scenarios in bulk and review the results for all the tests together.

The Test Cases view displays all test cases. From this view, you can also:

* Create new test cases
* Edit test cases
* Run all test cases
* Load test cases
* Duplicate test cases
* Update test case assertions in bulk
* Delete test cases

### Create a test case

1. In the expression rule, go to the Test Cases view.
2. Click NEW TEST CASE.
3. In the Create Test Case dialog.
4. Click CREATE.

### Save ad hoc test values as a test case

You can save your sample rule input values for later use by saving them as a test case.

To save your ad hoc test values as a test case:

1. From the Ad Hoc Test pane, under Test Inputs, enter an expression or value for your rule inputs.
2. Click Save as Test Case.
3. In the Create Test Case dialog, configure and review .
4. Click CREATE. The new test case appears in the Test Cases view.

To use a test case in the Ad Hoc Test view, select a test case and click LOAD TEST CASE.

### Assertion examples and recommendations

The following examples demonstrate how to use test case assertions to validate that your expression rules are returning the expected results for a variety of scenarios.

Example 1: Expression rules commonly validate that a value is what is expected in a process or interface. Below is an example of test cases for a rule that checks if an email address is valid.

Example 2: Another common scenario for expression rules is to manipulate a complex data type or Appian object input which could result in varied outputs. Examples of useful assertion expressions in these scenarios are below.

### Assertion recommendations:

### Expression rules should have enough test cases to accurately describe the behavior of the rule. This increases confidence that the rule works properly for runtime scenarios and helps with maintenance of the rule as it changes over time.

It is not necessary to have an exhaustive list of all possible permutations for your test cases.

* Avoid using the and() function to group assertions together when making an assertion expression since this will make debugging the rule's failed tests more difficult. Duplicate test cases and update their assertion expressions when you want to assert multiple things about the same output.
* When it is necessary to reference an Appian object in an assertion, use the Output matches the asserted output option to select the object. Objects selected from this option will be identified as precedents to the expression rule. It is also acceptable to create test objects and rules to use in test case assertions when necessary.

### Run test cases

Once you create your test cases, you can run a single test case or all test cases to see if they pass or fail.

Test results are not persisted with the expression rule on save, so tests must be run each time the rule is opened to view the results.

In addition to running test cases for an individual expression rule, tests for multiple rules can be run in bulk to test an entire application or system at once. See the Automated Testing for Expression Rules page for more details.

To run a single test case:

1. In the expression rule, go to the Test Cases view.
2. Select the checkbox next to the desired test case.
3. Click RUN SELECTED.

**To run all test cases:**

1. In the expression rule, go to the Test Cases view.
2. Click RUN ALL TESTS.

The results of these tests are displayed in the Test Cases grid, which includes the evaluated output of the rule for each test case and the test's status.

A test case can have one of the following resulting statuses:

* status OK Test passed - the test's assertion passed
* status warning Test failed: test output did not match asserted output - the test output was not identical to the user specified output
* status warning Test failed: assertion expression returned false - the user specified assertion expression did not evaluate to true
* progress stopped Test failed to run - the test could not be run because either one of its test inputs or the defined assertion failed to evaluate. When the test input failed, an error message is displayed in the cell for the corresponding input.
* progress stopped Test returned an error - the rule encountered an expression error during evaluation. An error message is displayed in the output column for that test case.

Export and import of test values

Test cases are always exported with the expression rule, but can only imported if the destination environment has the Allow Test Values to Be Imported with Design Objects setting enabled.

For more about this configuration, see the Deployment section of the Admin Console.

### Edit:

To edit an expression rule:

1. In your application, go to the Build view.
2. Select the expression rule name to open it.
3. Make any necessary updates.
4. Click SAVE CHANGES.

Each time you modify and save an expression rule, a new version is created. All expressions that use the rule use the latest version. All versions are accessible to developers who can view the rule, and a rule can be reverted back to a previous version at any time.

### Design guidance

While editing your expression rules, you may encounter guidance. Appian design guidance reinforces best practice design patterns that should be implemented in your objects. Design guidance for expression rules is calculated while editing the rule or when a precedent of the rule is updated.

When a recommendation or warning is triggered, you'll see an indicator icon in the header (A) and next to the corresponding line in the expression editor (B).

Click on the icon in the header to learn more about the suggested guidance (C) and how it can be addressed.

If a recommendation is not applicable to your use case, you can Dismiss (D) that individual recommendation for that rule. Learn more about recommendation dismissal. Warnings cannot be dismissed and should always be addressed to avoid complications when the logic in the object is executed.

### Move

Users with Administrator permission to this object or rule folder can move it to another folder:

1. Navigate to the parent folder that the object or folder is in.
2. Select the object or folder. You can select more than one.
3. Click the Move button on the toolbar. The Move Objects dialog appears.
4. Use the folder picker or browser to select the new parent folder.
5. You can also create a new folder in the browser, if you have Editor permissions to its parent folder.
6. Click Move.

### Duplicate

Developers with Viewer permissions to this object can duplicate it. There are two ways to duplicate an object:

* From any view in an application, you can select the object you wish to duplicate and use the toolbar option to launch the duplication dialog. The duplicated object will be added to the application you're working in. This capability is only available for single object selections from the grid.
* If you are in an interface, expression rule, integration or decision object, you can select Duplicate from the object's settings menu . From there, you can specify the target application for the new object.
* You can only duplicate the most recent version of an object. If you have unsaved changes and attempt to duplicate the object from within the designer, the most recent saved version of the object will be duplicated without the unsaved changes. A banner appears when there are unsaved changes to remind you before duplicating.

### Delete

Deleting an expression rule prevents users from further viewing or editing it. However, the last version of the rule is still available to be used in processes, record views, and reports.

Rules can be deleted by users with Administrator permissions to it. Appian does not recommend deleting rules that are in use because the rule can no longer be exported.

To delete an expression rule:

1. In your application, go to the Build view.
2. Select the checkbox next to the expression rule you want to delete.
3. Click DELETE.
4. In the Delete Expression Rule? dialog, ensure there are no objects that depend on the expression rule. If there are objects that depend on the expression rule, then deleting the expression rule will cause an error to appear in those objects.
5. Click DELETE.

### Security

Any user can invoke any expression rule that is used by an interface or process they can access.

The security role map of an expression rule controls which developers can see or modify it and its properties.

By default, expression rules inherit the security of the folder that they are saved in. However, after creating the expression rule, you can disable that inheritance and modify the rule's security.

See Editing Object Security to modify an expression rule's security.

