



Course 13
Broadway: Beyond Actors

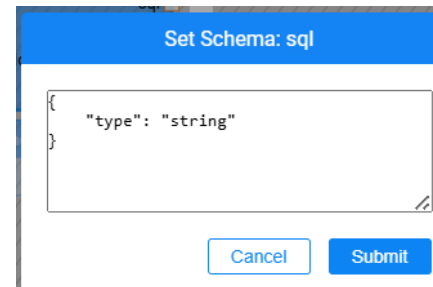
Agenda

- Supported Data Types in Broadway
- Link Actors
- Error handling
- Reset Actor State
- Build a Custom Actor
 - Schema Property
 - Editor Property
- Parallel Stage Execution
- Broadway metrics
- Recovery Point
- Deploy.flow



Supported Data Types in Broadway

- Actors pass Java objects between stages. Broadway supports a focused set of types and will **auto-convert** between them when an Actor expects a different (compatible) type.
- Supported types are:
 - Described by the parameter's **Schema** property

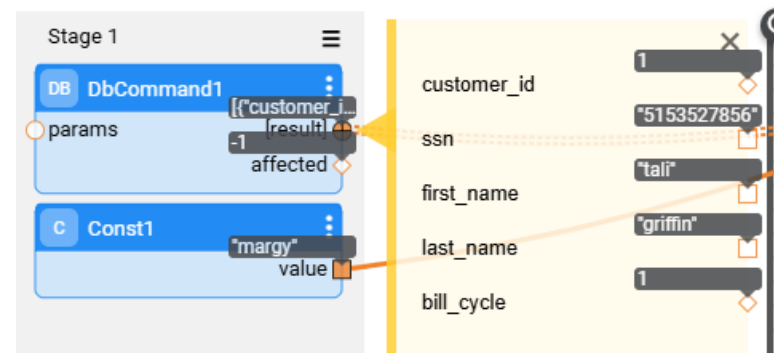


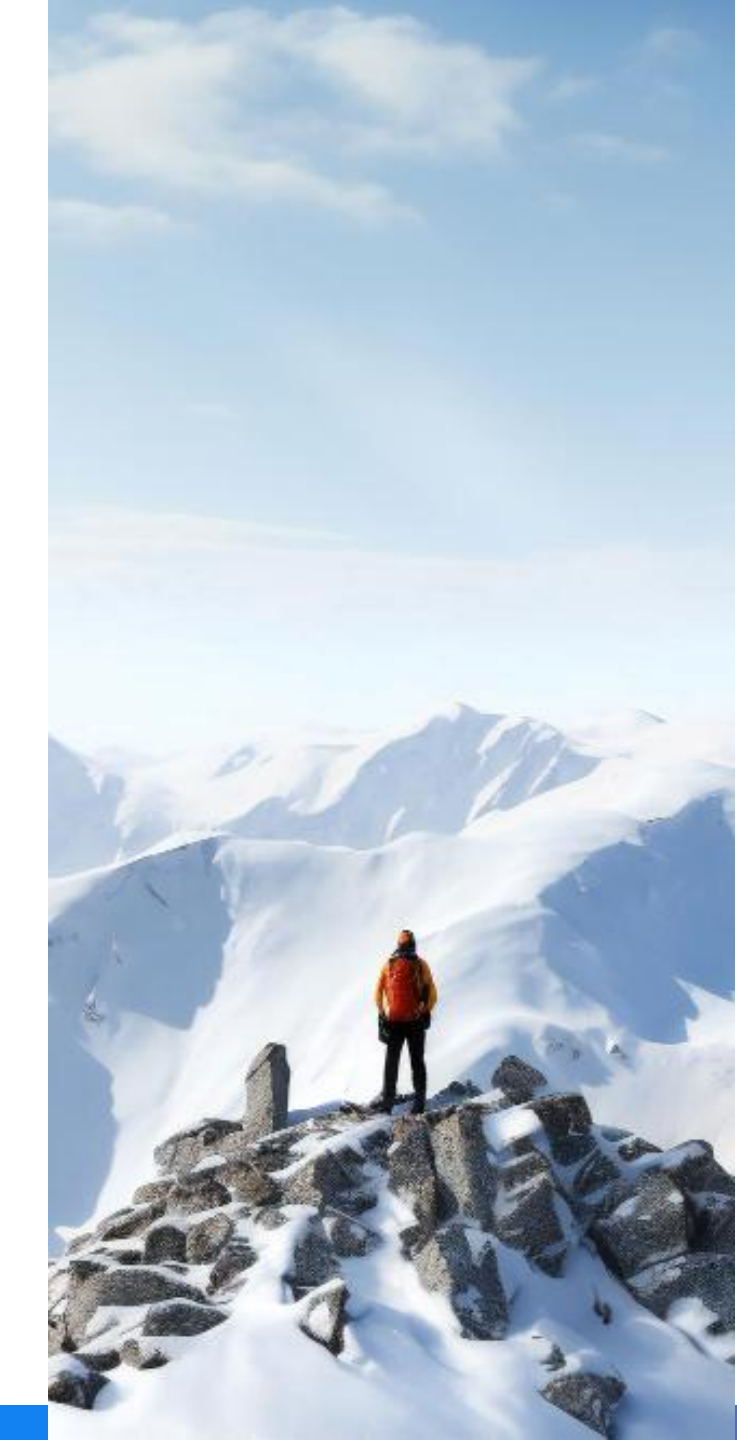
A dialog box titled "Set Schema: sql" with a text area containing a JSON schema:

```
{  "type": "string"}
```

. Below the text area are "Cancel" and "Submit" buttons.

- Visualized in the **Data Inspector**

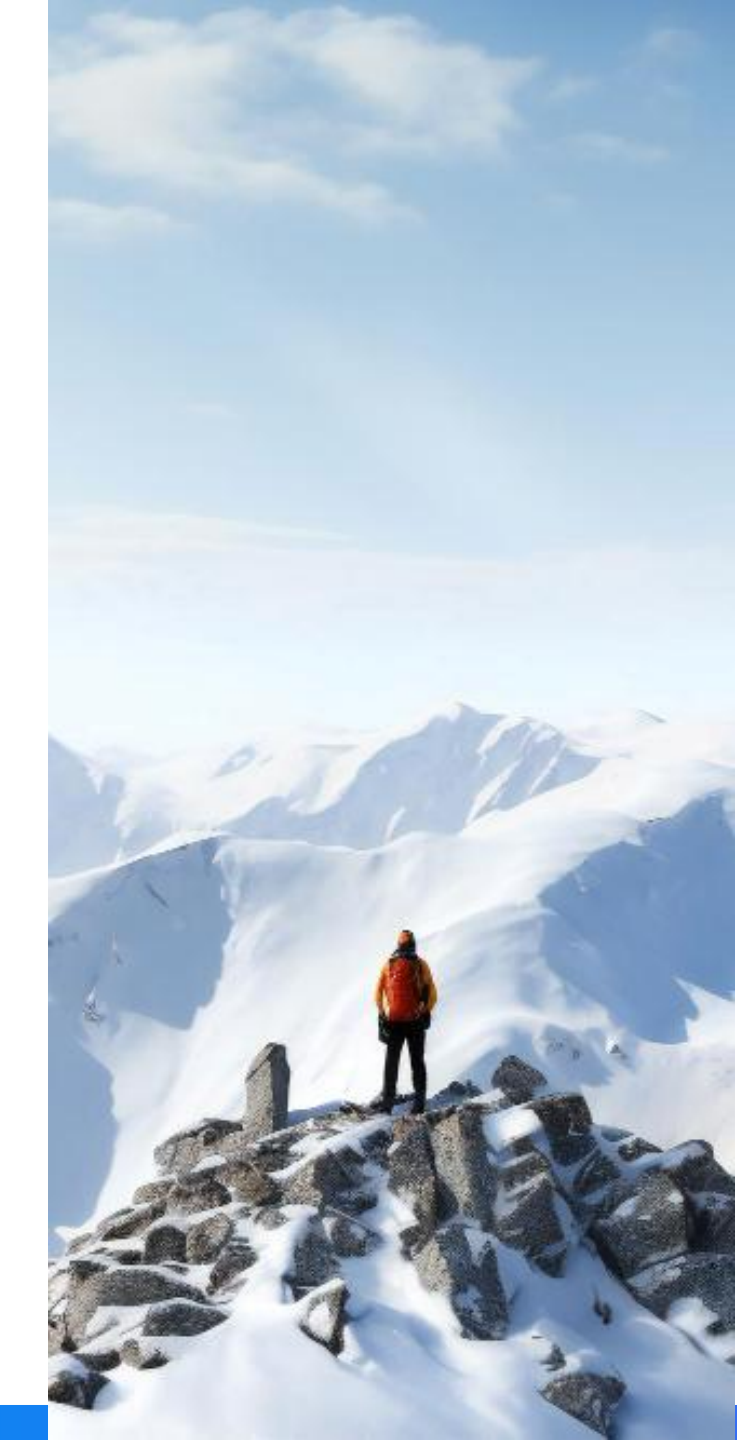




Supported Data Types in Broadway

Core supported types

- **Primitives:** String, Long, Real, Boolean, Date, byte[]
 - **Other numerics:** int/Integer, short/Short, byte/Byte, float/Float auto-convert to **Long/Double**
 - **Binary:** JDBC **Blob** → auto-converted to **byte[]**
- **Collections (Iterables):** supported through Java arrays or maps



