xCP2 Generate Numbers xCelerator Installation and User Guide

Version 2.0

Preface

Purpose

This document provides step-by-step instructions on how to install and configure the xCP2 Generate Numbers xCelerator.

Intended audience

The information in this guide is intended for process designers and developers using Documentum

Process Builder software. The objective of this xCelerator is to provide reusable java extensions to resolve common business problems.

Support information

The xCP2 Generate Numbers xCelerator is not supported by EMC.

Related documentation

The EMC Documentum xCelerator home page (https://community.emc.com/docs/DOC-6143) provides access to xCP xCelerators.

Revision history

Date	Version	Change
July 2010	1.0	Added the Generate Number activity template to the xCP Activity Templates xCelerator and simplified the installation procedure.
Oct 2012	2.0	Migration of this xCelerator to XCP 2.0

Contents

Overview

What is an xCelerator?

An xCelerator is one or more artifacts that can be used to accelerate the creation, adoption, and/or implementation of an EMC Documentum xCelerated Composition Platform (xCP) solution. An xCelerator is not necessarily a running application; instead, it is intended to hasten application development by providing key pieces of functionality out-of-the-box. Examples of this include:

- A sample business process flow created by Process Builder
- A sample high fidelity form using Forms Builder
- A sample Business Activity Monitor (BAM) dashboard
- A sample bit of code or configuration that connects the workflow with a third-party product
- Documentation that focuses on best practices and troubleshooting when developing and implementing an xCP solution
- A sample application

xCelerators are:

- Technology focused and can be samples, examples, starting points, or guides that accelerate the creation of an xCP solution
- Free and are not EMC-supported products, yet they do go through a formal release cycle, including quality assurance and documentation
- EMC intellectual property
- Sharable with multiple partners or anyone who wants to download them from the xCP community site

xCP2 Generate Numbers xCelerator

In Case Management solutions, it is frequently necessary to generate unique identities for the cases. These identifiers can, for instance, be used to route incoming correspondence to the correct case or grant.
The *Generate Number* java extension generates sequential numbers. It is flexible enough to handle multiple number sequences and a wide variety of formats.
Note: Depending on the specifics of your solution, especially in high volume multi-node scenarios, the Generate Number java extension might not be suitable for ID generation since it is implemented in a database agnostic way using a bounded retry strategy that relies on version stamps for uniqueness. In a high volume multi-node scenario, when the load on the number generator is sufficiently high, this retry strategy might run out of retry attempts, failing to generate the number, and thus causing the activity to fail. In this situation, the activity does not create a number.

Bill of Materials

The xCP2 Generate Numbers xCelerator is accessible through the Community xCelerator xChange home page (~~https://community.emc.com/docs/DOC-7597~~) by selecting the xCP Generate Numbers xCelerator download link.

This package contains:

Artifact	Path / File Name
Overview guide (this document)	xcelerator-generate_numbers.pdf
Installation files	Package/xcp2_generate_numbers.jar
Source Code	Source

Please note that you may need to create the folder **Java Modules** in **Artifacts** and the folder **modules** in **content**.

Installing xCP2 Generate Numbers xCelerator

Prerequisites

This excelerator requires a custom type **numgen_number_sequence** which must first be installed in the docbase.

This custom type is already provided in the xcp2_generate_numbers.jar.

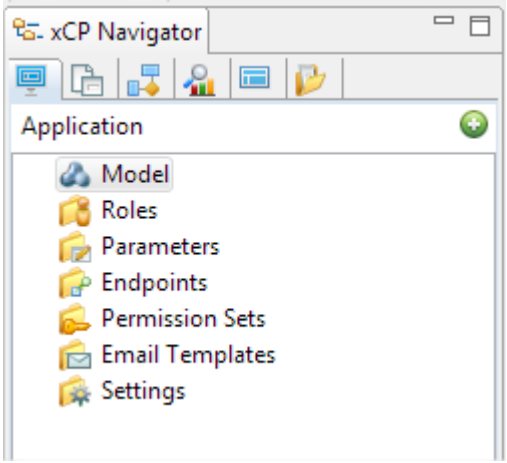
Install the xCP2 Generate Numbers xCelerator