# Chapter-6

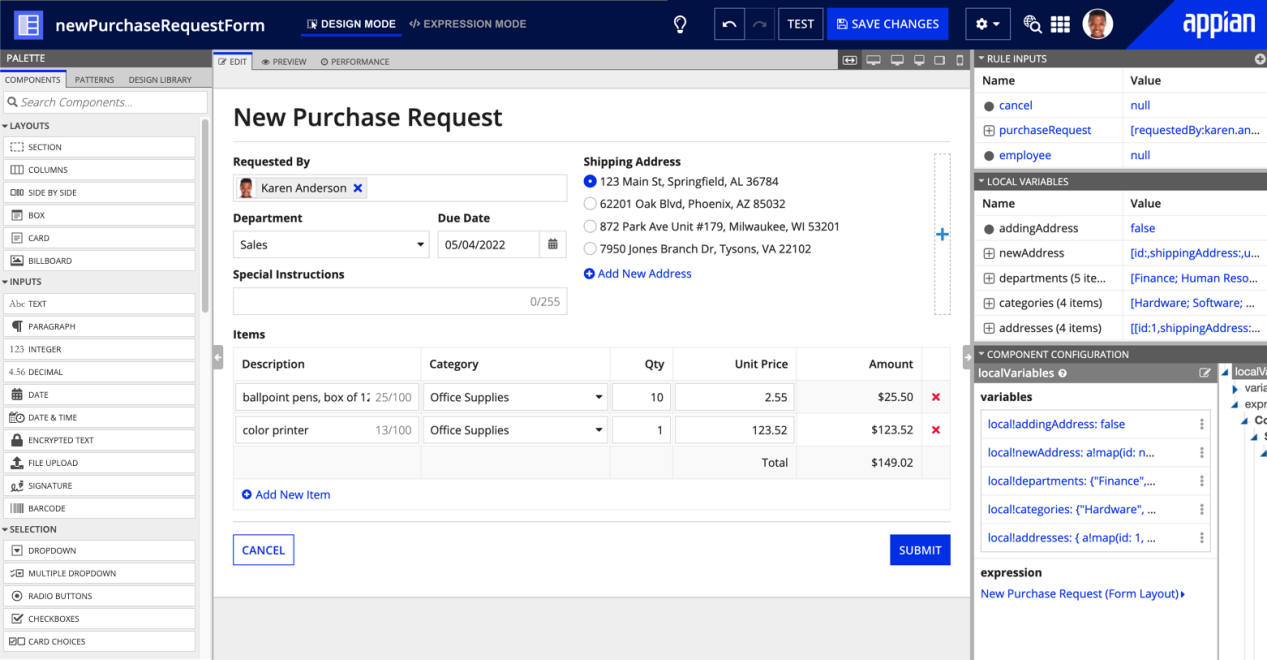
## Interface Object

### Overview

This page describes the features and functionality available when creating interfaces.

Interfaces can be built in design mode by dragging and dropping components from the palette or in expression mode by generating components via expressions.

If you are familiar with these concepts and want help in building your interface, see the pages in the Build section of this collection.

Fig:-6.1

For functional examples of different design patterns, see the Interface Recipes page. These examples will help you learn key component concepts and can be used as a starting point for your implementation.

Create an interface object

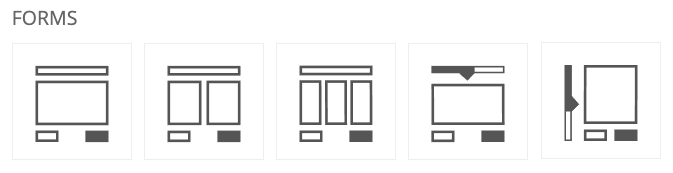
To create an interface object, select NEW > Interface in the Build view. Configure the initial properties for your interface. Then, Appian displays an interface ready for you to define.

Start with a template

When the interface has not yet been defined, a list of template options are available in design mode (shown below). These options give you a starting point from which to build your interface.

There are four categories of templates, (1) forms, (2) pages, (3) examples, and (4) builders.

**Forms:**

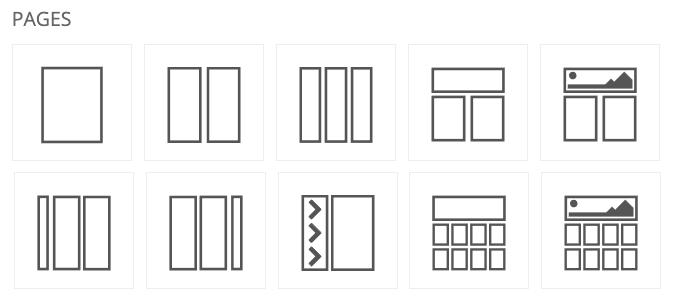
****

All forms come with at least a Submit and Cancel button. The different form options represent different arrangements of components and columns.

The Wizard-based templates each contain a functional, multi-page wizard with a confirmation at the end. To modify this for use in your interface, add your own components to the pre-defined pages and add new pages as necessary following the same pattern.

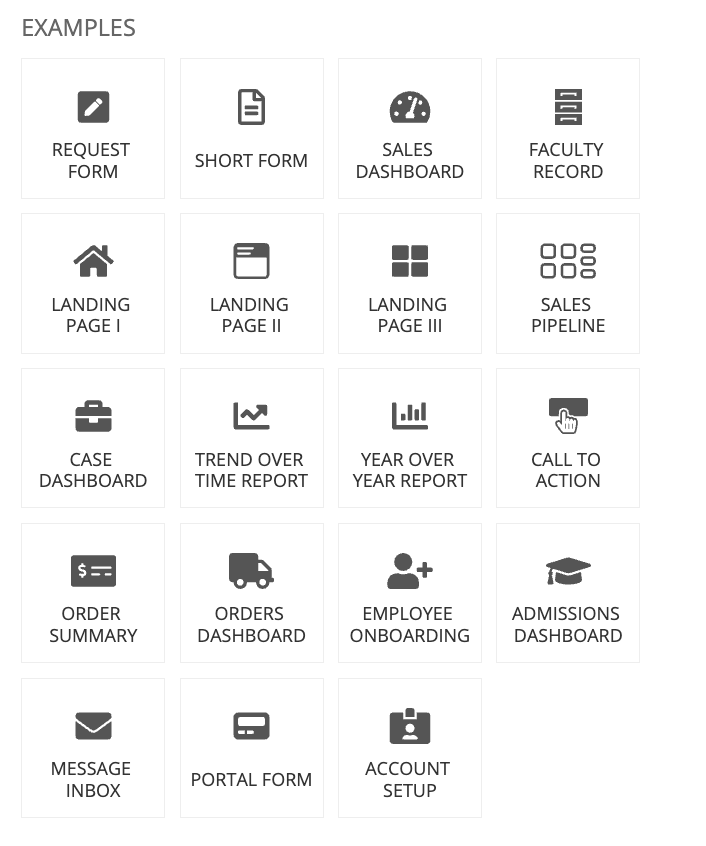
Outlines generate basic expressions for common layouts and use cases. These templates provide structural skeletons to use as a quick starting point that will be enhanced into a richer interface.

**Page:**

****

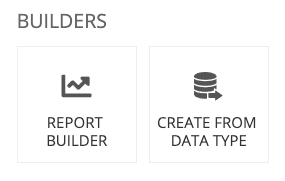
Pages provide only simple arrangements of columns and sections.

