Oracle
Primavera
Gateway Scripting Guide

Version 22
August 2022

Contents

Overview

Primavera Gateway is an application that facilitates sharing data between applications. Data is shared between applications by running a synchronization in Gateway. A synchronization executes a job that transfers the data between applications. Each synchronization uses a business flow to transfer data between applications. Each business flow is made up of a combination of flow steps. These flow steps are executed in a specific order based on their assigned sequence number when data is transferred. You can modify the combination of flow steps in a business flow by including additional flow steps that perform calculations which suit your needs. These are called custom steps.

Custom steps can be directly added to a business flow using the Primavera Gateway user interface. This type of custom step is written in Gateway Scripting Language (GSL). In a Gateway scripted custom step, you can:

- Create or remove business objects
- Insert, modify, or delete fields of a business object

A custom step is always associated with a specific:

- Provider corresponding to the application supported in Primavera Gateway
- Business flow type (master data or project data)
- Flow side of a business flow which determines whether the custom step applies to the source or destination application
- Sequence number which determines when this custom step will run in a business flow

The Gateway Scripting Language Guide describes how to create a custom step in GSL in Primavera Gateway. It also provides a language syntax reference supported in GSL. For more details, see the ***Language Reference*** (on page 11).

Gateway developers and administrators responsible for creating business flows should use this guide.

Within our documentation, some content might be specific for cloud deployments while other content is relevant for on-premises deployments. Any content that applies to only one of these deployments is labeled accordingly.

In This Section

About Personal Information (PI)

Personal information (PI) is any piece of data which can be used on its own or with other information to identify, contact, or locate an individual or identify an individual in context. This information is not limited to a person's name, address, and contact details. For example, a person's IP address, phone IMEI number, gender, and location at a particular time could all be personal information. Depending on local data protection laws, organizations may be responsible for ensuring the privacy of PI wherever it is stored, including in backups, locally stored downloads, and data stored in development environments.

Caution: Personal information (PI) may be at risk of exposure. Depending on local data protection laws, organizations may be responsible for mitigating any risk of exposure.

Accessing the Script Editor

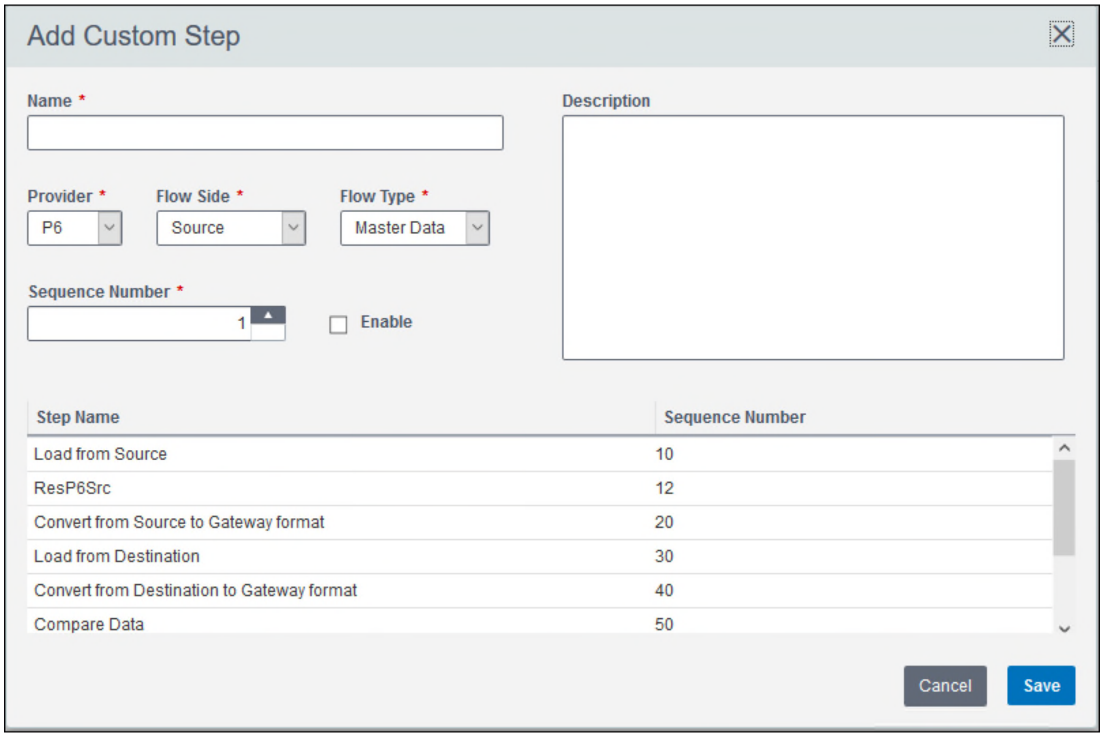
Access the Gateway script editor as follows:

1) In the sidebar, select **Configuration**.
2) Select the **Custom Steps** tab.
3) Select + **Add...** or ✎ **Edit...**.
4) In the **Custom Step** dialog box:
    a. Select a provider for which the custom step is being created from the **Provider** list.
    b. Select whether the custom step will be run from the source side or the destination side of a job from the **Flow Side** list.
    c. Select the business flow type associated with the custom step from the **Flow Type** list.
    d. In the **Sequence Number** field, enter the positioning of the custom step within the default sequence flow
    e. Select **Save**.
5) In the **Formula** section, enter the Gateway script and select **Validate** to check for errors.
6) Select **Save**.

Working with the Script Editor

Primavera Gateway includes a **Custom steps** tab in the **Configuration** menu that allows you to add or edit a custom step. It includes a script editor you can use to create valid Gateway scripts for defining custom steps.

Figure 1: Gateway Script Editor



Name *

Description

Provider * P6

Flow Side * Source

Flow Type * Master Data

Sequence Number * 1

Enable

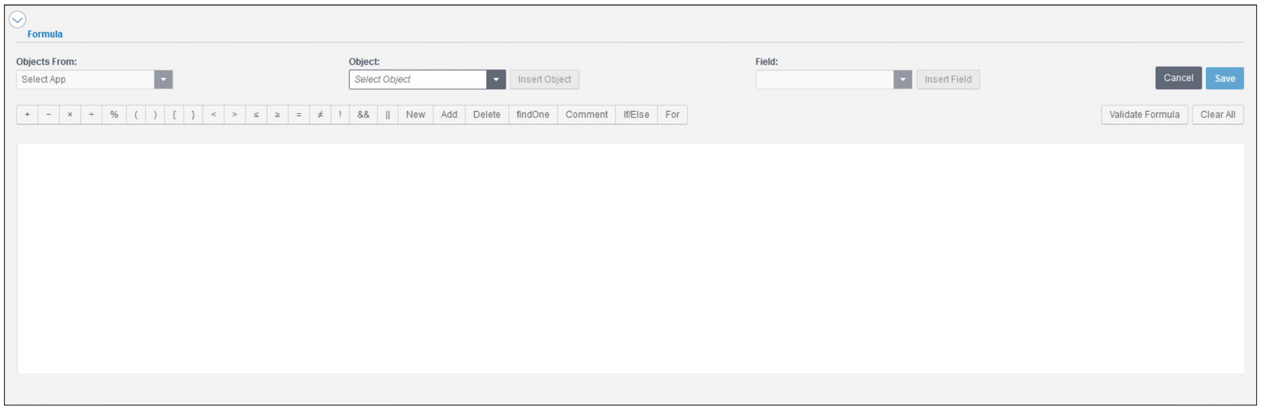
Step Name	Sequence Number
Load from Source	10
ResP6Src	12
Convert from Source to Gateway format	20
Load from Destination	30
Convert from Destination to Gateway format	40
Compare Data	50

The Gateway script editor allows you to:

- Select objects and fields available in a provider
- Enter Gateway expressions for the selected objects and fields
- Validate the script follows Gateway syntax rules

  On successful validation, the following message displays: *Syntax OK.*

Figure 2: Gateway Script Formula Section


Formula
Insert Object
Insert Field
Cancel
Save
+ − × ÷ % ( ) { } < > ≤ ≥ = ≠ ! && || New Add Delete findOne Comment If/Else For
Validate Formula
Clear All

Objects From: Select App

Object: Select Object

Field:

The custom script can now be used in a master data or project data business flow associated with the relevant provider.

Flow Step Sequence for Custom Steps in Gateway Scripting Language for Business Flows

For a specific provider, a custom step can be added to the flow step sequence of any business flow from within the Gateway user interface using Gateway scripting language. This flow step is an alternative option to:

- Adding a external custom step using Java
- Adding a internal custom step using Java

Note: Gateway scripting language is distinct from *Groovy* Scripting Language. For more details on how to code a custom step, see *Gateway Scripting Language Guide*.