To install the xCP2 Generate Numbers Excelerator Module:

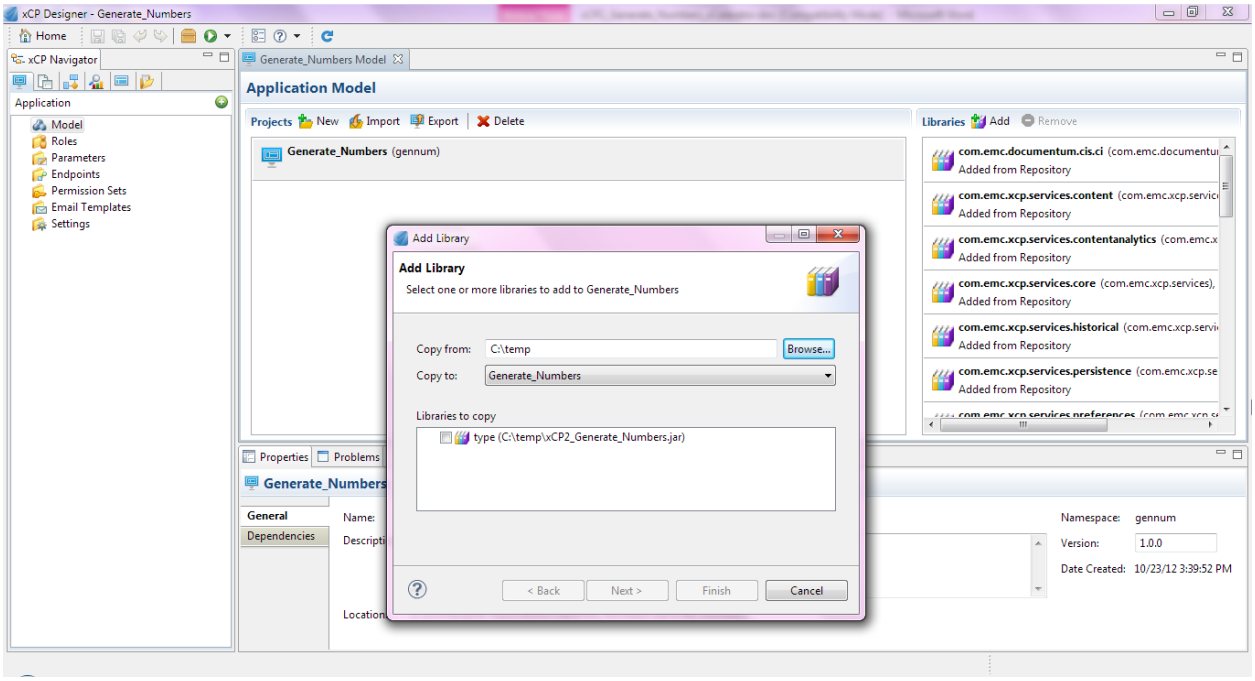
1. Download the xCP2_Generate_Numbers.zip file which is accessible through the EMC Documentum xCelerator home page (~~https://community.emc.com/docs/DOC-8174~~).

2. Unzip the file.

3. Go to the **Application** tab of xCP Designer and right click or double click on **Model** to open it.



4. On the right side, you can add the **xcp2_generate_numbers.jar** file in your project by clicking the **Add** button in the **Libraries** pane.



5. Select the folder where the jar file is located and click on Finish.
6. The custom type *gennum_number_sequence* will be copied as well as the *xcp2_generate_numbers.javamodule* and the 2 jars *xcp2_generate_numbers.jar* and *xcp2_generate_numbers_impl.jar* for the java service in your project.