**Examples:**

****

Examples are fully-formed interfaces that demonstrate the richness, density, and dynamism that Appian interfaces can offer. These templates are a great way to get ideas on how to display data and to learn how to apply best practices. However, it is not advised to use these templates as a base that you edit to create your own interface. These examples are configured to use specific test data, and you would have to spend significant time editing the example template to adapt the expressions and inputs to your business case.

**Builders:**

****

Builders allow you to quickly create an interface using existing data in your system.

### Create from data type

This option will populate an interface with fields corresponding to the definition in a Custom Data Type (CDT). Simply select a CDT, and choose whether you want the interface to be editable, read-only, or both—then the builder will do the rest. Once the builder is done (it's really fast), Appian displays the generated interface.

All fields defined in the CDT appear as components in the interface. The component type will be a best-match based on the data type of the corresponding field. For example, a text field will be represented with a Text Component in the interface; a date field will be a Date Time Component, etc.

To create an interface based on a CDT:

1. In the Builders section of the templates pane, click Create from Data Type.
2. In the Data Type field, start entering the name of the CDT you want to use, then select the CDT from the dropdown list that displays.
3. Specify what type of action you want users to perform in the interface:

* Editing Data: Users can edit the CDT data in the interface.
* Displaying Data: Users can view CDT data in the interface.
* Both: Users can either view or edit CDT data in the interface, based whether or not the interface is in read-only mode.

1. Click Generate. Appian displays the generated interface.

These rule inputs and interface components are connected as follows:

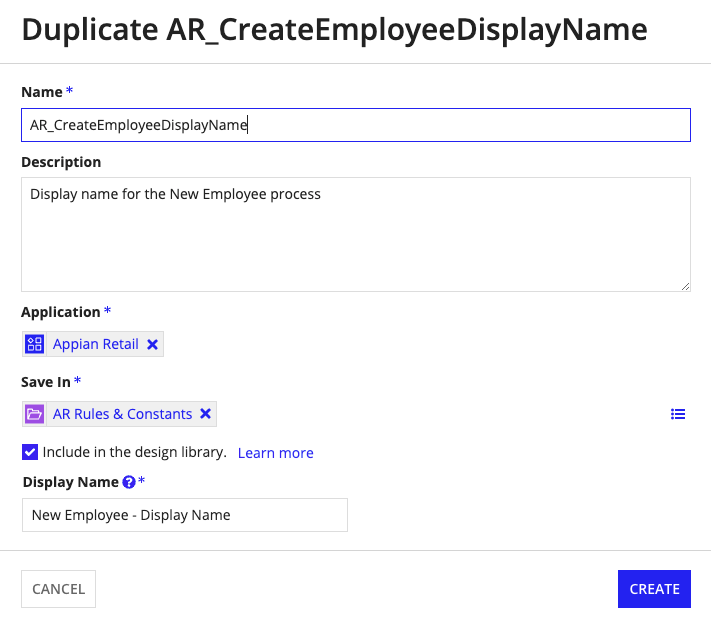
* selected CDT (CDT): All fields defined in the CDT are already connected to the pre-populated components, including the Submit button.
* cancel (Boolean): This rule input is connected to the Cancel button.
* readOnly (Boolean): This rule input is referenced by every component's Read Only parameter. When true, the components will not be editable. This is a common design pattern for reusing the same interface as both the input form and the display form. See Reusing Interfaces for more information.

### Duplicate an interface

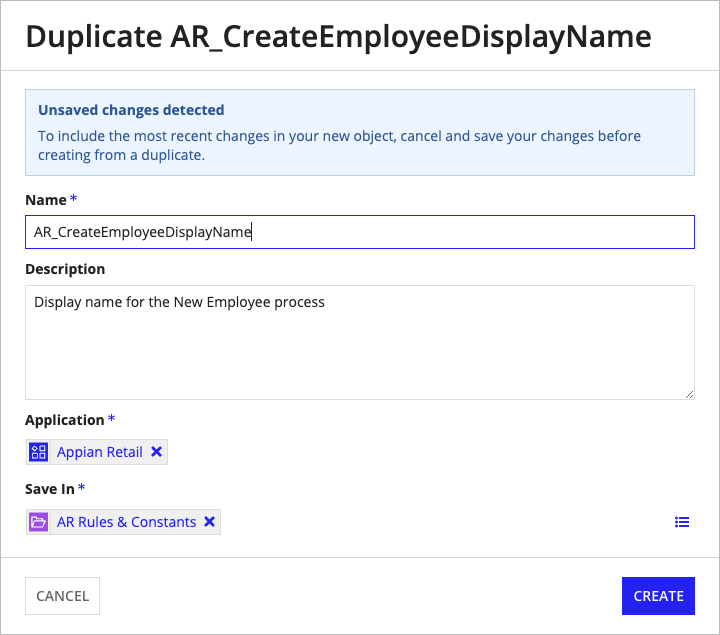
Developers with Viewer permissions to this object can duplicate it. There are two ways to duplicate an object:

* From any view in an application, you can select the object you wish to duplicate and use the toolbar option to launch the duplication dialog. The duplicated object will be added to the application you're working in. This capability is only available for single object selections from the grid.
* If you are in an interface, expression rule, integration or decision object, you can select Duplicate from the object's settings menu . From there, you can specify the target application for the new object.

Once you select the Duplicate option, you will see the following dialog:

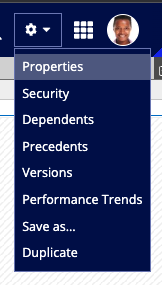


You can only duplicate the most recent version of an object. If you have unsaved changes and attempt to duplicate the object from within the designer, the most recent saved version of the object will be duplicated without the unsaved changes. A banner appears when there are unsaved changes to remind you before duplicating.



**Set interface properties**

To modify the interface properties, go to Properties in the Settings menu, as shown below.



This opens the Interface Properties dialog, where you can modify the description, modify the folder, make the interface available for offline use, and add the interface to your design library

### Editing rule inputs

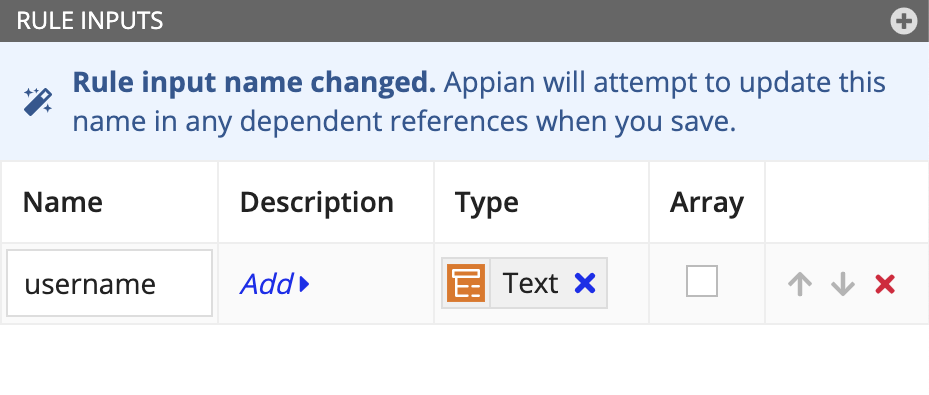
As naming conventions and requirements evolve, you may need to update the names of your inputs to certain objects. Over time, rules and interfaces can gather many dependents. This can make manual changes to update any keyword references to rules and interfaces difficult and prone to human error.