Supported Data Types in Broadway

Implicit conversion

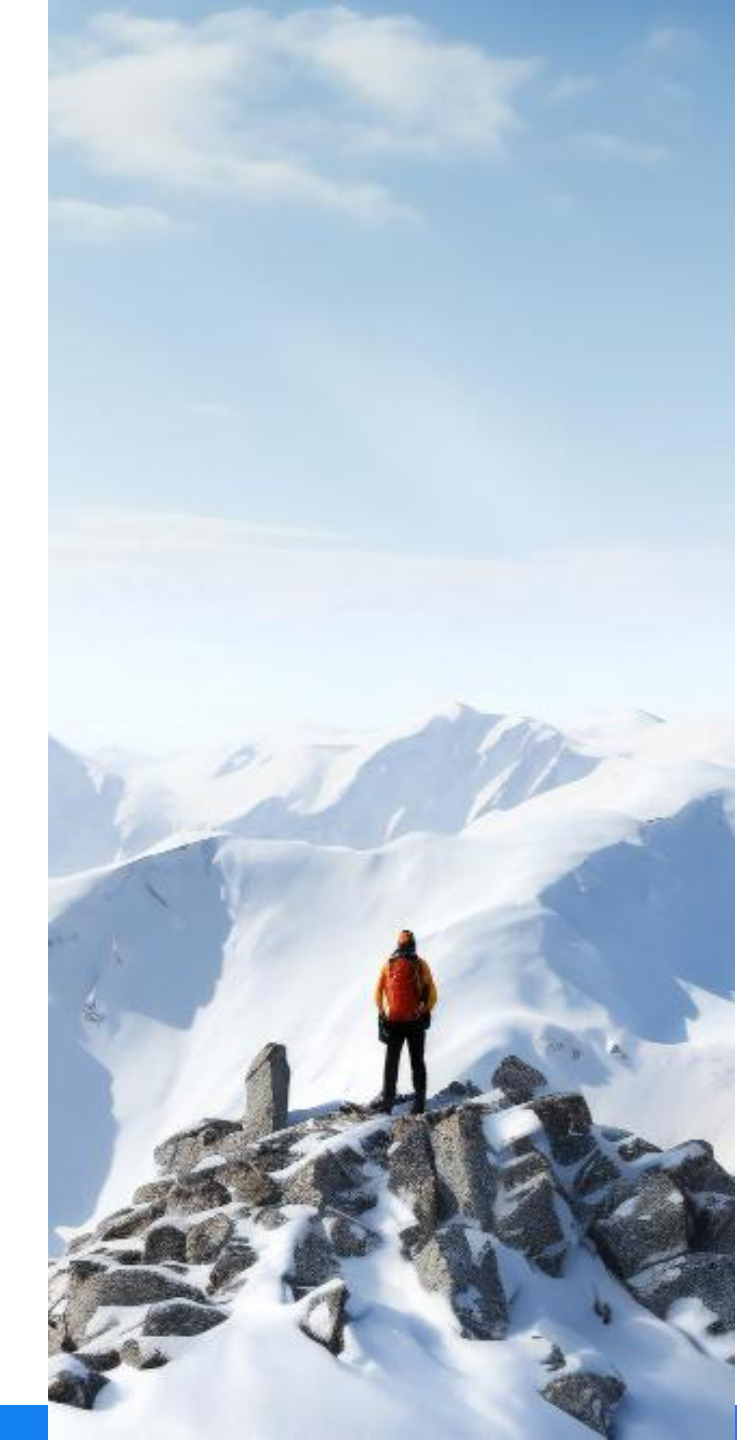
Broadway converts inputs to the type an Actor declares. “Reasonable” conversions are automatic; otherwise an exception is thrown.

Handy rules

- **Numbers ↔ Strings:** Long/Double $\rightleftharpoons$ String (parse/format)
- **Booleans ↔ Numbers:** true $\leftrightarrow$ 1, false $\leftrightarrow$ 0
- **Booleans ↔ Strings:** "true"/"false"; empty/"0"/"false" $\rightarrow$ false
- **Numbers ↔ Date:** numbers are **milliseconds since 1970-01-01 00:00:00 UTC**
- **Value $\rightarrow$ Iterable:** becomes a **single-item** iterable (7 $\rightarrow$ [7])
- **Strings ↔ byte[]:** via **UTF-8**; other types format to String first, then to UTF-8 bytes

Practical examples

- Read "0" into a Boolean input  $\rightarrow$ false
- Pass 7 to an Iterable input  $\rightarrow$ becomes [7]
- Convert byte[] to String  $\rightarrow$ UTF-8 assumed
- Try null into a Date  $\rightarrow$ not supported



Supported Data Types in Broadway

Null behavior

null is supported and **implicitly convertible** to every type with safe defaults:

String="", Long=0, Double=0.0, Boolean=false, Date=1970-01-01 00:00:00, byte[]=empty, Iterable=empty, Map=empty

Date formats

- **String output (format):** yyyy-MM-dd HH:mm:ss.SSS (UTC)
- **String input (parsing):**
 - yyyy-MM-ddTHH:mm[:ss[.SSS[Z]]]
 - yyyy-MM-dd HH:mm[:ss[.SSS[Z]]]
 - Delimiter T or space; seconds, millis, and timezone are optional

Broadway supports date/time manipulation Actors for more explicit date/time conversions and calculations.

Linking Actors

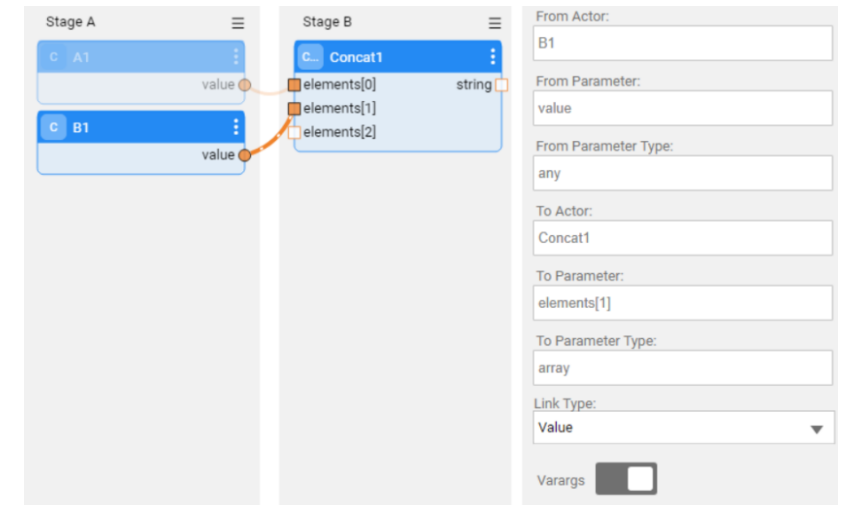
Link type

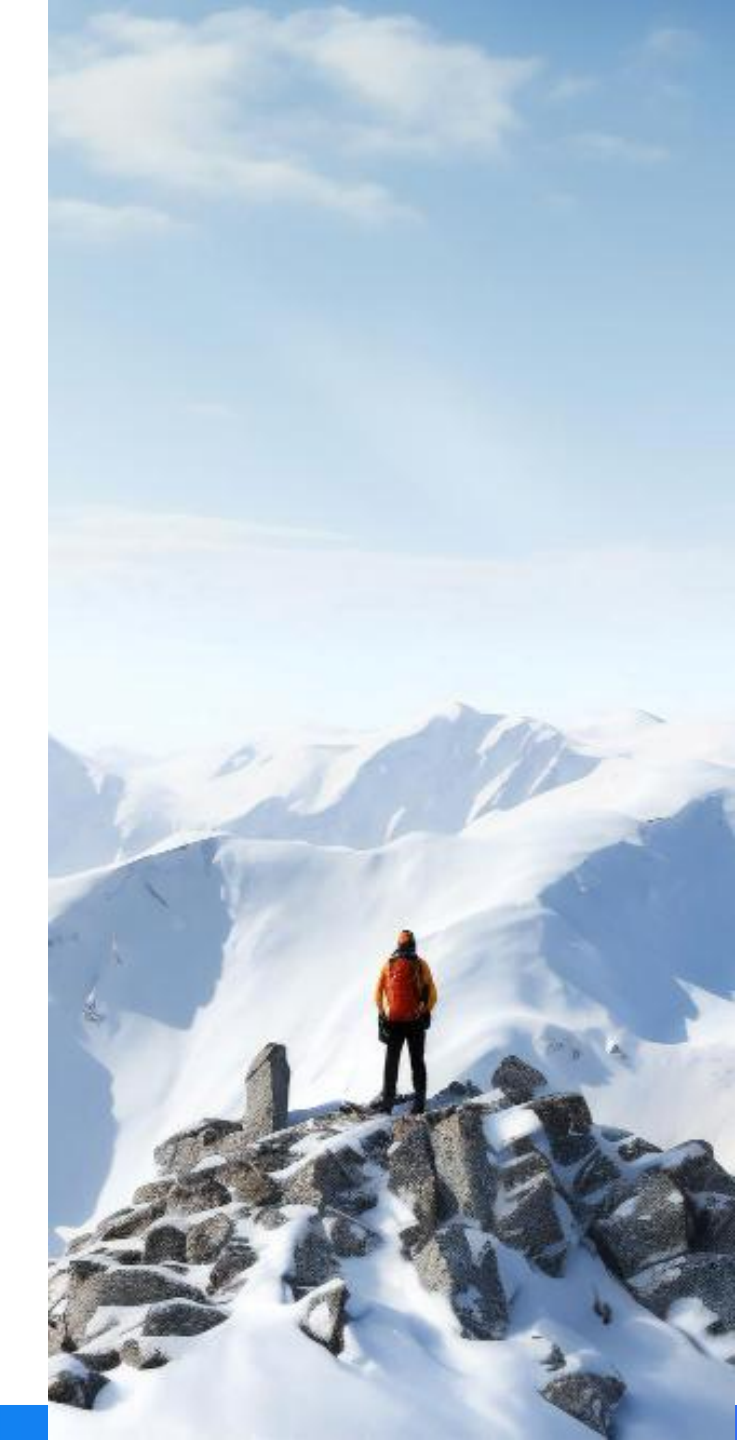
- **Value** (*default*) - Pass the value as-is.
- **Iterate** - Opens a loop over the linked value (Iterable/array).
The link appears as a **double-dashed** line.
Auto-behavior: if you connect an **array of T** to an input of type **T** (e.g., `string[]` → `string`), Broadway will automatically set the link to **Iterate**.
- **First**
Pass only the **first** item (e.g., the first record of a result set).

Varargs (*variable arguments*)

When **ON**, the target input becomes an **array**, and each additional source link appends another element to that array.

Useful for building arrays on the fly (e.g., feeding the **Concat** Actor with many strings).



A vertical photograph on the left side of the slide shows a person wearing a red jacket and dark pants standing on a rocky, snow-covered mountain peak. The person is looking out over a vast, snow-covered mountain range under a clear blue sky with some light clouds.

Error Handling

Broadway handles exceptions at the **Stage** level, similar to a Java try–catch. If any **Actor** in a Stage throws an exception, the Stage's error handler runs and decides the path forward:

- **return true** → suppress the error and continue the flow
- **return false** → do not suppress; stop the flow

What can act as an error handler?

- **Any Actor that return true or false.** Including:
 - JavaScript Actor with custom logic in its input parameters
 - An InnerFlow Actor
- Two built-in handlers designed for this purpose:
 - **ErrorHandler**
 - **ErrorFields**

For reuse across multiple flows or Stages, prefer implementing error handling as an **Inner Flow** and referencing it where needed.

Error Handling

ErrorHandler Actor

Catches exceptions raised by any Actor in its Stage and provides **Per-exception handling rules**.

- **Exception classification:**

- **SQL** (split into *Unique constraint* vs. *Other*).
Unique constraint detection is supported for Oracle, DB2, SQLite and SQL Server.
- **HTTP** - option to provide a specific status code
- General **Exception** - option to provide a specific error message regex

- **Suppression control**

- Suppress checkbox toggles whether to swallow the error.
- If an Inner Flow is configured for that error, it can override the checkbox by returning:
 - *true* → *suppress and continue*
 - *false* → *do not suppress; stop*

The screenshot displays the configuration for an 'ErrorHandler1 : ErrorHandler' actor. The left pane shows 'Stage 1' with a red 'ErrorHandler1' block and two blue 'GET FabricGet' blocks. The right pane shows the configuration for 'ErrorHandler1 : ErrorHandler'. It includes a dropdown for 'All Fields', an 'Inputs' section with 'error : any', and a 'config : any' section. The 'Exception' dropdown is set to 'was not found and sync is', with 'Suppress' and 'Log' checkboxes checked. The 'Log' checkbox is checked, and the 'Log' field is set to 'bwlInnerFlowErrorHandler'. The 'Interval MS' field is set to '1'. The 'Inner Flow' dropdown is set to 'Inner Flow', and the 'Interval MS' field is set to '500'. The 'Sql Error' and 'Http Error' options are also visible in the dropdown menu.