Using the xCP2 Generate Numbers xCelerator

Creating a number sequence

Run the following DQL query, using a program such as IDQL32 or the DQL component in

Documentum Administrator (DA), to create the number sequence:

CREATE number_sequence OBJECT

SET object_name = ‘name of your sequence’,

SET current_value = 1,

SET increment_amount = 1;

Replace *name of your sequence* with a unique name that has meaning to your process, such as Court

Case Number Sequence or Grant Number Sequence.

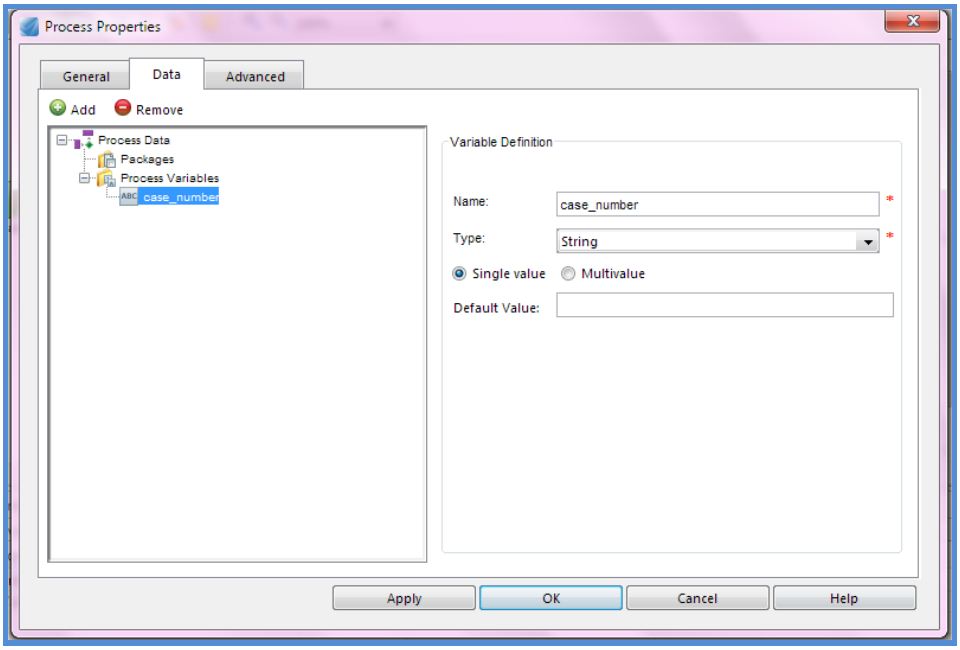
For example: SET object_name = ’Grant Number Sequence’,

The *current_value* is your starting number. It can be 0 (zero) or larger. Negative numbers in *current_value* are not supported.

The *increment_amount* is the amount by which the sequence number increases. It can be 1 or larger, but it is typically 1. Negative numbers in *increment_amount* are not supported.

Adding a process variable

Define a process variable to store the generated number. The variable must be of type String and the length must be sufficient to hold the entire generated number. For example, the following figure shows the addition of a process variable named *case_number* with a Type value of *STRING.*



Calling the java extension

This java extension provides you with 4 different methods.

- String generateNumber (String numberSequenceName)

Argument	Type	Desription
arg0	String	The name of the number sequence object.

This method generates a new case number in the specified number sequence. The number is not formatted in any way.

- String generateNumber (String numberSequenceName, String numberFormatPattern)

Argument	Type	Desription
arg0	String	The name of the number sequence object.
arg1	String	The format for the generated number. An empty format means that no formatting takes place.

This method generates a new case number in the specified sequence, formatting it according to the format (the second argument). For formatting details, refer to the chapter Case Number Formatting.

- String generateNumber (String numberSequence, String[] prefix, String[] suffix)

Argument	Type	Desription
arg0	String	The name of the number sequence object.
arg1	String[]	Any number of strings to prepend to the resulting case number. This is the case number prefix.
arg2	String[]	Any number of strings to append to the resulting case number. This is the case number suffix.