If you update the name of an input while editing an expression rule or interface, Appian will update any keyword references to that rule input in dependent objects.

There are certain cases when Appian cannot update the dependent object(s):

1. The design object is locked.
2. The design object is not an expression rule or interface.
3. The design object does not use keyword syntax.
4. There are more than 20 direct dependents.

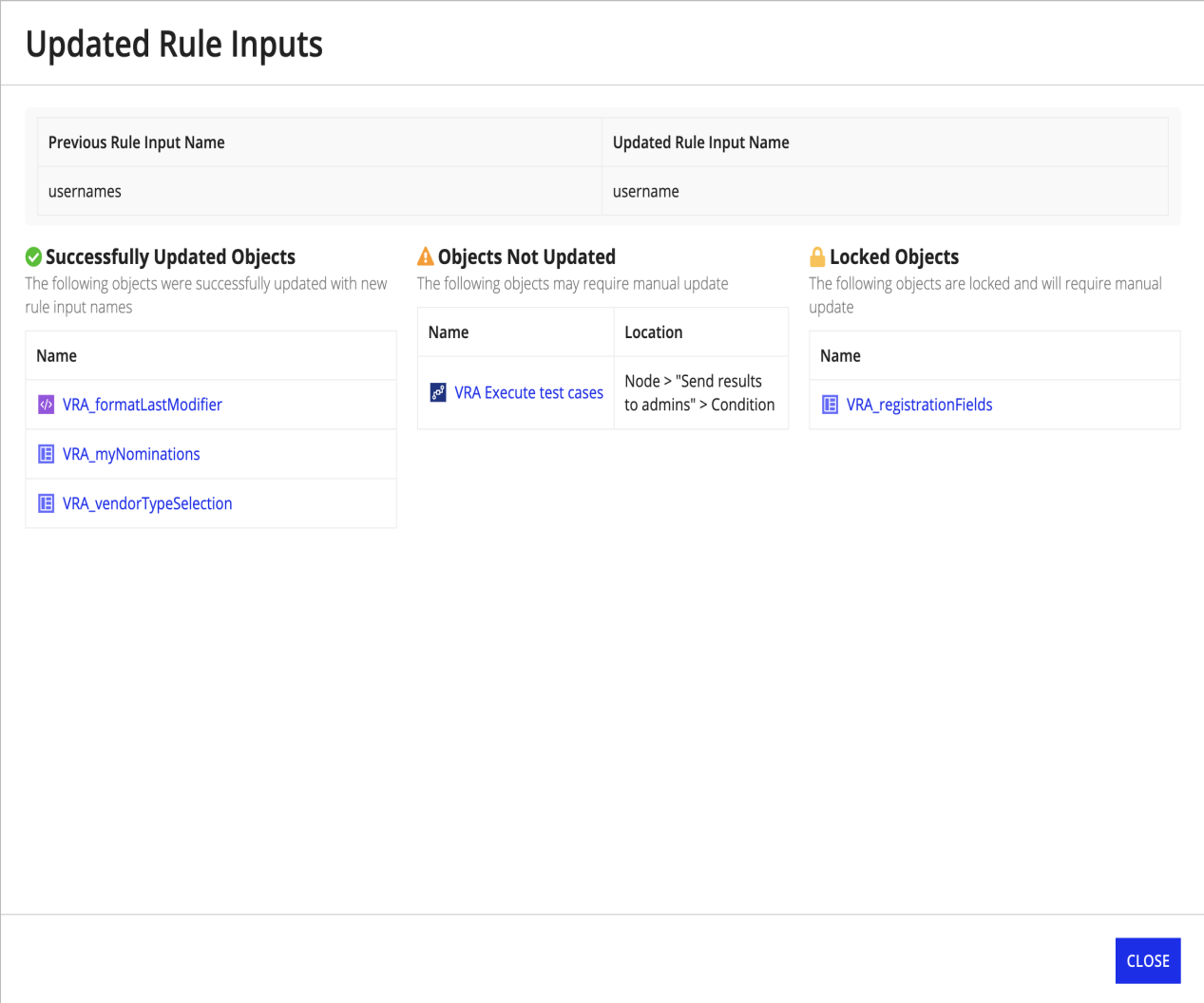
If there is a blue banner in the rule inputs pane, Appian is going to attempt an automatic update of the keyword references.



After you save your changes, a dialog will appear with a summary of what dependent objects were updated and any that you may need to update manually.

If you accidentally exit the rename dialog before you've completed all manual updates, use design guidance to find the remaining objects to update. Any object using the wrong keyword reference will be flagged with the invalid keyword syntax design guidance on the health dashboard.

All objects that are updated automatically as a result of the rename will show the developer who renamed the input as the last modifier for the dependent objects.



If you accidentally exit the rename dialog before you've completed all manual updates, use design guidance to find the remaining objects to update. Any object using the wrong keyword reference will be flagged with the invalid keyword syntax design guidance on the health dashboard.

All objects that are updated automatically as a result of the rename will show the developer who renamed the input as the last modifier for the dependent objects.

### Edit an interface

When you save a new version of an interface, the latest version will be available immediately. This means that record views, reports, process tasks, and other interfaces that use this interface will immediately use the new version. It is important to carefully consider the impact on running processes when changing interface definitions.

Appian recommends the following best practices to help you manage changes to interfaces:

* When calling rules in your interface definition, pass rule inputs by keyword.
* Take advantage of database-backed records and design short-lived processes. See the Database-Backed Record Type Tutorial page for more information.
* If you need the version of your interface to remain in sync with the version of your process, create new interfaces instead of new versions. If you need to create a new version of the interface, create conditional logic to execute the new behavior in the new version. Either approach will let older process instances continue to use the old interface behavior.

### Edit an interface

When you save a new version of an interface, the latest version will be available immediately. This means that record views, reports, process tasks, and other interfaces that use this interface will immediately use the new version. It is important to carefully consider the impact on running processes when changing interface definitions.

Appian recommends the following best practices to help you manage changes to interfaces:

When calling rules in your interface definition, pass rule inputs by keyword.

Take advantage of database-backed records and design short-lived processes. See the Database-Backed Record Type Tutorial page for more information.

If you need the version of your interface to remain in sync with the version of your process, create new interfaces instead of new versions. If you need to create a new version of the interface, create conditional logic to execute the new behavior in the new version. Either approach will let older process instances continue to use the old interface behavior.