ErrorHandler Actor

- **Built-in Retry**
 - Configure **Retries** and **Interval (ms)** in the ErrorHandler editor (defaults: Retries=0, Interval=500ms).
 - If an **Inner Flow** is used as the handler, it may override behavior by returning **result = true / false / retry**
 - * “result” external name must be used.
- **Logging:**
 - Optional “Log” checkbox writes the error to logs
 - If retries are enabled, the current retry attempt is also logged (e.g., “Stage <name> – retrying <Actor>: attempt <n>”)

The screenshot displays the K2view ErrorHandler editor. On the left, a stage diagram shows 'Stage 1' with an 'ErrorHandler1' actor (red box) and two 'GET FabricGet' actors (blue boxes). The 'ErrorHandler1' actor has an 'error' input and a 'result' output. Below it, the 'GET FabricGet 500' actor has an '[iid]' input and a '[result]' output. The 'GET FabricGet 1500' actor also has an '[iid]' input and a '[result]' output. On the right, the 'ErrorHandler1 : ErrorHandler' configuration panel is shown. It includes a dropdown for 'All Fields', an 'Inputs' section with 'error : any', and a 'config : any' section. The 'config : any' section has an 'Exception' dropdown set to 'was not found and sync is', with 'Suppress' and 'Log' checkboxes checked. Below these are input fields for '1' and '500'. A 'Log' checkbox is also present. At the bottom, a dropdown menu is open, showing options: 'Sql Error', 'Http Error', and 'Exception'. The 'Log' checkbox is checked, and the 'Interval MS' field is set to '500'.

Error Handling

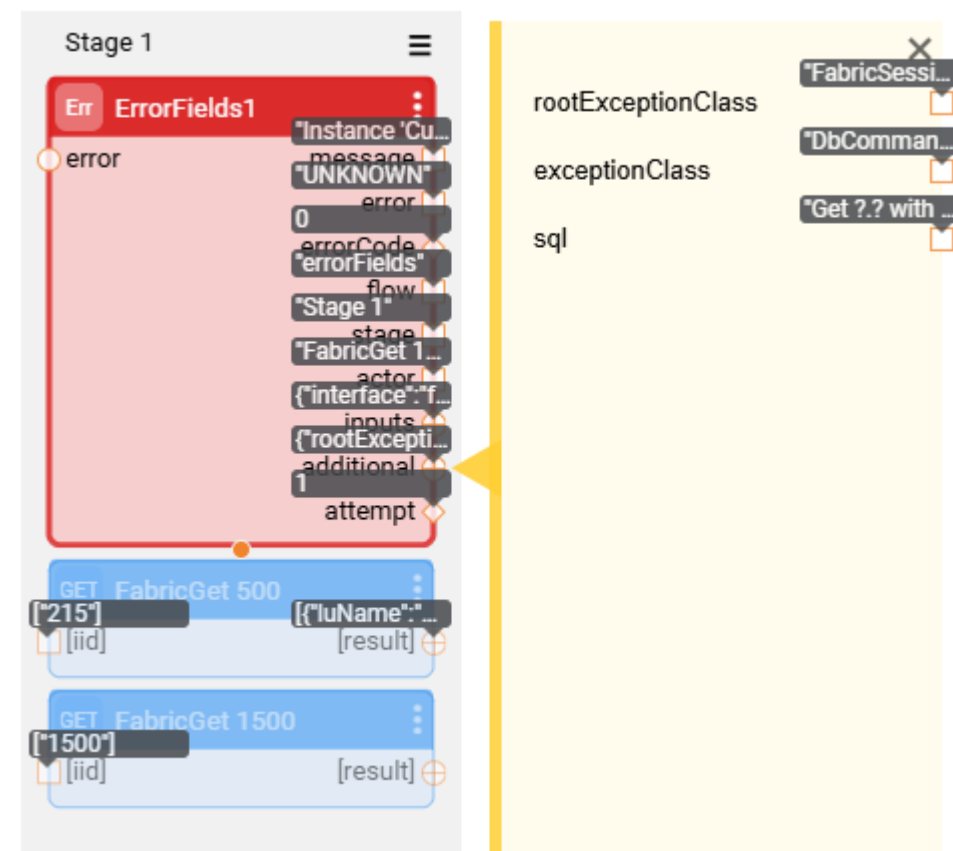
ErrorFields Actor

Always returns true (i.e., it suppresses the error), so the flow continues. It exposes structured context you can use to make routing decisions:

- **Error message**
- **Error code**
- **Origin:** flow, stage, and actor names
- **Actor inputs** that triggered the exception
- **Additional info:** exception class, SQL statement (when relevant), and **stack trace** (if requested)
- **Number of retry attempts**

Usage patterns

- **As the Stage's error handler** (directly)
- **Inside an Inner Flow** that serves as the Stage's error handler
 - The Inner Flow receives the error as input, runs **ErrorFields** to unpack it, and then applies custom logic to decide next steps.
 - The error input parameter of the ErrorFields must be set as External.



Error Handling

Retry Mechanism

When a Stage throws an error, BW can automatically retry the **failing Actor**.

Ways to drive retries:

- **ErrorHandler settings**

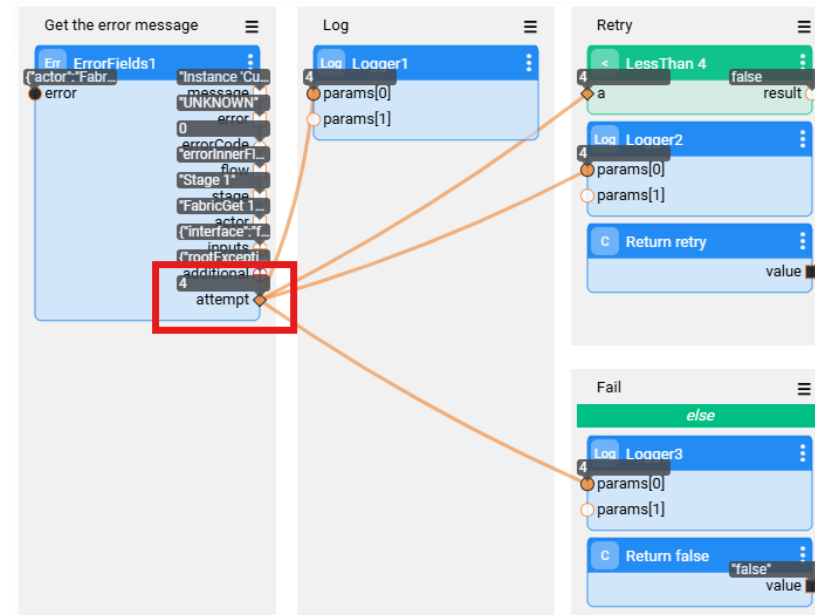
Set **Retries** and **Interval** in the ErrorHandler.

If using an **Inner Flow** as the handler, it can return **result = retry** to continue attempts

Note: The output parameter must have “result” external name

- Any actor defined as an error handler in a flow ('red actor').
Have the handler return **retry** (instead of true/false) to trigger another attempt.

If the error handler implemented using an **Inner Flow**, you can read the current attempt count via **ErrorFields.attempt** and branch logic accordingly.



Reset Actor State

Broadway allows resetting an Actor's state during flow execution. This is particularly useful when handling **nested loops**.

For example, when using a **StringBuild Actor** to aggregate values inside an inner loop, the aggregation must be cleared at the start of each iteration of that loop.

How to Reset Actor State

- **Context Menu Reset (per iteration)**
 - Right-click the Actor → **Reset on iteration 0**.
 - This option is available only when the Actor is placed inside an internal iteration (level ≥ 2).
 - Once enabled, a reset icon appears on the Actor badge.
 - At runtime, the Actor's state is reset at the beginning of iteration 0, just before execution.
 - Applies only to Actors that maintain internal state.
- **ResetActors Actor**
 - Use the **ResetActors** Actor to programmatically reset multiple Actors by ID (Actor names).
 - The specified Actors have their state cleared before continuing execution.

Transactions in K2View Broadway

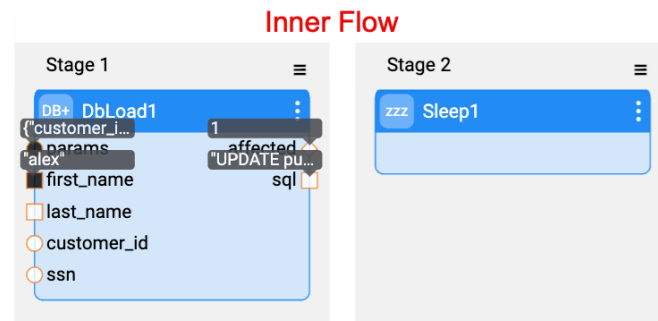
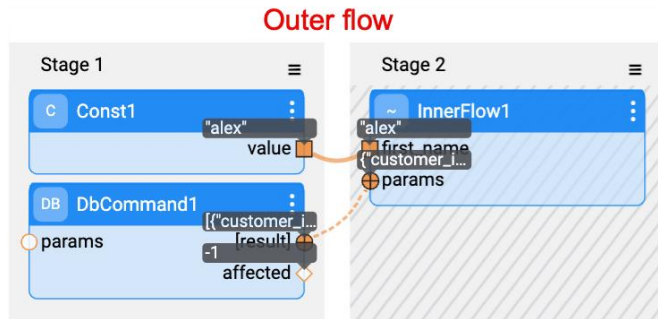
Broadway Transactions Handling

- Transactions ensure consistency and isolation of data changes.
- Transaction starts when a transaction-marked Actor initiates a connection.
- Sequential transaction-marked Stages share the same transaction.
- Transaction ends at the last transaction Stage or flow completion.
- Followed by commit or rollback based on flow outcome: if error occurs in the transaction stages – rollback will be initiated. Otherwise – commit.

