BPMN Default Error Handling

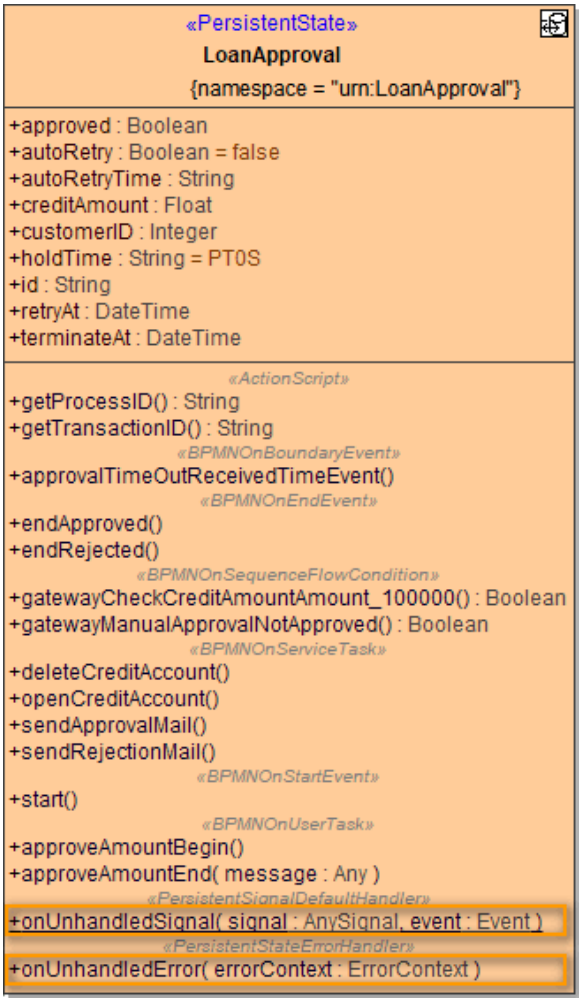
The generated model contains prepared procedures for

- unhandled signals,
- unhandled errors and
- retrying the execution of the process, in case the error has been fixed.

The service interface part (e.g. **BPMNLoanApprovalServices**) contains a persistent state class that is derived from the abstract definition generated from the BPMN. This class contains overridable operations that are called by the xUML Runtime on unhandled signals and errors.

On this Page:

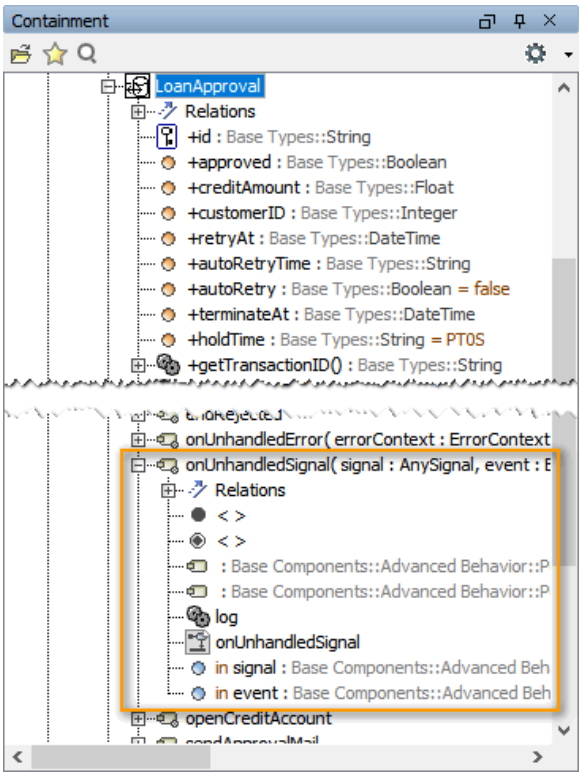
- Signal Errors
- Unhandled Errors
- Retrying Process Execution
  - Manual Retry
  - Automatic Retry



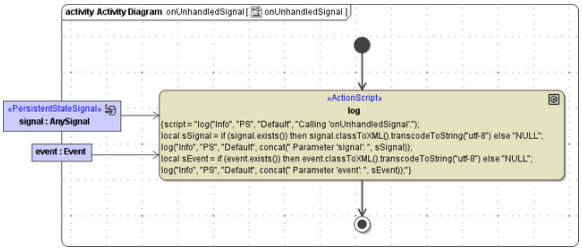
You can amend these operations as needed.

Signal Errors

Unhandled signals are caught by the xUML Runtime and the overridable operation **onUnhandledSignal** is called. This operation is defined in the derived persistent state class (see figure above).



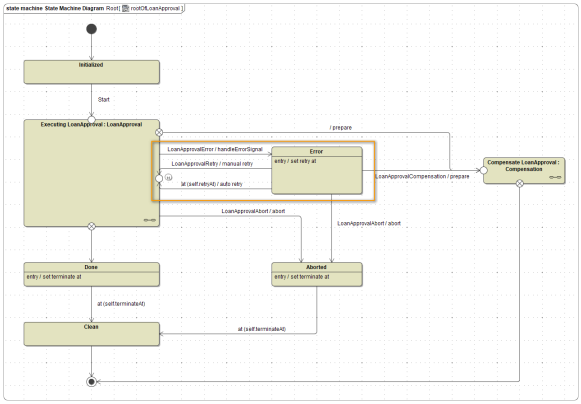
By default, unhandled signals are logged to the bridgeserver log .