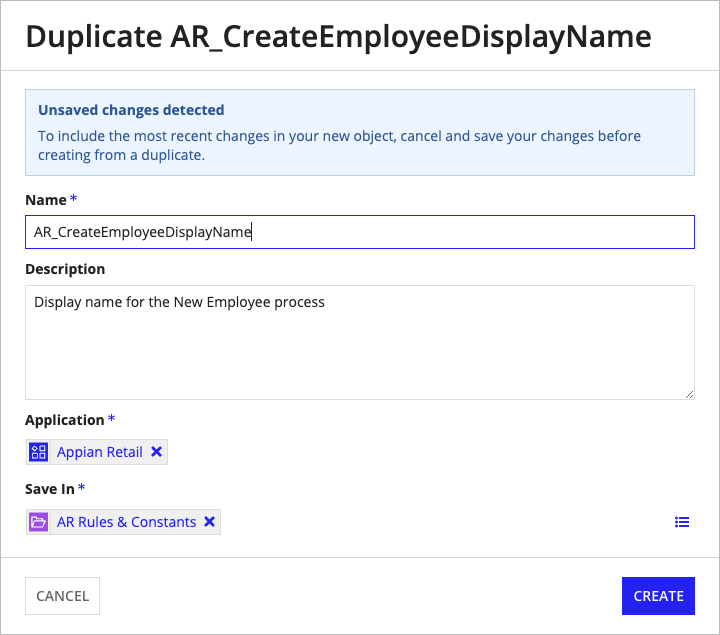
Speed up interface design with keyboard shortcuts.

**Versions**

1. Each time you modify and save an interface, a new version is created. All objects that use the interface will use the latest version. All developers that can view the interface can edit it, and interfaces can be reverted to a previous version at any time.
2. For information on how to manage object versions, see Managing Object Versions.
3. Design guidance
4. While editing your interfaces, you may encounter design guidance. Appian design guidance reinforces best practice design patterns that you should use in your objects. You'll see design guidance for interfaces while editing expressions within the interface or when updating an interface precedent.
5. When a recommendation or warning is triggered, you'll see an indicator icon in the header (A) and next to the corresponding line in the expression editor (B).
6. Click on the icon in the header to learn more about the suggested guidance (C) and how to address it.
7. If a recommendation isn't relevant, you can Dismiss (D) the recommendation. Learn more about recommendation dismissal.
8. Warnings cannot be dismissed and should always be addressed to avoid complications when the logic in the object is executed.

You can also find interface guidance on the Health Dashboard.

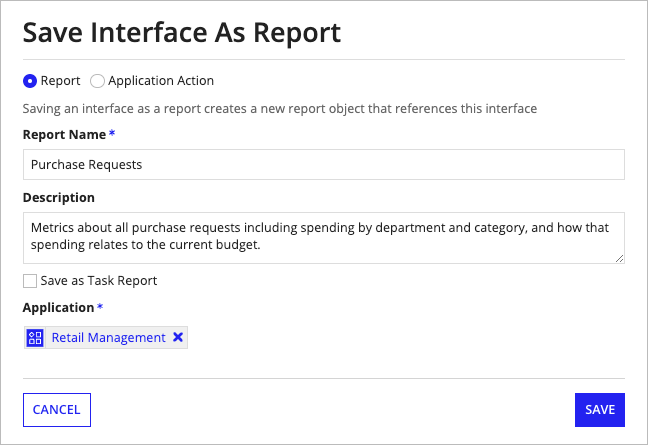
Design guidance is only provided for the interface's saved default test values. See design guidance for the full list of possible guidance.



**Save as report**

You can create a report directly from within the interface object. This automatically creates a report that calls the interface and makes it available from the Reports tab in Tempo.

To save an interface as a report, use the Report option in the Save Interface As dialog.

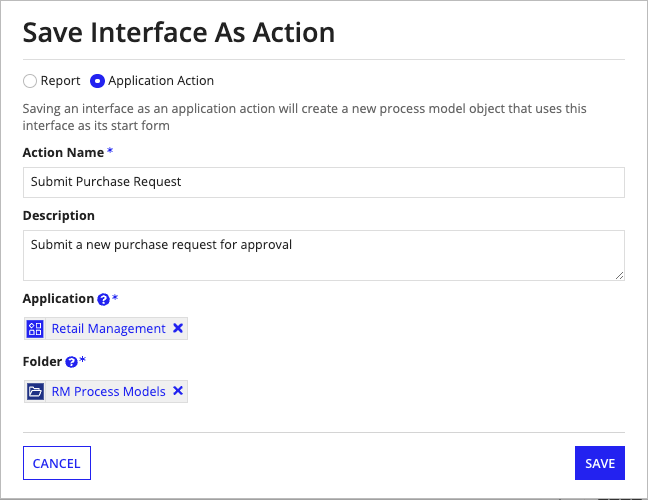


After clicking Save, you will be prompted to set security on your new report. The report will automatically be added to the selected application.

**Save as application action**

You can create an application action directly from within the interface object. When you do, Appian automatically creates a process model with a start form that calls the interface. The start form includes a process parameter for each input. Then, Appian creates an action on the selected application, making it available from the Actions tab in Tempo.

To save an interface as an action, use the Application Action option in the Save Interface As dialog.