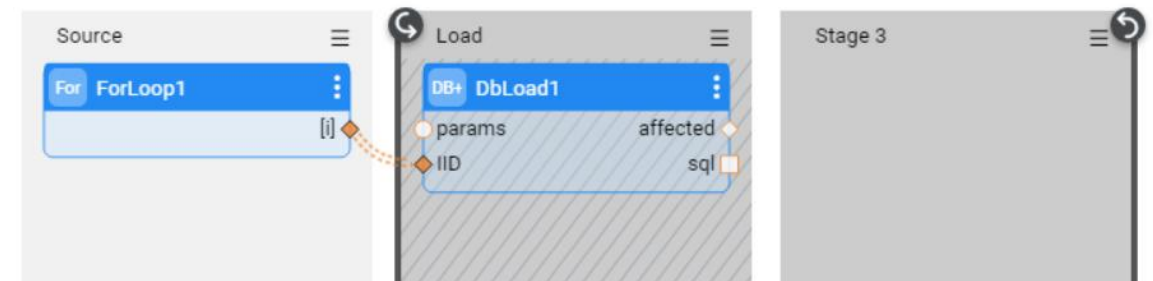
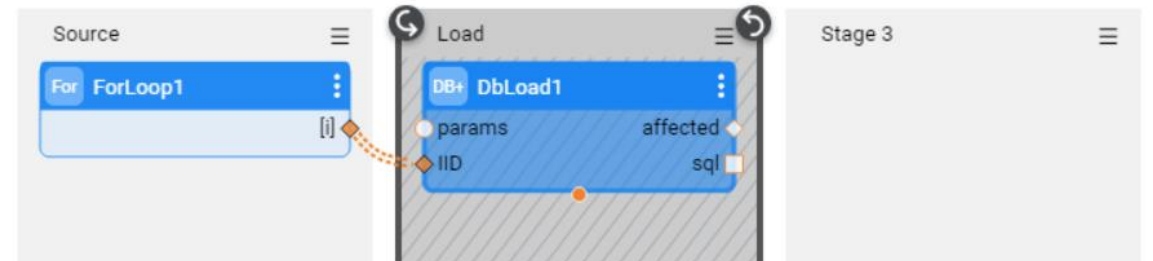


Transaction Scope: Inner Flows & Iterations

- Transactions in Inner Flow
 - Inner flows inherit the transaction of the calling (outer) flow.
 - They use the same shared resources but do **not** close the transaction.
 - The transaction is always closed by the **outer flow**.



- Transactions in Iterations
 - If **all Stages** in an iteration are transactional, a **single** transaction continues and commits after the full iteration.
 - If **non-transactional** Stages exist, each iteration starts and commits a **separate transaction**.



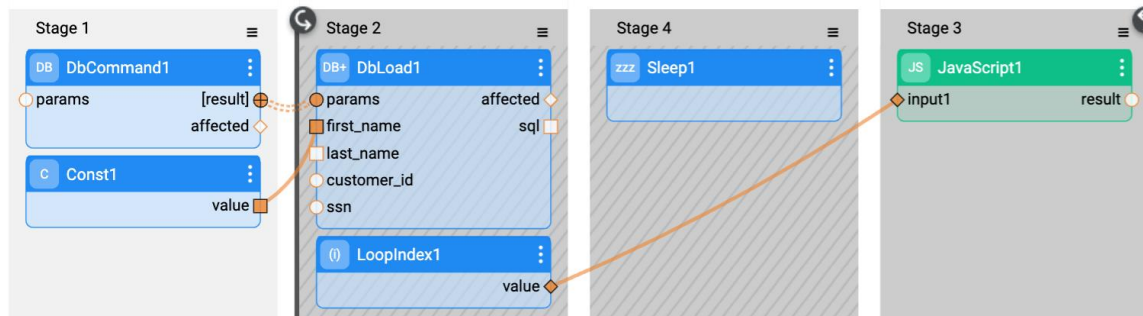
Transaction Scope: Various Iteration Examples

Committing transactions in large data sets

- For a large data sets, you can commit every X records using a Stage Condition
- Example: to commit every five records using JavaScript actor as a Stage Condition:

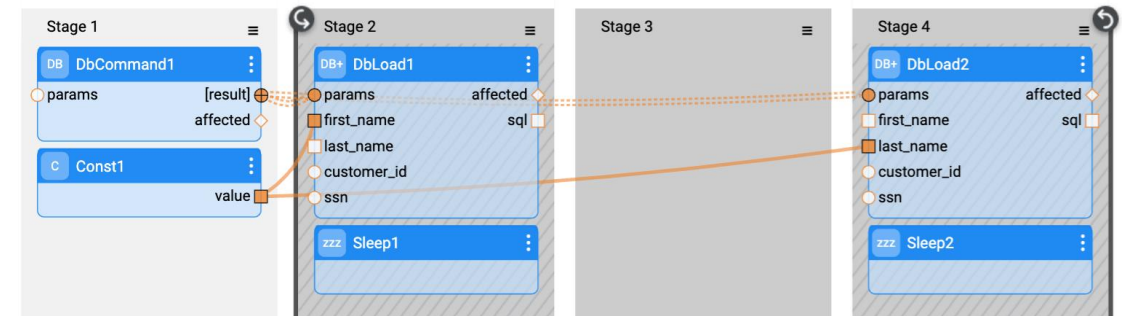
```
var i = input1;
(i + 1) % 5 == 0;
```

- The Check Stage executes only every 5th records.
- Since it's **non-transactional**, a **commit occurs**, and a **new transaction** starts for the next batch.



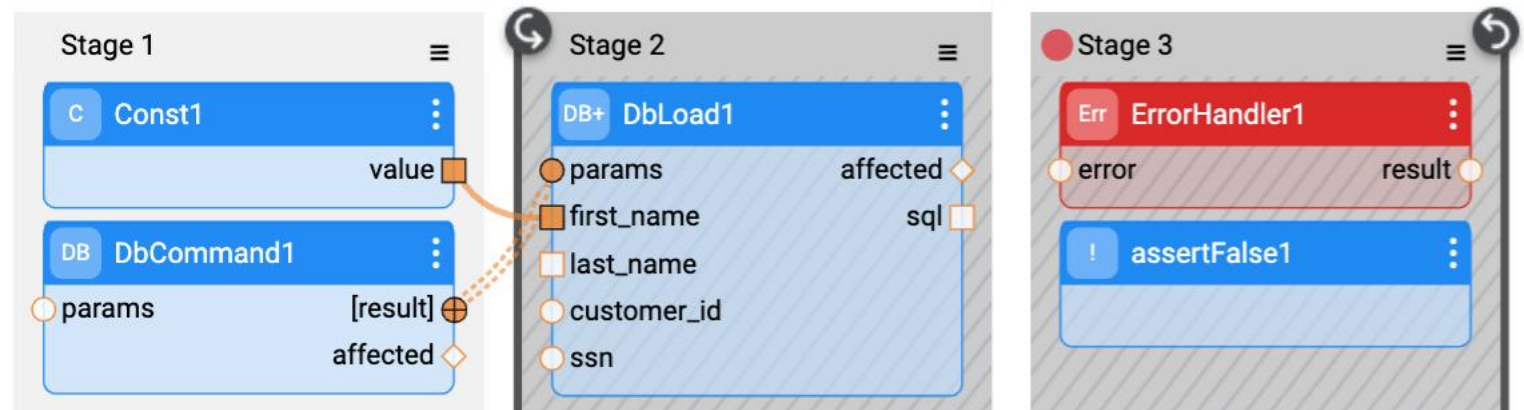
Transactions in a Loop with Mixed Stages

- When a loop contains both transactional and non-transactional Stages, the transaction scope resets based on Stage configuration.
- A transaction starts at the first transactional Stage and commits when a non-transactional Stage is reached.
- A new transaction begins with the next transactional Stage.
- This cycle continues through the loop, committing and restarting transactions as defined by the Stage types.
- The final transaction is committed at the end of the loop if not already committed earlier.



Error Handling for Transactions

- When an Error Handler is attached to a transactional Stage, it controls flow continuation
- If it returns **false** (do not suppress), the transaction is **rolled back** and the flow **stops** with an error message
- If it returns **true** (suppress), the flow **continues**, the transaction **remains active** and commits when a non-transactional Stage is reached.



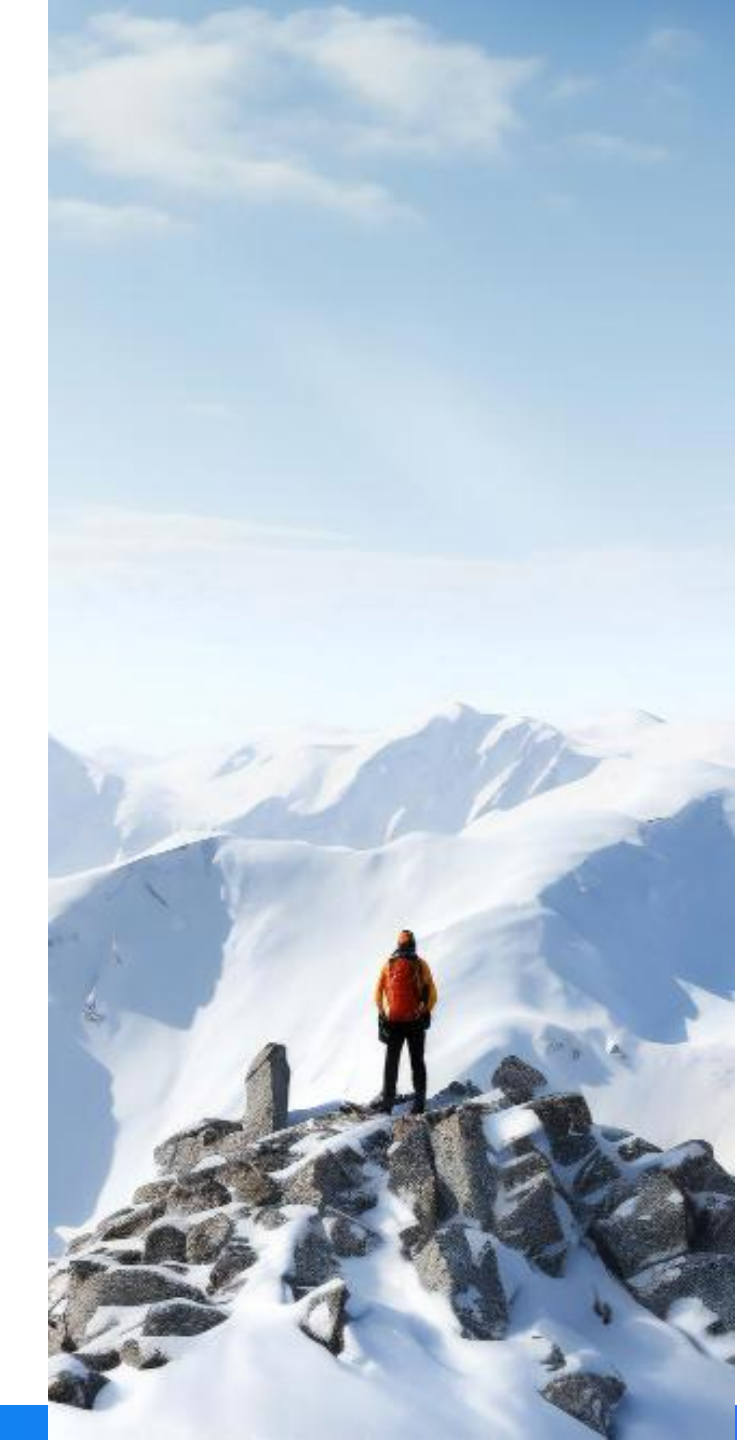
NoTx Actor

- The **NoTx Actor** marks an interface as **non-transactional** within an active transaction context.
- It prevents the execution of **begin**, **commit**, and **rollback** commands on that interface.
- It must be placed before any transactional operation is performed on the interface.
- An exception will be thrown if used outside of a transaction or if the interface is already participating in a transaction.