You can adapt this activity diagram as needed (e.g. return of an error code to the caller).
Except for the behavior described above, unhandled signals are ignored by the Runtime.

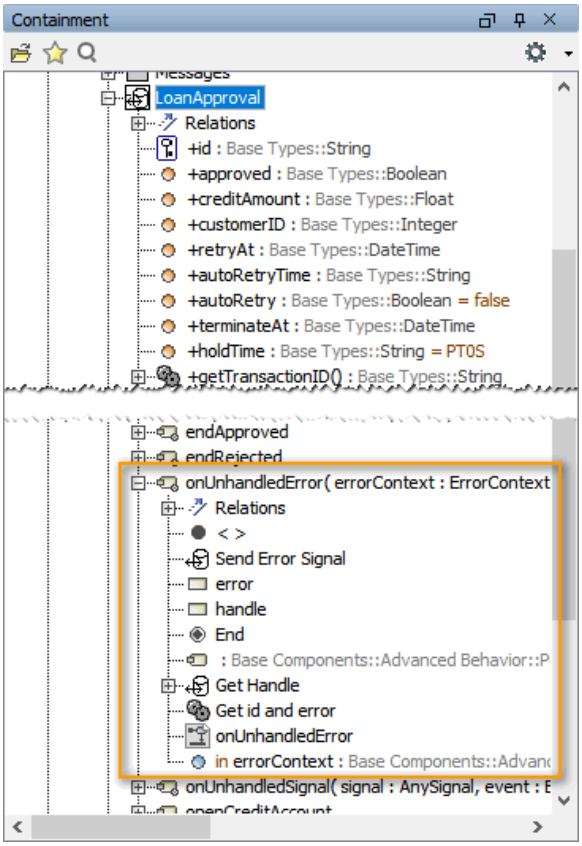
Unhandled Errors

The generated model contains a **root state machine** that invokes the process state machine. This root state machine contains a default error state that manages all unhandled errors. It is designed to handle technical errors during the course of the process.
After the error having been caught and rectified, the process can be continued by sending a retry signal.

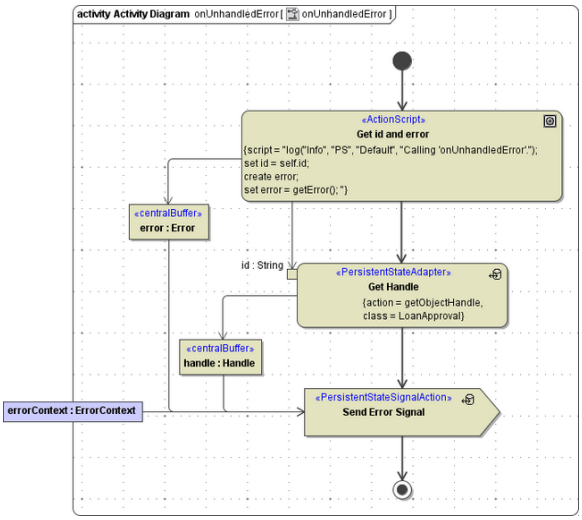


When an error occurs during process execution, the overridable operation **onUnhandledError** defined in the derived persistent state class (see figure above) gets executed.

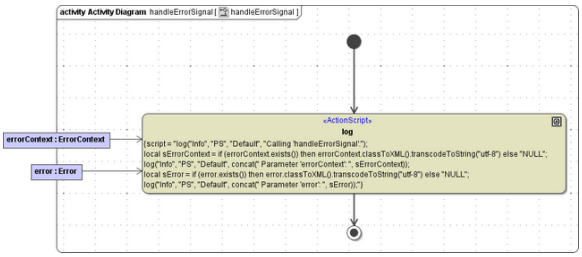
> Unhandled errors are caught by the xUML Runtime, no exception is thrown. This results in database transactions not being rolled back or committed. You need to do this manually in the related error handler if needed (see below).



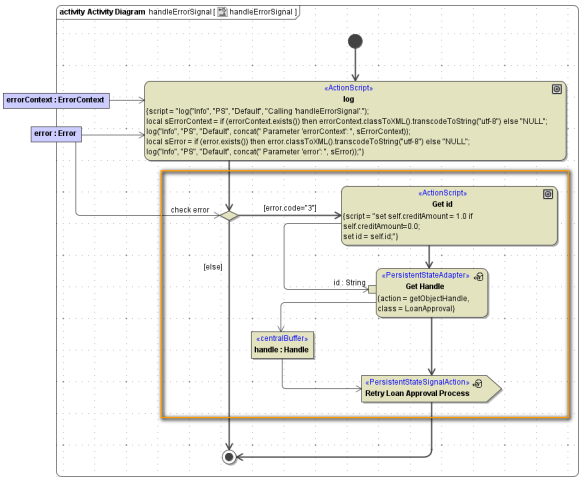
On unhandled errors, the error signal is send to the persistent state object.