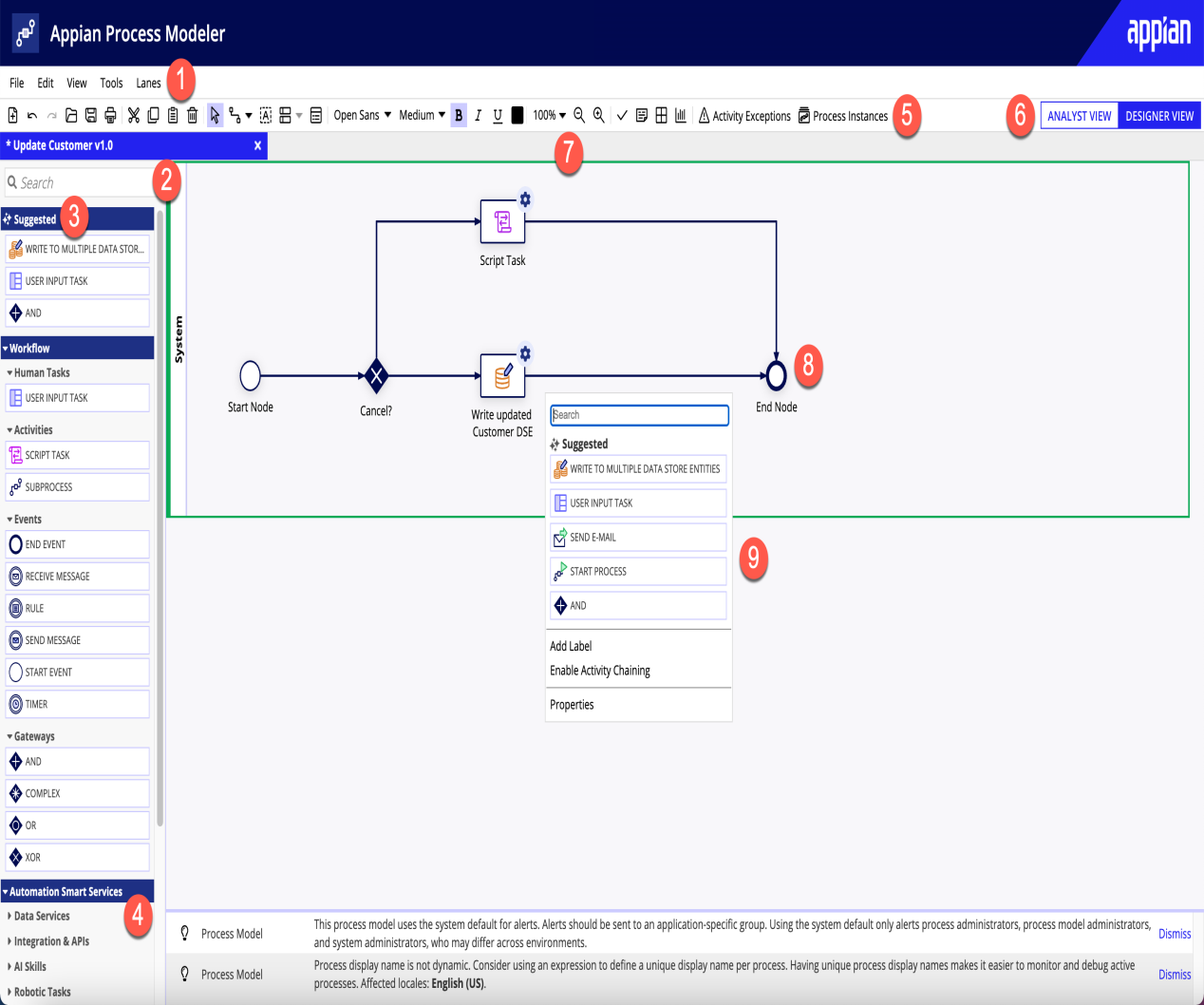
After clicking Save, you will be prompted to set security on your new process model. The process model will automatically be added to the selected application and the application will be published

# Chapter-7

## Process Model Object

### Overview:

This page discusses the basic features and functionality of process models and the Process Modeler.



1. Menu Bar: The menu bar has a multitude of features that are helpful when working with process models. See Menu Bar Options for a full list of features.
2. Smart Service Search Bar: This search bar allows you to quickly search for any smart services.
3. Suggested Smart Service Palette: This section provides you with AI-based suggestions for the next smart service or node to add to the canvas.
4. Workflow and Automation Smart Service Palette: The palette lists out all nodes and smart services. They are grouped by two categories—Workflow and Automation Smart Services. If the smart service is a custom plugin the icon appears next to the listing in the palette.
5. Process Model Toolbar: The toolbar has a collection of functions that will help you easily create and manage your processes when working on the canvas. See the Toolbar Options section for a full list of features.
6. Process Model Role View: You can select the Process Analyst View or the Process Designer View in a Process Model. The Process Analyst View allows you to draft high-level diagrams, while the Process Designer View allows you to configure advanced options and publish process models.
7. Process Model Canvas: The canvas is where you can drag nodes and start connecting your activities.
8. Suggested Smart Service Popup: Right-click on a connector between two nodes to search smart services and nodes, as well as see AI-based suggestions.

### Creating a process model:

Basic users must be a member of the Process Model Creators group in order to **create new process models.**

To create a process model:

1. In the Build view, click NEW > Process Model.
2. Configure the following properties:
3. Click CREATE.
4. Configure security for the process model.
5. Click SAVE. Appian creates and publishes the process model, then opens it in the Process Modeler. Learn how to start designing your process in the Process Modeler.

### Process analyst view

The Analyst View provides flowchart tools with standard activities, events, and gateways—as well as the set of Appian smart services. Each of these process nodes has limited configuration options, allowing an analyst to describe the overall process without configuring data, users, and other node properties such as a task form.

To create a process model that your application can actually use after designing a flowchart in this view, you must switch to the Designer View.