Use Case: Writing to a Logging Table in a Separate Database

- Your primary transaction runs on Database A, but you log activity to Database B.
- To avoid cross-DB transaction issues or distributed transaction overhead, mark Database B as NoTx.
- This ensures logs are saved even if the main transaction on Database A is rolled back.





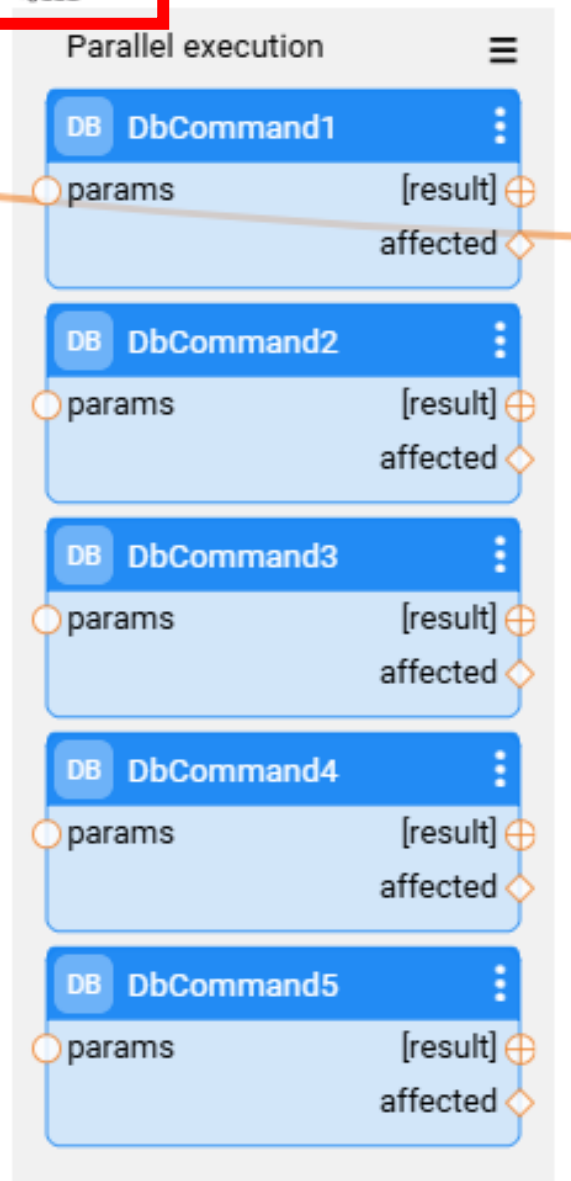
Transaction Best Practice

Broadway includes a built-in transaction management mechanism. For iterative processing, transactions can be committed in three ways:

- **Per Iteration** – commit after each loop.
- **End of Iteration** – commit once after all data is processed.
- **Batch Commit** – commit every X records.

The choice depends on data volume, performance, and business requirements. For example:

- Large data sets (e.g., 1M records) → use **batch commit** for efficiency.
- Full rollback required on failure → use **end of iteration commit**.



Stage Actors Parallel Execution

A Stage can run multiple Actors (or Inner Flows) in parallel to increase throughput.

Configure parallelism

- Open the Stage menu (icon at the stage's top-right).
- Select Parallel and enter the number of threads.
- After saving, a Parallel icon with the thread count appears above the Stage name.

Disable parallelism

- Open Stage menu → Parallel, set the thread count to 0.

Build a Custom Actor

Broadway provides two options for creating a custom Actor:

- **Create a new Actor**
 - Based on Java code, **or**
 - Based on a BW Flow
- **Inherit from an existing Actor**



Create a New Actor

New Actor Based on Java code

1. Create the Java code

- In the project, create a new Java file (src → right click → New Java File).
- Extend the Actor class.
- Implement the action method. This method is executed when the Actor runs.
- Use the input and output maps to handle parameters:
 - Retrieve input: `input.string("inputName")`
 - Set output: `output.put("outputName", value)`

Example:

```
public class MathSubtract implements com.k2view.broadway.model.Actor {  
    public void action(Data input, Data output) throws Exception {  
        Integer a = Integer.parseInt(input.string("a"));  
        Integer b = Integer.parseInt(input.string("b"));  
        output.put("result", Math.subtractExact(a, b));  
    }  
}
```

2. Create the new Actor in Broadway, to execute the Java code

Broadway → Right click → **New Actor**.

- Provide the Actor name
- Set **Tag** (category in the palette).
- Set **Badge** (icon next to Actor name).
- Set the java path to your Java **Class** (Java class path).

3. Define input/output parameters with types and editors.

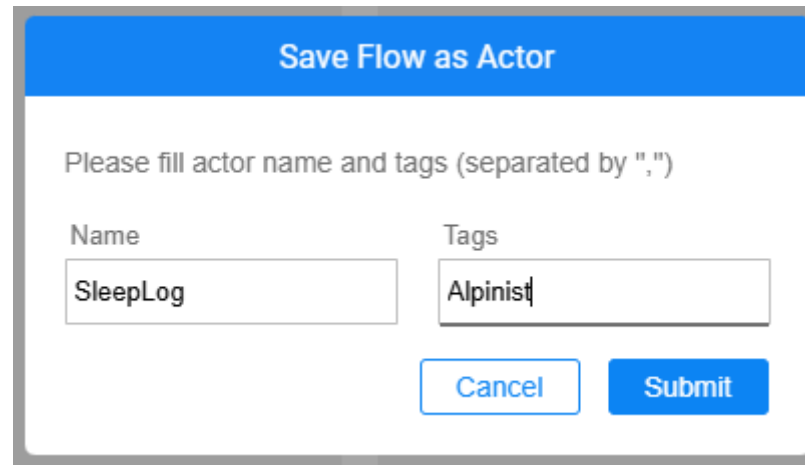
The screenshot shows the Broadway IDE interface. On the left, a 'Stage' view displays a blue actor box labeled 'MathSubtract'. It has two input ports on the left labeled 'a' and 'b', and one output port on the right labeled 'result'. On the right, a configuration panel is open. It has a tab labeled 'All Fields' and a section titled 'Actor'. Below this, there are fields for 'Tags' (containing 'math'), 'Badge' (containing '-'), 'Add Inputs' (containing 'false'), 'Add Outputs' (containing 'false'), 'Parent' (empty), and 'Class' (containing 'com.k2view.cdbms.usercode.lu.Customer3.MathSubtract'). A blue arrow points from the 'Class' field in the configuration panel to the 'Set the java path to your Java Class' step in the text.

Create a New Actor

New Actor Based on Broadway Flow

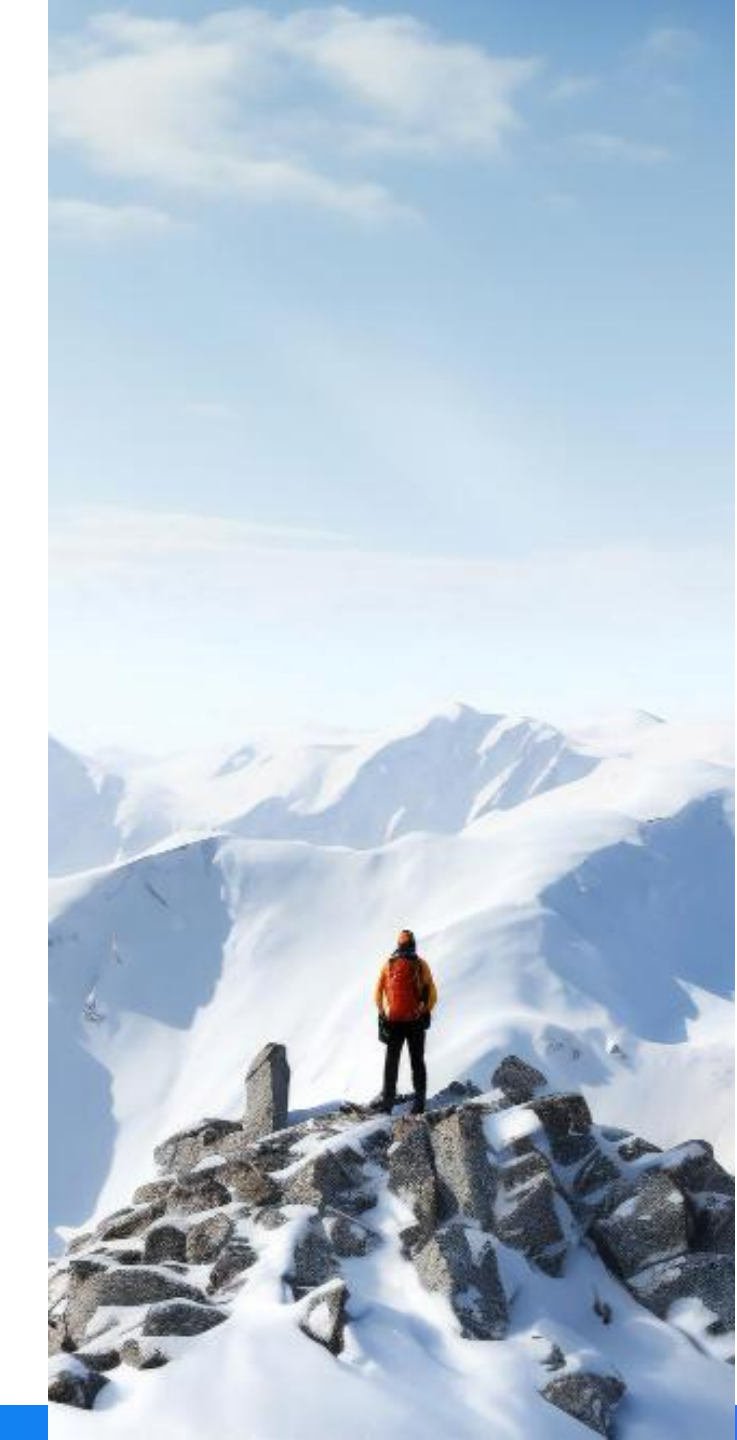
Each BW Flow can be wrapped as a single Actor:

- From the flow menu, select **Save as Actor....**
- Provide the Actor's name and tag, then click **Submit**.
- Inputs and outputs of the flow are automatically added to the Actor.



The screenshot shows a dialog box titled "Save Flow as Actor" with a blue header. Below the header, there is a text prompt: "Please fill actor name and tags (separated by ',')". There are two input fields: "Name" with the text "SleepLog" and "Tags" with the text "Alpinist". At the bottom right, there are two buttons: "Cancel" and "Submit".