A business flow is executed as an ordered sequence of flow steps. So, the positioning of a custom step depends on the role of the provider in a business flow.

Use the following table to position a custom step in the default flow step sequence:

If Provider Role in Business Flow is...	Add Custom Step...
Source	with a sequence number in the range 1 - 19.
Destination	with a sequence number in the range 61 - 79.

You can also add a custom step in Gateway between sequence numbers 21 - 59.

The steps can then be organized as follows:

- (Optional for Source Provider) **<Custom Step Name>:** This step runs the custom step to the source data. It can be added in the flow sequence for a *source* provider with a sequence number in the range 1 - 9.
- **Load from Source:** This step loads the data from the source application into the Gateway framework so that it can be processed.
- (Optional for Source Provider) **<Custom Step Name>:** This step runs the custom step to the source data. It can be added in the flow sequence for a *source* provider with a sequence number in the range 11 - 19.
- (Optional in Gateway) **<Custom Step Name>:** This step runs the custom step within Gateway. It can be added in the Gateway flowside sequence with a sequence number in the range 21 - 59.
- **Convert from Source to Gateway Format:** This step converts the source data to the Gateway format.
- **Convert to Destination Format:** This step converts the data from Gateway format to the destination format.

- (Optional) **<Custom Step Name>:** This step runs the custom step to the destination data. It can be added in the flow sequence for a *destination* provider with a sequence number in the range 61 - 69.
- **Review data:** This step enables you to review the source data before updating the data in the destination application.
- (Optional) **<Custom Step Name>:** This step runs the custom step to the destination data. It can be added in the flow sequence for a *destination* provider with a sequence number in the range 71 - 79.
- **Update Destination:** This step saves the data into the database of the destination application.

Note: Although custom steps can be added after the last **Update Destination** flow step (sequence number 80), these will not be processed by the business flow.

Writing a Gateway Script

The following is a Gateway script example which iterates over all of the roles and tries to find a resource that has the same id field. If one is found, the script will then search for a resource role matching the existing resource and role object id fields. If no match is found or there is no resource role for this role/resource, a new resource role object will be created and added.

```
for (role : <Role>) {
        <Resource> resource = <Resource>.findOne(r -> ( r.Id == role.Id ));
    if (resource != null) {
            <ResourceRole> myrr = <ResourceRole>.findOne(rr ->
((rr.ResourceObjectId == resource.ObjectId) && (rr.RoleObjectId == role.ObjectId)));
        if (myrr == null ) {
            <ResourceRole> resRole = new <ResourceRole>;
                resRole.ResourceObjectId = resource.ObjectId;
                resRole.RoleObjectId = role.ObjectId;
            resRole.ObjectId = -100;
                add resRole;
        }
    }
}
delete <Role>;
delete <Resource>;
```

Language Reference

Primavera Gateway scripting language supports several data types, operators, and statements. The language reference section provides detailed information on the data types and programming constructs.

These include:

- ***Data Types*** (on page 11)
- ***Operators*** (on page 12)
- ***Comparison Operators*** (on page 14)
- ***Special Operators*** (on page 16)
- ***Gateway Statements*** (on page 17)

In This Section

Data Types

Supported Data Type	Example	Note

Boolean	`Bool = true;`
	`OtherBool = false;`
		Boolean values can be set as either `true` or `false`.

Date	`Date DateExample = 1987-12-20T04:04:30`	The Date must be in the ISO format `yyyy-mm-ddThh:mm:ss`. No other formats are allowed.

Double	`double example1 = 5.5;`
	`double example2 -3.4;`
	`double example3 = 3;`	

Integer	`int z = 4 + 5.5`	Assignment expressions in math such as / or * that have double values as part of the expression will truncate the decimal. In the example, `int z = 9`

Object reference	`<Resource> res = new <Resource>;`
	`<Resource> res2 = null;`
	`<Resource> res3 = <Resource>.findOne(r → 2 == 2);`
		References to all the fields of the object are included by default once you declare an object variable.

null
Note: You can set a variable of any data type to null, but you cannot create a null type variable.
	`int i1 = null;`
	`double i2 = null;`
	`Date i3 = null;`
	`<Resource> myResource = null;`
	`myResource.ObjectId = null;`	

String	`String sample = "Hello World"`	

Operators

Gateway scripting language provides operators for computation and comparison. Operators can only be applied to data types they support. The following tables list operators supported in Gateway code.