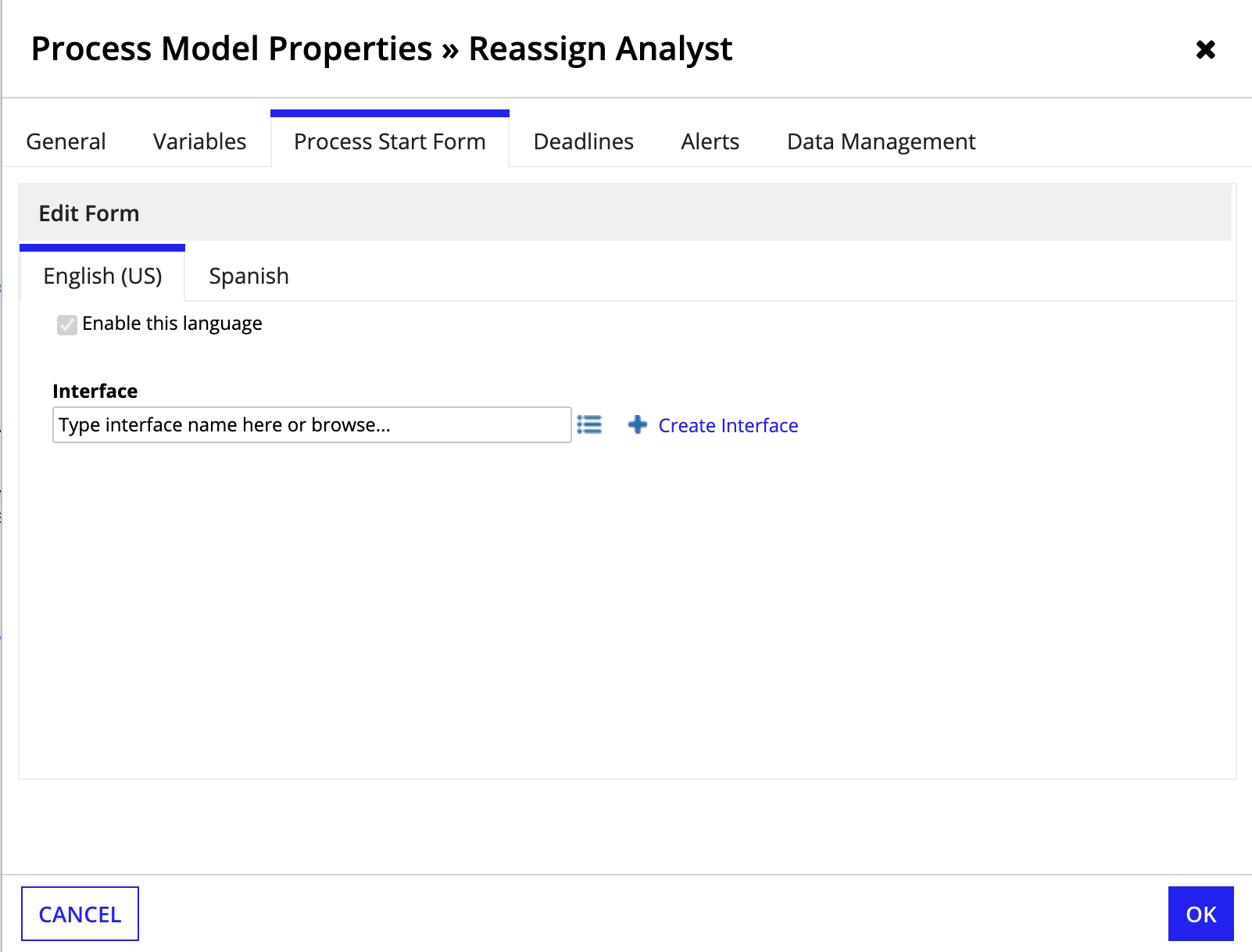
**Process designer view:**

* This view allows you to perform all of the modeling and documentation activities of the Analyst View, plus you can double-click each activity, gateway, or event and incorporate your business rules, variables, and forms
* The Search bar lets you find a node or smart service without needing to browse from the categories in the palette. Additionally, you can search for certain design objects and add them directly to your process model. Appian preconfigures the related smart service for you so it references the object you want to use. This is available for process model, integration (excluding RPA), and interface design objects.
* Type the name of the object you want to use. The object must be within the same application as the process model you're editing.
* Find the object in the list of results.
* Drag the object from the search results to the canvas. Appian preconfigures the corresponding smart service to reference that object.
* Open the process node's properties to further configure its behavior.

### Process start form tab

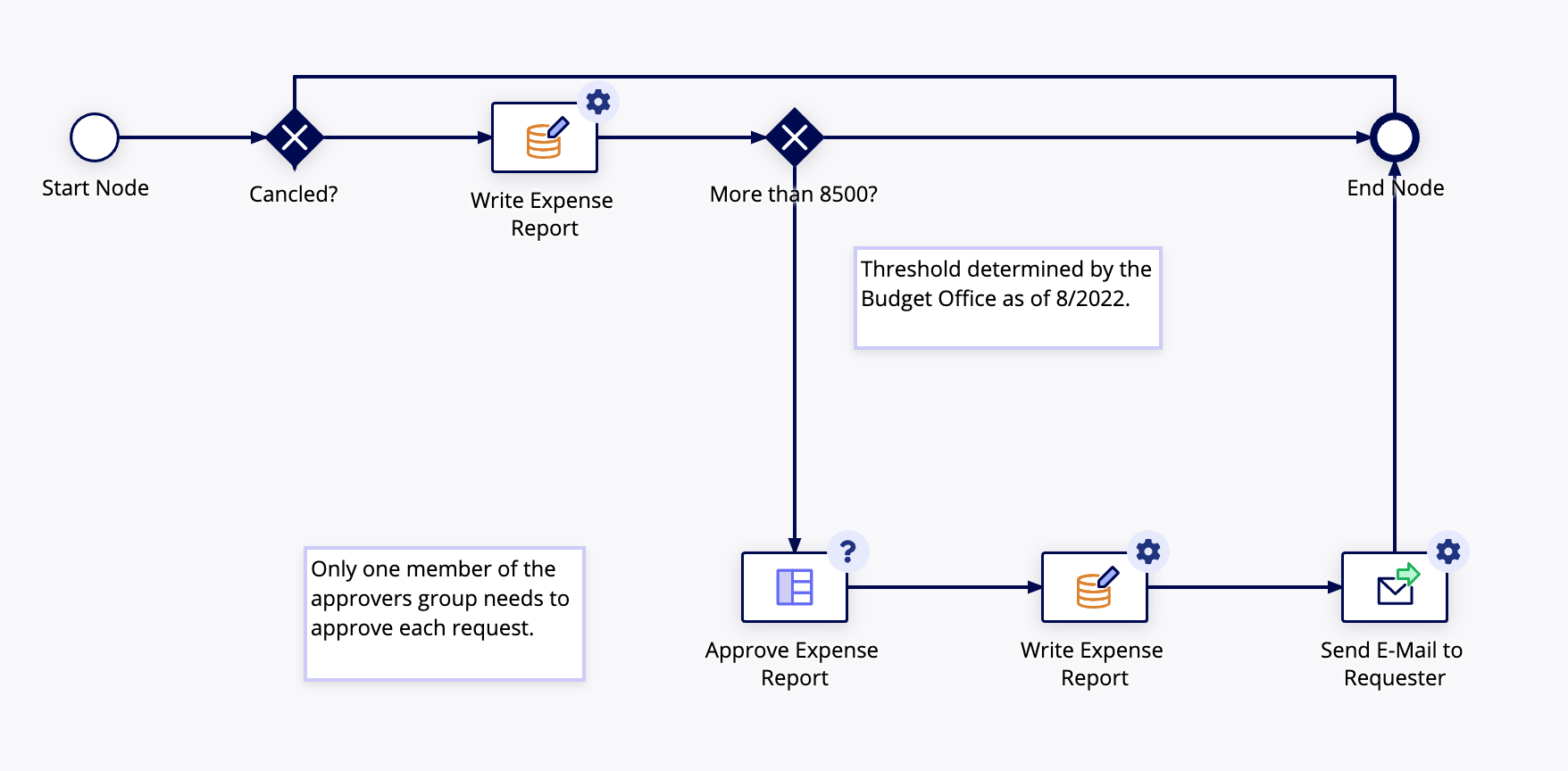
If you have data that must be provided to start a process, you can use a process start form to gather this data. The data collected by a start form must be saved into a process parameter.

* To add a start form, do either of the following:
* Create a new interface from existing process parameters by clicking Create Interface.
* Select an existing interface by searching within the Type interface name here or browse… search box.



### Canvas annotations

The annotate tool allows you to add text descriptions to your process models.



To add an annotation to the process model, complete the following:

1. In the Process Modeler, click the Annotate button on the toolbar.
2. When the pointer becomes a plus symbol +, click the Process Modeler canvas and drag the pointer to expand the text field.
3. Click inside the field you created.
4. Enter the text of your note.
5. If you enter more text than can be displayed in the window, scroll bars appear on the annotation. These can be removed by resizing the text window.

## Process model recommendation dismissal

* You can dismiss process model recommendations that do not apply to your use case by clicking Dismiss at the end of a recommendation. Learn more about recommendation dismissal here.
* In addition to the general recommendation dismissal behaviors, note the following about recommendation dismissals in the Process Modeler:
* Only recommendations that are dismissed on a published process model are saved. Any recommendations that are dismissed on a process model draft will reappear the next time that you open the process model.
* Recommendations can only be dismissed in the Process Designer View, and cannot be dismissed in the Edit Mode of an active process.