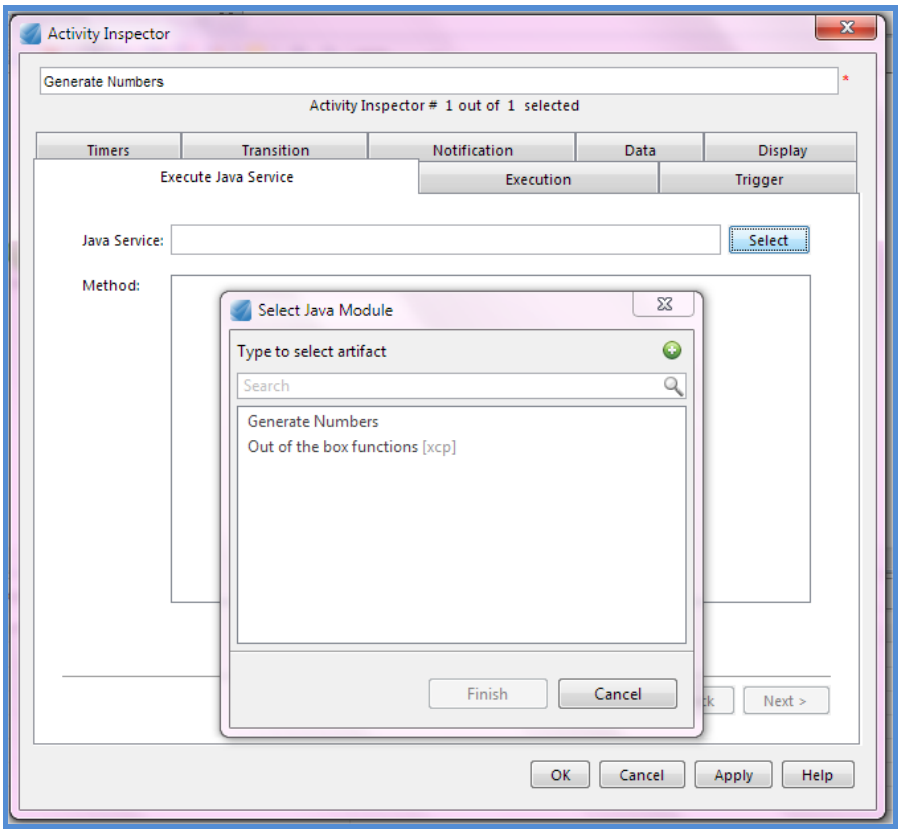
This method generates a new case number in the specified sequence, prepending any number of prefixes (second argument) and appending any number of suffixes (third argument).

- String generateNumber (String numberSequenceName, String numberFormatPattern, String[] prefix, String[] suffix)

Argument	Type	Desription
arg0	String	The name of the number sequence object.
arg1	String	The format for the generated number. An empty format means that no formatting takes place.
arg3	String[]	Any number of strings to prepend to the resulting case number. This is the case number prefix.
arg4	String[]	Any number of strings to append to the resulting case number. This is the case number suffix.

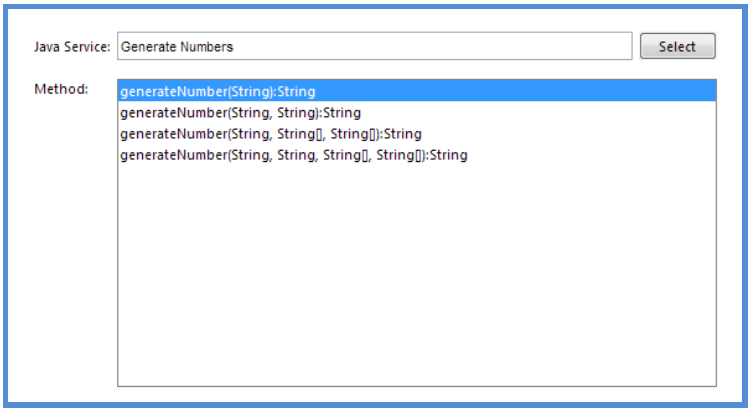
This method generates a new case number in the specified sequence, formatting it according to the format (the second argument), prepending any number of prefixes (third argument) and appending any number of suffixes (fourth argument).