- ***Numeric Operators*** (on page 12)
- ***Logical Operators*** (on page 13)

Numeric Operators

Supported Numeric Operators

Use numeric operators to perform calculations on numeric data types, such as doubles.

Operator	**Name**	**Description**	**Applicable Data Types**	**Return Data Type**	**Example**
+	Addition	Sums two Double values.	Double	Double	`2 + 5; //returns 7`
-	Subtraction	Subtracts two Double values.	Double	Double	`5 - 3; //returns 2`

*	Multiplication	Multiplies two Double values.	Double	Double	`2 * 5;`
					`//returns 10`

/	Division	Divides two Double values.	Double	Double	`10 / 2;`
					`//returns 5`
					`3 / 2;`
					`//returns 1.5`

Logical Operators

Supported logical operators to manipulate and combine Boolean values and to construct conditional statements.

Operator	**Name**	**Description**	**Applicable Data Types**	**Return Data Type**	**Example**

||	OR	Returns the result of combining Boolean values using a logical OR. If one value or expression contained in the OR statement evaluates to true, the OR expression returns true.	Boolean	Boolean	`true || false;`
					`//returns true`

&&	AND	Returns the result of combining Boolean values using a logical AND. If both values or expressions in the AND statement evaluate to true, the AND expression returns true.	Boolean	Boolean	`true && false;`
					`//returns false`

!	NOT or Inversion	Negates the specified Boolean value. Applying the NOT operator to an expression that evaluates to true will return false.	Boolean	Boolean	`!true;`
					`//returns false`

Comparison Operators

Use comparison operators to measure values against each other. The data types of compared values must match, otherwise the application will return an error. The result of a comparison operation is a Boolean, true or false.

Operator	Name	Description	Applicable Data Types	Return Data Type	Example

==	Equal	Tests if two values are equal.	All	Boolean	`1.0 == 1.0;`
					`//returns true`
					`1.0 == 2.3`
					`//returns false`
					`"Hello" == "Hello"`
					`//returns true`

!=	Does Not Equal	Tests if two values are not equal.	All	Boolean	`2 + 2 != 5;`
					`//returns true`
					`2 + 2 != 4;`
					`//returns false`
					`"Hello" != "Goodbye"`
					`//returns true`

>	Greater Than	Tests if one value is greater than another.	All	Boolean	`50 > 100; //returns false`
					`50 > 5; //returns true`
					`"2" > "10" //returns true`
					`"Hello" > "World" //returns false`
					`5 > "4" //error. The types of compared values must match.`

>=	Greater Than or Equal To	Tests if one value is greater than or equal to another	All	Boolean	`60 >= 50; //returns true`
					`60 >= 70 //returns false`
					`60 >= 60 //returns true`
					`"Hello" >= "World" //returns false`

<	Less Than	Tests if one value is less than another.	All	Boolean	`50 < 100; //returns true`
					`50 < 40 //returns false`
					`"2"< "10" // returns false`
					`"Hello" < "World" //returns true`
					`"3/1/17" < new Date() //error the types of compared values must match`

<=	Less Than or Equal To	Tests if one value is less than or equal to another.	All	Boolean	`60 <= 50; //returns false`
					`60 <= 90 // returns true`
					`60 <= 60 //returns true`
					`"10" <= "2" //returns true`

Special Operators

Use special operators to create structure in your Gateway code and specify how Primavera Gateway should interpret and execute the expressions.

Operato r	Name	Description	Applicable Data Types	Example
+	Concatenation	Combines a String with another data type, yielding a new String. String concatenation is applied when the leftmost operand of the + operator is of the data type String. The right operand can be a value of any type. When concatenated, Date values may not serialize in the format anticipated.	String	`"Hello" + " World" //returns "Hello World"` `"Lucky Number " + 7; //returns "Lucky Number 7"` `"a" + 1 + 2 //returns "a12"` `5 + "Hello"; //returns an error. To perform concatenation, the leftmost operand must be a String.`
=	Assignment	Assigns a value of a supported data type to a variable.	All	`def myVar = 50;`

Gateway Statements

Gateway supports a variety of statements you can use to determine the control flow of Gateway code.