Save Flow as Actor	
Please fill actor name and tags (separated by ',')	
Name	Tags
SleepLog	Alpinist
Cancel Submit	



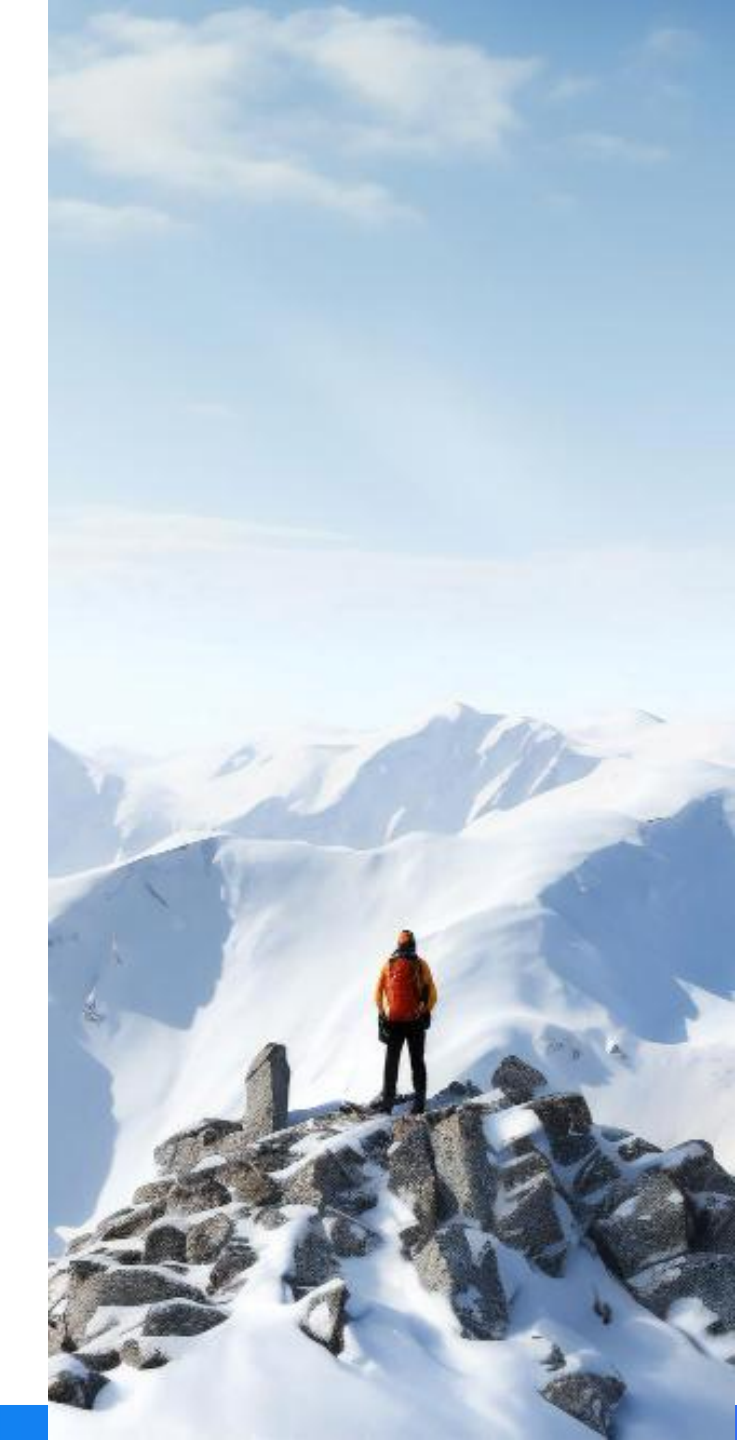
Inherit an Actor

1. Create the Java code

- In the project, create a new Java file (src → right click → New Java File).
- **Extend** the inherited Actor class.
- **Override** the action method. This method is executed when the Actor runs.
- Use input and output maps to handle parameters:
 - Retrieve input: `input.string("inputName")`
 - Set output: `output.put("outputName", value)`

Example - Extending Logger:

```
public class MyLogger extends Logger {  
    @Override  
    public void action(Data input, Data output) {  
        String message = input.string("projectName") + ": " + input.string("message");  
        input.put("message", message);  
        super.action(input, output);  
    }  
}
```



Inherit an Actor

2. Create the new Actor and select the Actor it extends

Two options:

- **Via New Actor**

- *Broadway → Right click → **New Actor**.*
- *Provide a name.*
- *In the Actor's properties, set the **Parent Actor** from which current Actor will inherit*

- **Via Export**

- *Open the Actor's menu → **Export Actor**.*
- *Provide a new Actor name (do not check the Override option)*

The Actor will automatically inherit the Actor's input and output parameters.

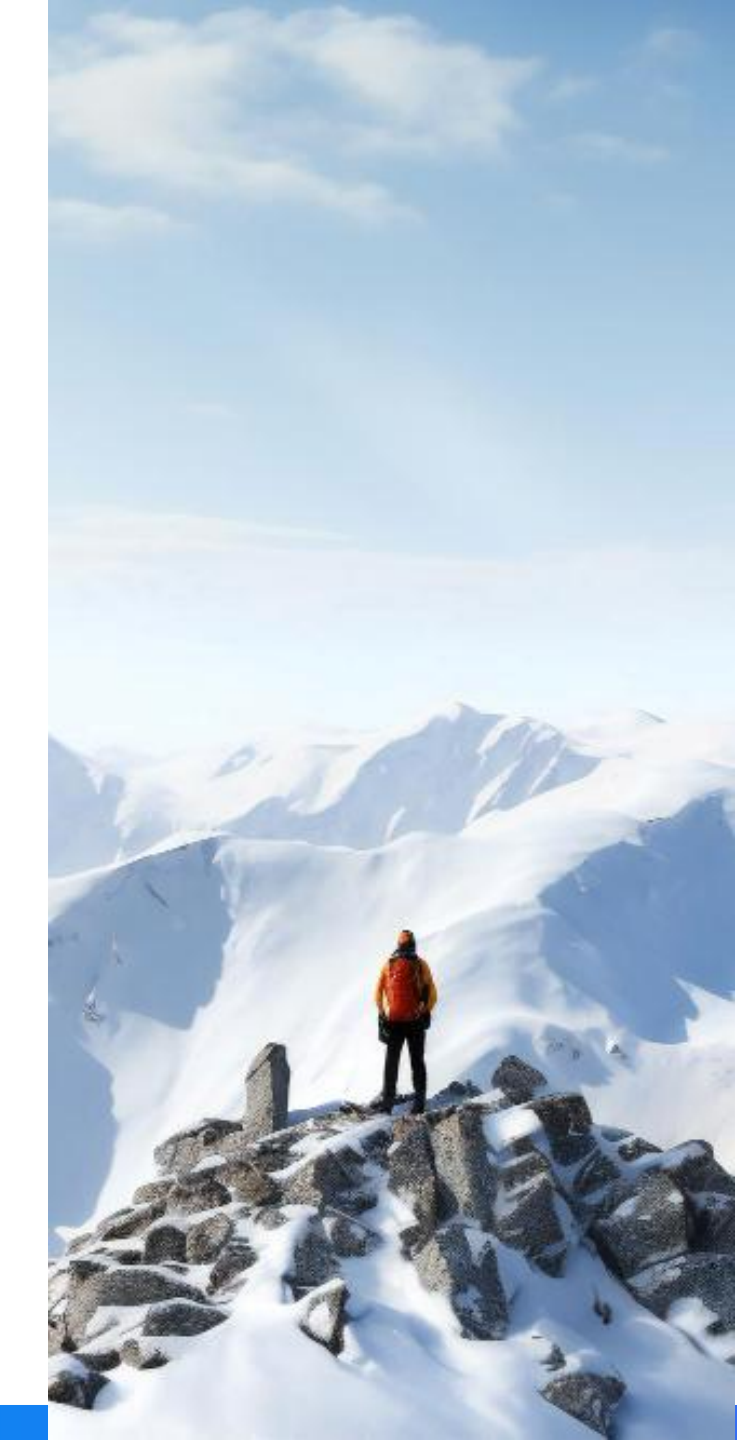
3. In the Actor's properties, set the java path to your Java **Class** (Java class path)

Override a Custom Actor

To override a custom actor:

- Use **Export Actor** on the source Actor and select **Override current**
- After overriding, adjust parameters (defaults, mandatory, modifiers, editor/schema) or add parameters as needed

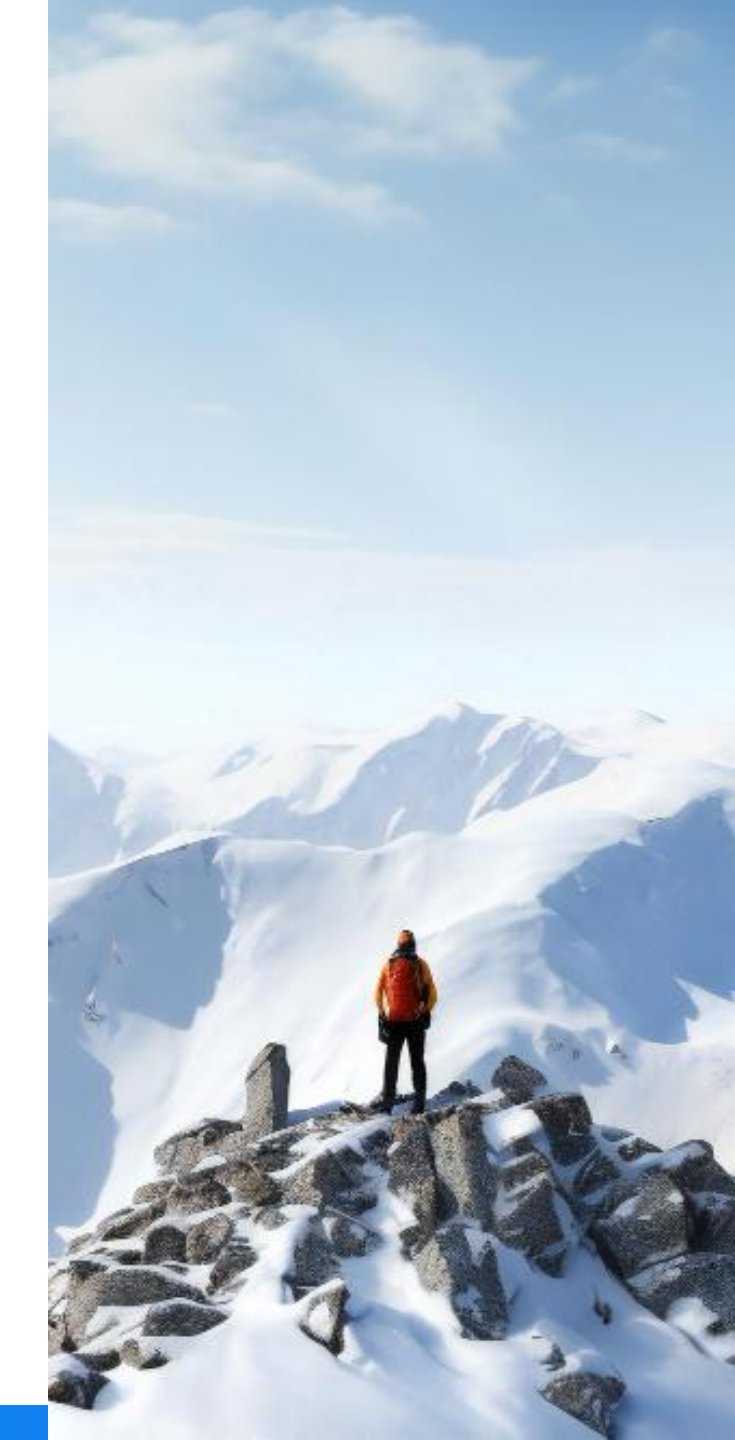


A vertical photograph on the left side of the slide shows a person wearing a red jacket and dark pants standing on a rocky, snow-covered mountain peak. The person is looking out over a vast, snow-covered mountain range under a clear blue sky with some light clouds.