Drag and drop an *Execute Java Service* activity to your process and name it **Generate Numbers**.



Select the Generate Numbers java module and click Finish.

The methods present in the java module will be displayed.



Case Number Formatting

Case number formatting is defined as an argument, numberFormatPattern to two of the four methods of this xCelerator. The format pattern is specified by:

• A # (hash) character is replaced by the digit in that position or nothing if no such digit exists.

• A 0 (zero) character is replaced by the digit in that position or a 0 if no such digit exists.

• A \ (backslash) character is used to escape any of the above characters, so \# will yield #.

• A ? (question mark) character followed by any character is equivalent to 0 but instead of producing

a 0, the second character is produced. That is, ?0 is the same as 0.

• Any other character is replaced with itself, unless it occurs after a # character and before a # character, which would be replaced with nothing.

Case number formatting examples

Number	Pattern	Result
1	###-###-#	1
1	000-000-0	000-000-1
12	###-###-#	1-2
12	000-000-0	000-001-2
1234	###-###-#	123-4
1234	000-000-0	000-123-4
1234	The number is ###-###-#	The number is 123-4
1234	The number is 000-000-0	The number is 000-123-4

Note: Include enough digits to cover the entire range of case numbers that could be created to avoid a case number format overflow exception. For example, a pattern such as "##" cannot handle a number over 99 and it fails if that happens. To prevent this, either include enough digit positions in your format or implement a number sequence / series retirement procedure.

Sample case numbers

All of the following examples assume a number sequence called *Sample Number Sequence*. This sample sequence must first be created, and the current value is 5.

Case Number	Configuration
5	Method: String generateNumber (String numberSequenceName) arg0: "Sample Number Sequence"
0005	Method: String generateNumber(String numberSequenceName, String numberFormatPattern) arg0: "Sample Number Sequence" arg1: "0000"
AB000005	Method: String generateNumber(String numberSequenceName, String numberFormatPattern) arg0: "Sample Number Sequence" arg1: "AB000000"
US-0000-05	Method: String generateNumber(String numberSequenceName, String numberFormatPattern,String[] prefix,String[] suffix) arg0: "Sample Number Sequence" arg1: "-0000-00"

	arg2: country_code from SDT arg3: “”
2010/SE/0000/05	Method: String generateNumber(String numberSequenceName, String numberFormatPattern, String[] prefix,String[] suffix) arg0: “Sample Number Sequence” arg1: “/0000/00” arg2: year from sdt,”/”,country code from SDT arg3: “”
Sample Number Sequence – 5	Method: String generateNumber(String numberSequenceName, String numberFormatPattern, String[] prefix, String[] suffix) arg0: “Sample Number Sequence” arg1: “ - ######” arg2: “Sample Number Sequence” arg3: “”

Example

In this example, we add an Execute Java Service activity to a process and use it to generate Grant Request identifiers with the following formatting:

GR-000-001
GR-000-002
GR-000-003
…
GR-000-999

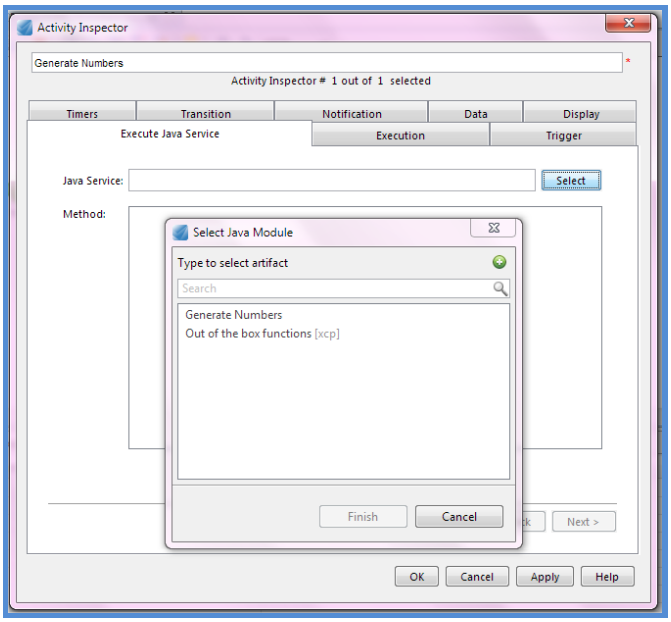
GR-001-000

1. Create a number sequence called Grant Request Number Sequence by executing the following

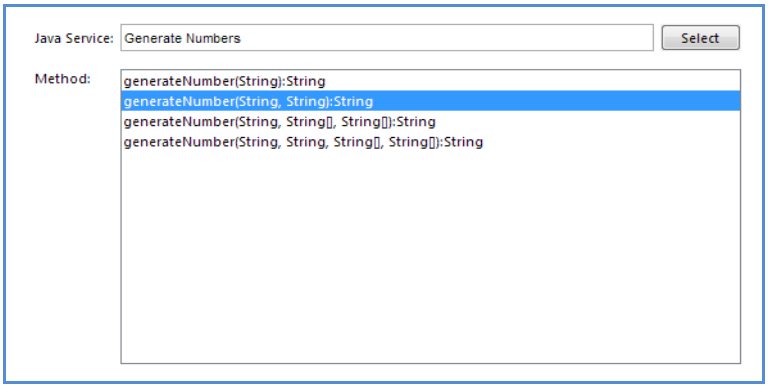
   DQL in DA or IDQL32:

```
CREATE number_sequence OBJECT

SET object_name = 'Grant Request Number Sequence',

SET current_value = 1,

SET increment_amount = 1;
```

2. Define a process variable to store the sequence number return value.
3. Add an *Execute Java Service activity* to your process.
4. Open the Activity Inspector for this activity.
5. Select the Generate Numbers java module



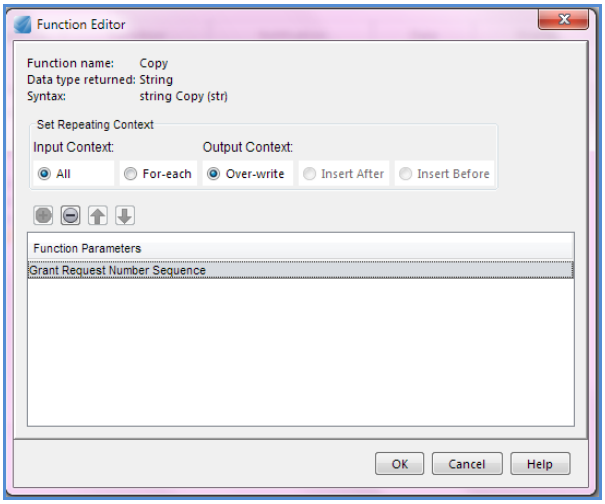
6. Choose the second method generateNumber (String numberSequenceName, String numberFormatPattern).