You can adapt this error procedure as needed.
On receiving the error signal, activity **handleErrorSignal** is executed and, by default, logs the error to the bridgeserver log.

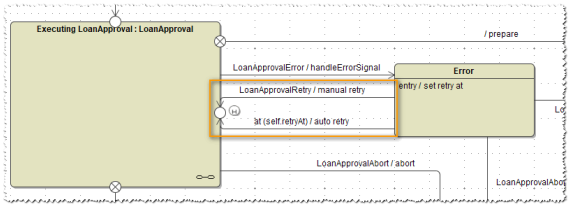


You can adapt this activity diagram as well, e.g. to implement retry logic.



Retrying Process Execution

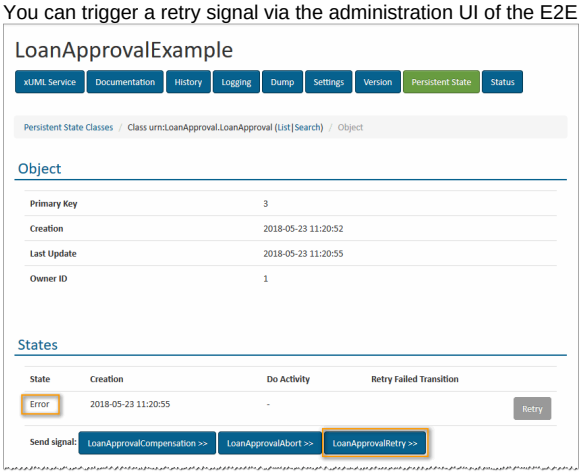
The generated root state machine features automatic and manual retry of process execution.



Manual Retry

If the process is stalled in an error state, you can manually send a retry signal to the process. This can be done in two ways:

- You can trigger a retry signal via the administration UI of the E2E Bridge.



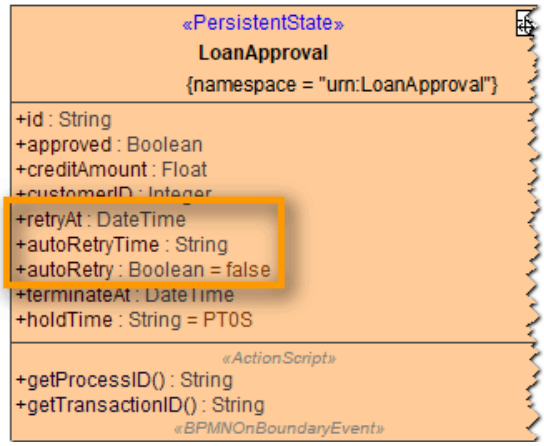
  To do this, the retry signal must not have any parameters (as is in the default implementation).
- You can implement a service operation that triggers the retry and call this operation e.g. via the E2E Analyzer. In this case, you may add parameters to the retry signal, e.g. to rectify erroneous data.

> Keep in mind, that you will not be able to send such a retry signal with parameters via the administration UI of the E2E Bridge anymore.

It may be that you need to resolve the root cause of the error first before retrying the process, though.

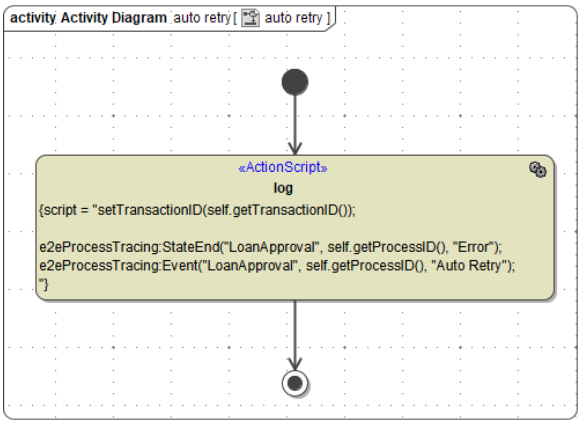
Automatic Retry

You can configure the root state machine to perform automatic retries after a specified period of time. This configuration can be done on properties of the persistent state class.



Property	Type	Description	Values	
autoRetry	Boolean	Specify whether automatic retry is activated or not.	true	Automatic retry is activated.
			false	Automatic retry is disabled (default).
autoRetryTime	String	Specify a duration pattern that defines the interval in which an automatic retry should be performed.	A valid duration pattern, like described on Time Durations, e.g. `PT1H`.	
retryAt	DateTime	This field cannot be set by the modeler but is calculated in case of retry using **autoRetryTime**. It indicates the point in time when the next retry will be performed.	A date/time.	

If a process is in error state and the point in time indicated by **retryAt** has been reached, the retry will be triggered by the root state machine (see `at(self.retryAt)/auto retry` in the root state machine).



As per default, the activity diagram performing the retry contains a logging action for the E2E Process Dashboard. You can add additional code if necessary.