These include:

Statement	Description	Example
abort	Stops a job and marks it as Failed with a message provided in the formula. You can use this statement to test for a specific scenario and if found, stop the job from processing. Syntax: `abort "<message to show why job is being aborted>";`	`abort "Invalid project ID.";`

add	Adds an object to the synchronization.
	Objects that are added will be sent to the destination.
		`add myResource;`
		`//Adds the myResource variable with all associated fields to the synchronization.`

attribute	Determines what action to take on a row of data. Valid actions to execute include: create or update. Syntax: `setAttribute("action", "<action to execute>");`	setAttribute("action", "create");

Child Object	Finds a child object of an object (Code/UDF) based on an object and used in the formula.
	Note: Use this statement to only find an object. You cannot create a new object.
		`Object childObj = <Parent ObjectName variable>.findOneChild (variable -> variable.fieldname`
		`functiontoperform variable.fieldname);`

delete	Removes objects and fields from the synchronization.
	Objects and fields that are removed, will not be sent to the destination.
	Note: Does not necessarily remove the object from the destination application.
		`//removes all resource objects`
		`delete <Resource>;`
		`//removes a single resource object`
		`delete myResoure;`
		`//removes the Id field`
		`delete myResource.Id;`

{ } (block)	Specifies a sequence of expressions to evaluate. Used to structure Gateway scripts, establish variable scope, and improve readability.
	Variables defined and assigned values within a code block are accessible only within the code block and other structures the code block contains.
		`def a = 8;`
		`{`
		`def b = 10;`
		`}`
		`return a + b; // error. B is not defined within scope.`

Field	Use a field within a formula after it has been removed by the Compare Step. Note: To use this option if the field will be removed, place the Custom Step after Compare and before the Convert to Destination Format step.	

for loop	Iterates over the objects of a specific type in Gateway.	//Iterates over all of the resource objects to find a resource rate that exists for the given resource and date. If no ResourceRate is found, a new one will be created. At the end, all resource objects are removed.
		for (resource: <Resource>) {
		    <ResourceRate> resRate = <ResourceRate>.findOne(rr2 -> (rr2.ResourceObjectId == resource.ObjectId) && (rr2.EffectiveDate == 2019-12-20T00:00:00));
		    if (resRate == null) {
		        <ResourceRate> rr = new <ResourceRate>;
		        rr.ObjectId = -100;
		        rr.EffectiveDate = 2019-12-20T00:00:00;
		        rr.ResourceObjectId = resource.ObjectId;
		        rr.PricePerUnit = 5;
		        add rr;
		    }
		}
		
		//remove all resource objects
		delete <Resource>;

find one	Returns an object when the given condition is met.	`<Resource> myRes = null;`
		`myRes = <Resource>.findOne( r → r. Id == "My Resource" );`
		`//Sets the myRes variable to the first resource that matches the given condition. If no resource is found to match the condition, myRes variable will be set to null.`

if	Code contained in an if block will only be evaluated if the condition specified by the if statement evaluates to true.	`if (1 + 1 == 2) {`
		`//block executed if condition is true`
		`return "True";`

if/else	Code contained in an if block will only be evaluated if the condition specified by the if statement evaluates to true.
	If the specified condition evaluates to false, code in the else block will be evaluated.
		`if (1 + 1 == 2) {`
		`//block executed if condition is true`
		`return "True";`
		`} else {`
		`//block executed if condition is false`
		`return "Not true!";`
		`}`

if/else if	Code contained in an if block will only be evaluated if the condition specified by the if statement evaluates to true.
	If the specified condition evaluates to false, code in the if else block will be evaluated.
	If the specified condition evaluates to false in the if else block, then the next block of if else code will be evaluated.
	Note: Any number of if else blocks can be specified.
		`if (1 + 1 < 2) {`
		`//block executed if condition is true`
		`return "True";`
		`} else if {`
		`//block executed if condition is false`
		`return "Not true!";`
		`} else if {`
		`//block executed if condition is false`
		`return "Equal";`

new	Creates an empty object.	`<Resource> myResource = new <Resource>;`
		`//Creates an empty resource object and assigns it to myResource variable. At this point the myResource variable is not null but it has not been added to the synchronization and will not be sent.`

parameter	Creates a parameter using the following syntax:
	Parameter["parameter name", "parameter provider"] = "parameterValue";
		Parameter["SomeParameter", "Unifier"] = "someValue";
		Parameter["parameter name", "Primavera Cloud"] = "parameterValue";
		string someVar = Parameter["SomeParameter"];
		Parameter["SomeParameter", "Unifier"] = "someValue";
		string someVar = Parameter["SomeParameter", "Unifier"];

Copyright