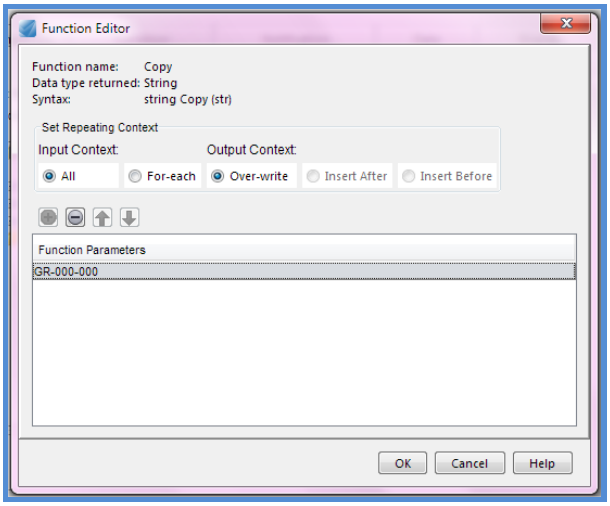


7. Click next to get the Input Message Mapping window and configure it as follows:

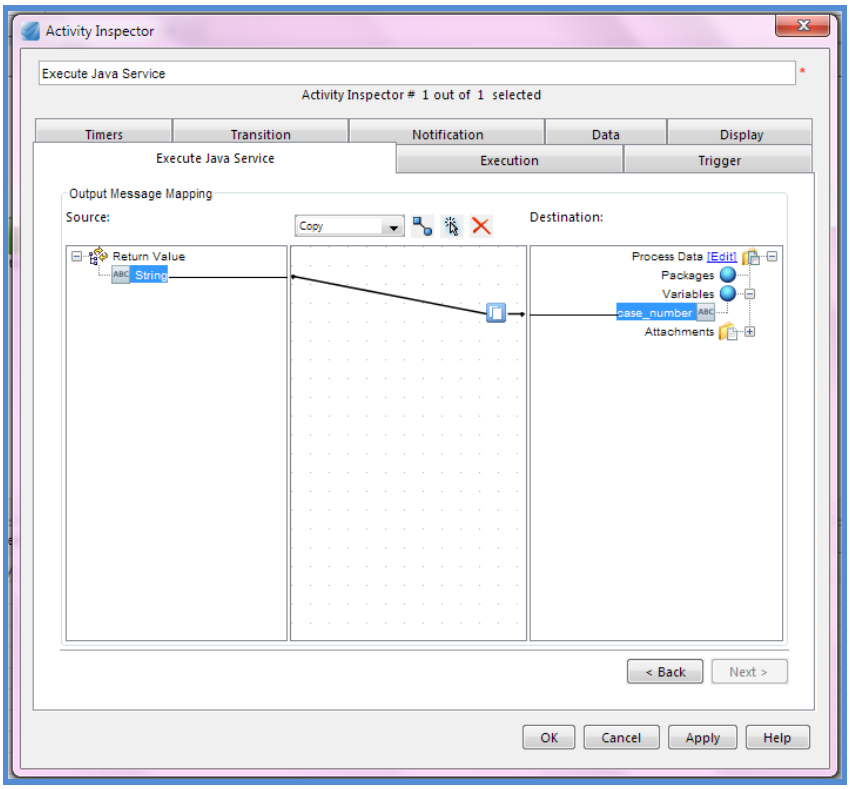
- arg0: Grant Request Number Sequence
- arg1: GR-000-000

To enter a value for an argument, double-click the argument on the Input Configuration screen to access the Function Editor for that argument. Click the + button to add a function parameter line, enter your parameter value, and then click OK.





8. Click Next.
9. Configure the Output Message Mapping window. Map the formatted number to your process variable.



10. Click Ok to finish the configuration of this activity.

Considerations

Since regular ACL-based security applies to the Generate Number activity, restrict the ACL on the number sequence objects to only the user that executes the activity. The current value of a number sequence is stored as an Integer in the database, effectively limiting the quantity of numbers you can generate to 2147483647 different numbers. You should be aware of this and, if necessary, define a retirement procedure for the number sequence well in advance of exhausting the sequence.

Troubleshooting

If your *Execute Java Service* activity is not working as expected, check your input message mapping:
• Repeating arguments need at least one value for each argument. Values can be blank or empty. Examples: prefix and suffix.
• Provide at least an empty value for each argument.
To add an empty value to an argument, open the Function Editor for the argument, click the + button to add a function parameter line, and leave it blank.
Click **OK**.