Actor Parameters

When defining a new Actor, each parameter should be configured with the following properties:

- **Default Value** – the initial value assigned to the parameter.
- **Schema** – defines the input type (e.g., Boolean, String, Integer).
- **Editor** – specifies the editor presented to the user when setting the parameter value.
- **Description** – provides an explanation of the parameter's purpose and how it affects the Actor's logic.
- **Advanced Options:**
 - **Mandatory** – whether the parameter must be set.
 - **Modifier:**
 - *Final* – default value cannot be changed.
 - *Hidden* – parameter is not visible to the user.



Schema Property

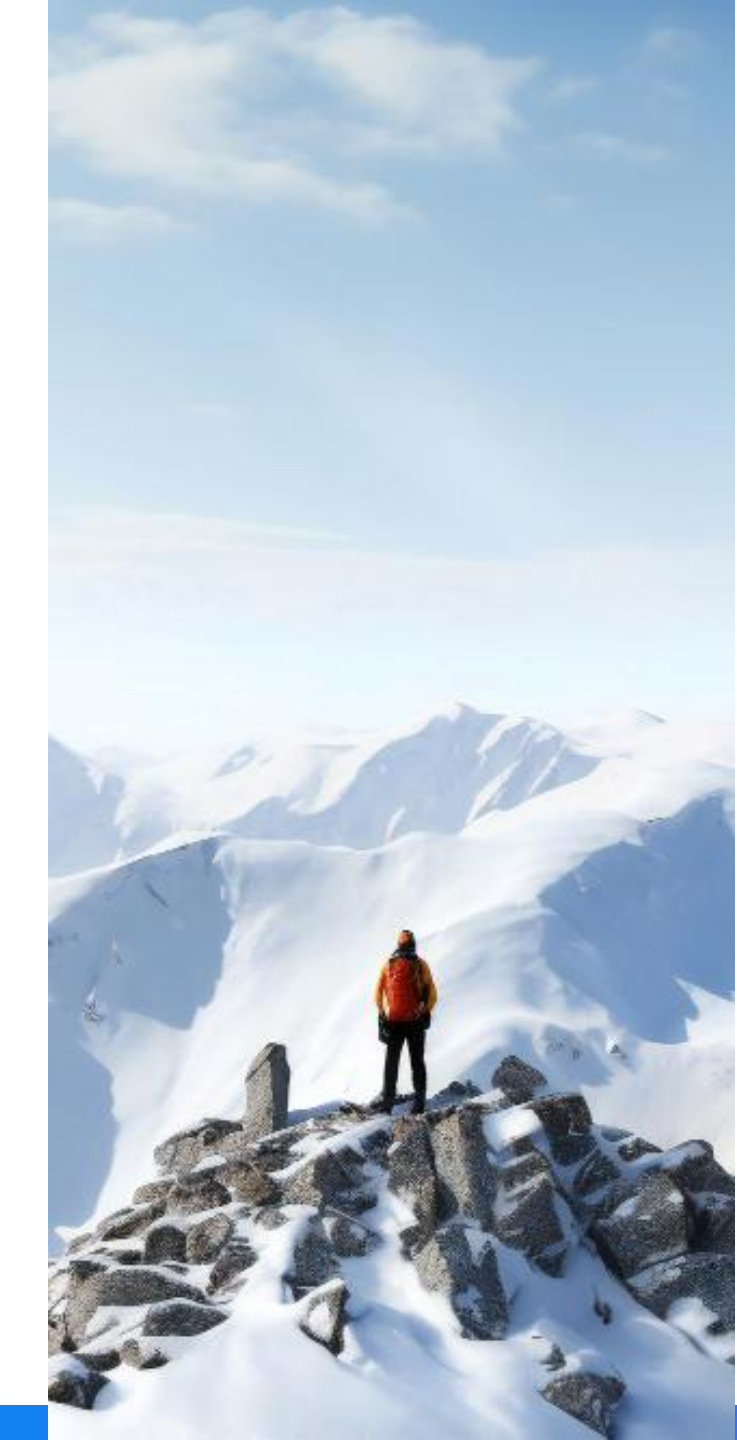
The **Schema** property determines the parameter's type.

Example basic structure:

```
{  
  "type": "string"  
}
```

Supported types:

- string
- integer
- decimal
- number
- date
- boolean
- array
- object



Editor Property

The **Editor** property defines the editor shown to the user when assigning a parameter value.

Default editor:

```
{  
  "id": "com.k2view.default"  
}
```

Common Editor Configurations:

- **Select from Logical Units (with optional empty entry)**

```
{  
  "id": "com.k2view.logicalUnit",  
  "addEmptyEntry": true  
}
```

- **Select from Interfaces (filtered by type)**

```
{  
  "id": "com.k2view.interface",  
  "interfaceType": ["database"]  
}
```

- **Select from available Broadway flows**

```
{  
  "id": "com.k2view.innerflow"  
}
```

- **Drop-down list of values**

```
{  
  "id": "com.k2view.dropdown",  
  "options": ["1", "2"]  
}
```



Editor Property

All available editors are located in:
`/opt/apps/fabric/workspace/fabric/staticWeb/editors`

Supported Editors:

- | | | | |
|-----|---|-----|--------------------------------------|
| 1. | <code>com.k2view.code</code> | 13. | <code>com.k2view.dbtable</code> |
| 2. | <code>com.k2view.distribution</code> | 14. | <code>com.k2view.errorHandler</code> |
| 3. | <code>com.k2view.graphitFiles</code> | 15. | <code>com.k2view.integer</code> |
| 4. | <code>com.k2view.llmPrompt</code> | 16. | <code>com.k2view.mTable</code> |
| 5. | <code>com.k2view.multipleSelection</code> | 17. | <code>com.k2view.schedule</code> |
| 6. | <code>com.k2view.table</code> | 18. | <code>com.k2view.timezone</code> |
| 7. | <code>com.k2view.dataviewer</code> | 19. | <code>com.k2view.default</code> |
| 8. | <code>com.k2view.dropdown</code> | 20. | <code>com.k2view.functions</code> |
| 9. | <code>com.k2view.innerFlow</code> | 21. | <code>com.k2view.interface</code> |
| 10. | <code>com.k2view.logicalUnit</code> | 22. | <code>com.k2view.mTableKey</code> |
| 11. | <code>com.k2view.regex</code> | 23. | <code>com.k2view.strings</code> |
| 12. | <code>com.k2view.textarea</code> | | |

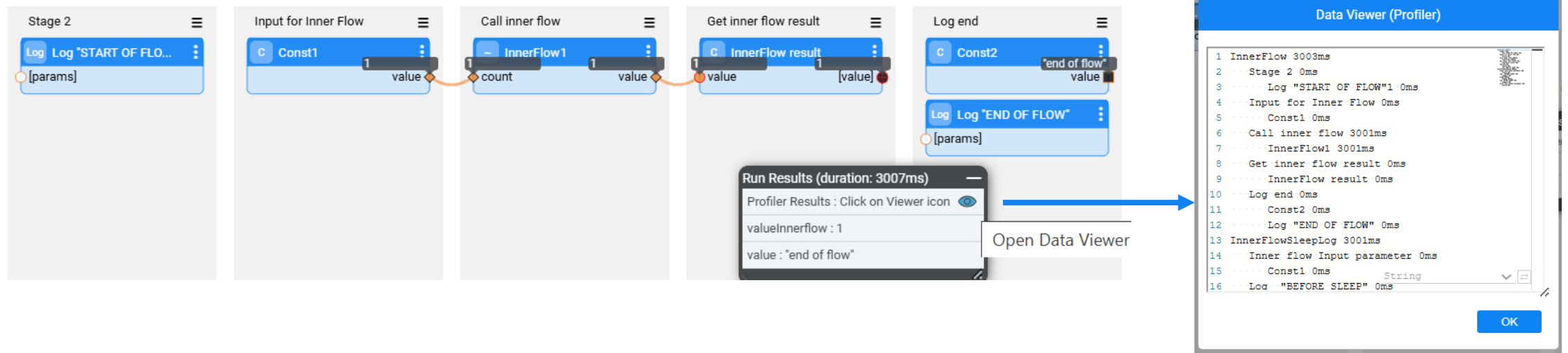
Broadway metrics

Broadway Profiler

The **Broadway Profiler** can be enabled during flow execution to provide a detailed breakdown of results by **Flow, Stage, Actor, and Iteration**. It can be activated from **Fabric Studio** or via the **broadway command**.

Enabling the Profiler in Studio:

- In the **Main menu** toolbar of the **BW flow**, select **Actions > Profiler** and run the flow.
- After execution, the **Run Results** window displays a line: *"Profiler Results: Click on the Viewer icon."*
- Click the **Viewer icon** (image) to open and review the Profiler results.



Broadway metrics

Broadway Profiler

Running the Profiler via Broadway Command

- To enable the Profiler when executing a flow with the **broadway** command, set the profilerTelemetry argument to **true**.
- This adds the Profiler output to the command's results.

```
fabric>broadway CRM.callGraphIt profilerTelemetry=true;
```

```
|column          |value|
+-----+-----+
|profilerTelemetry|callGraphIt 37ms
  Stage 1 6ms
    Http1 4ms
  Stage 2 18ms
    Http2 18ms|
(1 rows)
```



Broadway metrics

Trace Command

Get the metrics of a single BW flow by invoking the trace command of your session, before executing the BW flow.