### Generating process model documentation

The Process Model Documentation view displays a report of your process model design and the objects it contains. The documentation view can be displayed from Appian Designer or the Process Modeler. Links displayed on the report display the details for the selected object. The report provides the following details:

* Process model properties
* Business Process Modeling Notation (BPMN) view of the process model
* Properties for each activity, event, and gateway
* Swimlane properties
* Annotations
* Process flow connection properties
* Viewing Process Model Documentation
* To view process model documentation, you must have Administrator or Editor rights to the process model.

To view this documentation from Appian Designer:

* In the Build view, select a process model.
* In the toolbar, click MORE.
* Select View Documentation. The latest published process model is rendered as documentation in a separate window.
* To view this documentation from the Process Modeler:

### Open a process model.

1. Select Tools > Generate Documentation from the toolbar. The current process model (saved or unsaved, published or unpublished) is rendered as documentation in a separate window.
2. Process model properties
3. The process model properties include the following:
4. Process Display Name: The process display name can be either a typed string of text or an expression.
5. Model Id : This report item lists an ID number used internally to reference the process model.
6. UUID : This report item lists a unique string of 32 hexadecimal digits assigned to the process model once you save it.
7. Last Updated: This report item lists the user name of the user who last saved the process model.
8. Learn how to use process model properties in expressions.

### Process model diagram

The report displays a BPMN view of the process model.

1. Click each activity to view its details.
2. Clicking a Subprocess activity displays the Subprocess diagram.
3. Process model contents
4. The Process Model Contents section of the report lists the objects that appear on the Process Modeler canvas.

**Process start form type**

If there is a process start form, it is listed as an interface.

**Process variables**

This section of the report lists each of the process variables configured for the process model, and the number of process variables in the process model. The following columns are displayed:

* Type: The type column lists the process variable's data type.
* Value: The value column lists any configured default value.
* Parameter: This column lists whether the process variable is required at the start of the process, as with a process start form.
* Required: This column lists whether a value must be stored for the process variable.
* Multiple: This column lists whether the process variable holds more than one value.
* Hidden: This column lists whether the process variable is hidden from reports.
* Lanes
* This section lists the swimlanes configured for process activities in the process model. The following columns are displayed:
* ID : A number used internally to reference the swimlane.
* Label: This report item lists the name of the swimlane as it appears on the process model.
* Height: This report item lists the height setting for the swimlane.
* Attended: An attended lane overrides individual assignment settings for attended activities in the lane, and sets assignment to this designated user or group.
* Assignees/Run As: When a swimlane is assigned, this column lists the user or group that the activities in the lane are assigned to. When activities are unassigned, this column lists the user account that is used to resolve security requirements.

### Node list

This section of the report lists all nodes that appear in the process model. The following columns are displayed:

* ID: A number used internally to reference the node at runtime.
* GUI ID: A number used internally to reference the node at design time.
* Name: The name displayed on the Process Modeler canvas.
* Task Display Name: The name displayed on any tasks assigned by the activity.
* Lane: The name of the swimlane that contains the node, if any.
* Attended: This column lists whether the node is assigned to a user or group as a task.
* Assignees/Run As: When an activity is assigned, this column lists the user or group that the activity is assigned to. When the activity is unassigned, this column lists the user account that is used for security permissions.
* Annotations
* This section of the report lists any notes that appear in the process model. The following columns are displayed:
* GUI ID: A number used internally to reference the note.
* Text: The content of the note.
* Connections
* This section of the report lists the flow connectors that appear in the process model. The following columns are displayed:
* Start Node: The process node where the flow connector originates.
* End Node: The process node where the flow connector terminates.
* Activity-Chained: This column lists flows that are configured to automatically display the next assigned task.
* Process model node details
* Each node that appears in the process model is listed in greater detail in this section. The following information appears for each node below its name and icon:
* Node Type: This line item lists the name of the type of node used as it appears on the Process Modeler palette. It also lists the internal object name in parentheses.
* Form Type: Smart service activities display the type of form configured (dynamic or JSP). A dynamic form uses the Process Modeler. JSP forms list the path to the JSP.
* UUID: This line item lists a unique string of 32 hexadecimal digits assigned to the associated node once you save the process model.
* Description: This line lists any descriptive text entered for the activity on the general tab of the node properties dialog.
* Task Display Name: The name displayed on any tasks assigned by the activity. A rule or expression may be used to resolve the display name at runtime.
* Lane: The name of the swimlane that contains the node, if any.
* Attended: This item lists whether the node is assigned to a user or group as a task.
* Assignees: When an activity is assigned, this item lists the user or group that the activity is assigned to. When the activity is unassigned, this item states that the node is run as either the process initiator or the process designer.
* Exception Flow Triggers: Smart services list any rules configured to trigger an exception flow.

### Event Nodes

Process events are listed according by name and Process Modeler icon. The following details are listed for each process event:

* Event Type: This line lists the type of event node, or the type of event configured for a node such as send message, or receive message.
* Conditions to activate event node: Each logic test that must be satisfied before the event is triggered is listed.
* Send Message events
* Message Type: This line lists the type of message being sent such as email or process.
* Message Body: This line lists the body text of the message that is sent.
* Mappings
* The event mappings are listed in a table that displays the following columns.
* Name: The name of each property.
* Value: The data listed for each property configured for the event mapping.
* Receive Message event
* Listen for messages of type: Typically, this property lists either email or process-to-process messages.
* Once activated, listen for messages for the life of the process: True/False. This property lists whether the an event is activated once, or whether it can be activated multiple times.
* Gateways
* Each gateway node is listed with its standard node properties. Each gateway also lists the conditions that route process flow to different nodes in a table. This table displays the following columns.
* Condition: The logical test that must be true to activate an outgoing flow.
* Result: The outgoing flow to activate, listed according to its destination node.
* Path Label: The label that appears on the outgoing flow (if any).
* Node Inputs table

**REFERENCES**

1. Appiandocumentation,Available:https://docs.appian.com/suite/help/23.1/Appian\_Data\_Types.html#:~:text=An%20Appian%20Object%20data%20type,recognizable%20within%20the%20Appian%20system
2. Interface Object, Available:https://docs.appian.com/suite/help/23.1/interface\_object.html .
3. Expressions, https://docs.appian.com/suite/help/21.4/Expressions.html.
4. Process Model https://docs.appian.com/suite/help/23.1/process-model-object.html.
5. CreatingGroup,https://docs.appian.com/suite/help/23.1/Creating\_Groups.html#:~:text=Appian%20users%20can%20be%20organized,further%20organized%20into%20group%20type.
6. DocumentFunction,https://docs.appian.com/suite/help/23.1/fnc\_scripting\_document.html