Syntax:

```
trace [session_scope/global_scope] <TRACE_NAME> '[TRACE_PARAM=[TRACE_VALUES]]';...
```

```
fabric>trace session_scope test_trace 'level=debug;categories=broadway';
|name          |live|since                |operations|resources|categories|level|messages|bytes|maxCols|maxColLen|syncWrite|scope  |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|test_trace|true|2025-08-18 12:57:32.176|broadway  |[]        |broadway  |debug|0        |0      |10      |256      |false   |Session|
```

(1 row)

```
fabric>broadway Customer3.InnerFlow profilerTelemetry=true;
```

test_trace.fabrictrace x

opt > apps > fabric > workspace > traces > test_trace.fabrictrace

```
1 db4b5de9-32f5-4c7c-b370-15725dea7e94,1755521854893,2717,1,start,broadway,info//InnerFlow//flow start//InnerFlow<|>
2 db4b5de9-32f5-4c7c-b370-15725dea7e94,1755521854894,2718,2,interim,broadway,info//InnerFlow//stage start//InnerFlow//Stage 2<|>
3 db4b5de9-32f5-4c7c-b370-15725dea7e94,1755521854894,2718,2,interim,broadway,info//InnerFlow//actor start//InnerFlow//Stage 2//Log "START OF FLOW"1<|>
4 db4b5de9-32f5-4c7c-b370-15725dea7e94,1755521854894,2718,2,interim,broadway,info//InnerFlow//actor end//InnerFlow//Stage 2//Log "START OF FLOW"1//0//<|>
5 db4b5de9-32f5-4c7c-b370-15725dea7e94,1755521854894,2718,2,interim,broadway,info//InnerFlow//stage end//InnerFlow//Stage 2//0//<|>
6 db4b5de9-32f5-4c7c-b370-15725dea7e94,1755521854894,2718,2,interim,broadway,info//InnerFlow//stage start//InnerFlow//Input for Inner Flow<|>
7 db4b5de9-32f5-4c7c-b370-15725dea7e94,1755521854894,2718,2,interim,broadway,info//InnerFlow//actor start//InnerFlow//Input for Inner Flow//Const1<|>
8 db4b5de9-32f5-4c7c-b370-15725dea7e94,1755521854894,2718,2,interim,broadway,info//InnerFlow//actor end//InnerFlow//Input for Inner Flow//Const1//0//<|>
9 db4b5de9-32f5-4c7c-b370-15725dea7e94,1755521854894,2718,2,interim,broadway,info//InnerFlow//stage end//InnerFlow//Input for Inner Flow//0//<|>
10 db4b5de9-32f5-4c7c-b370-15725dea7e94,1755521854895,2719,3,interim,broadway,info//InnerFlow//stage start//InnerFlow//Call inner flow<|>
11 db4b5de9-32f5-4c7c-b370-15725dea7e94,1755521854895,2719,3,interim,broadway,info//InnerFlow//actor start//InnerFlow//Call inner flow//InnerFlow1<|>
12 db4b5de9-32f5-4c7c-b370-15725dea7e94,1755521854895,2719,3,start,broadway,info//InnerFlow//flow start//InnerFlowSleepLog<|>
```

≡	k2statistics
-	broadway
-	broadwayFlow The execution duration of a Broadway flow
last	0:00:03.004
average	0:00:02.852
count	19
timestamp	2025-08-18 12:57:37.897 UTC
since	0:02:07.561
total	0:00:54.200
-	InnerFlow
last	0:00:03.004
average	0:00:03.004
count	3
timestamp	2025-08-18 12:57:37.897 UTC
since	0:02:07.561
total	0:00:09.012
+	InnerFlowAsyncWithJoin
+	InnerFlowSleepLog
+	bwPubSub
-	broadwayLoop The execution duration of a Broadway loop (flow)
last	0:00:00.003
average	0:00:00.003
count	19
timestamp	2025-08-18 12:37:59.986 UTC
since	0:21:45.472
total	0:00:00.073
+	InnerFlowAsyncWithJoin.Input for Inner Flow
+	bwPubSub.Itterate over the batch
+	bwPubSub.Stage 1

Broadway metrics

JMX Stats

JMX Stats provide runtime performance metrics for Broadway flows, broken down by **Flow, Stage, Actor, and Iteration**.

This allows monitoring and analysis of flow behavior across multiple executions, complementing the single-run insights from the Profiler.

Recovery point

Recovery Points provide a mechanism to resume execution from a defined stage instead of restarting the entire flow.

- When a Recovery Point is set, Broadway serializes the flow's data and stores it in the `broadway_recovery_point` table under the `k2system` keyspace.
- If a failure occurs (e.g., outage), the flow can restart from the last saved point.
- Once execution completes successfully, the recovery data is automatically removed from the System DB
- Recovery Points are best used after completing a sub-process and before starting the next major stage of the flow.

1 select *

2 from k2system_shani_mountain8_solutions.broadway_recovery_point

Columns to show ▾

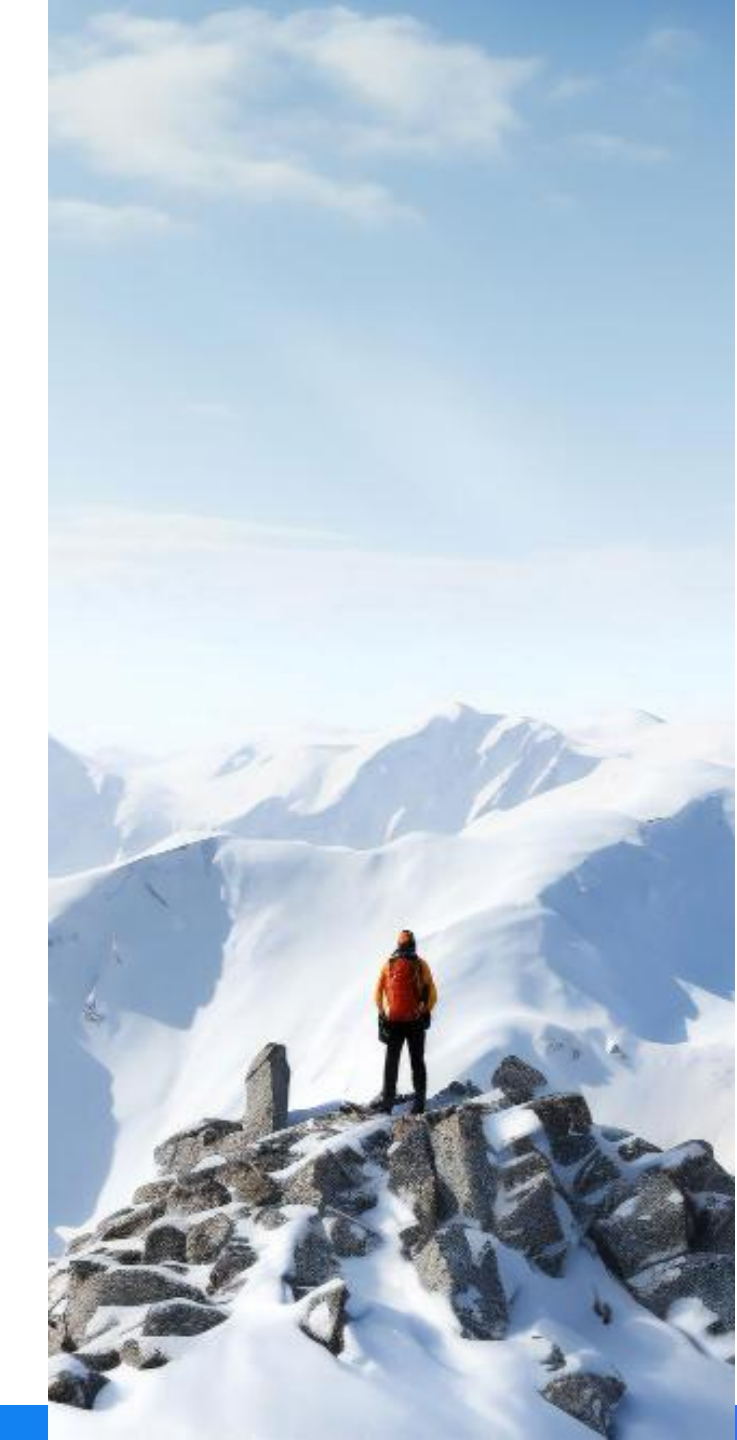
↓ CSV

Displaying Records: 1 | Query executed successfully | Duration: 6 ms | Affected Records: 1

< id ▾	< data ▾	< first_execution ▾	< last_execution ▾	< maxtries ▾	< row_ttl ▾	< tries ▾
debug.Customer3.RecoveryEx...	{"hb":["31,-117,8,0,0,0,0,0,-1,-...	1755780846265	1755780849342	100000	86400	0

Limitations: Recovery Points cannot be set on:

- Stages with DB result sets
- Transactional Stages
- Stages inside iterations



Recovery point

How to Set a Recovery Point

- Open the Stage context menu (:).
- Select Recovery Point → a recovery icon appears on the Stage.
- Repeat for additional Stages if multiple Recovery Points are required in the flow.

Running a Flow with Recovery Point

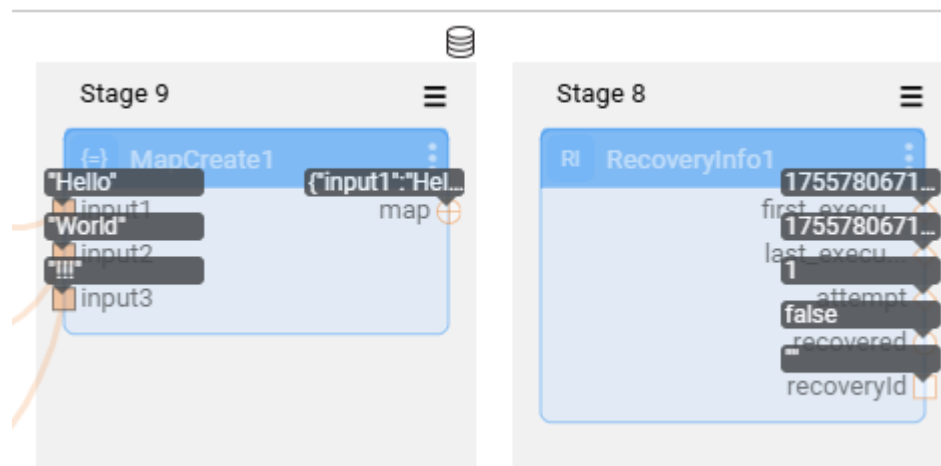
Broadway flows with Recovery Points can be executed in three ways:

- Via the BROADWAY Command
 - *Must include a Recovery ID to enable the Recovery mechanism.*
 - *If a crash occurs, rerun with the same Recovery ID to resume.*
- Via the STARTJOB Command (Broadway Job)
 - *Recovery mechanism is enabled automatically.*
 - *No need to specify a Recovery ID.*
- Via Fabric Studio (Simulation)
 - *Set a breakpoint after the Recovery Point.*
 - *Select Actions > Run with Recovery Point.*
 - *When execution hits the breakpoint, click Stop Run to abort.*
 - *Rerun with Recovery Point → flow resumes from Recovery Point.*

Recovery point

RecoveryInfo Actor

- The **RecoveryInfo Actor** can be used to retrieve runtime recovery details, including:
- Recovery ID
- Number of recovery attempts
- This Actor should be placed in the flow **after** a Recovery Point.



Automatic Flow Execution on Deploy

A Broadway flow can be executed automatically during a Logical Unit (LU) deploy.

- If a `deploy.flow` is defined under an LU, it will be triggered every time that LU is deployed.
- If `deploy.flow` exists only at the Shared level, it is inherited by all LUs.
- If a Soft Deploy is used, the `deploy.flow` will not be executed.

Auto-Generated `deploy.flow`

When a new LU is created, a `deploy.flow` is automatically generated with predefined constants:

- `lu_name` – Name of the deployed LU.
- `nosync` – Controls synchronization behavior:
 - *`NOSYNC=TRUE`: Only schema changes trigger a sync after deploy.*
 - *`NOSYNC=FALSE`: Any deploy triggers a sync the first time an instance is accessed.*
- `is_first_deploy` – Boolean indicating whether this is the LU's first deploy.
- `is_studio` – true if deployed in the Studio debug environment.



Best practice:

Add Broadway jobs to `deploy.flow` if they must automatically restart after a deploy.