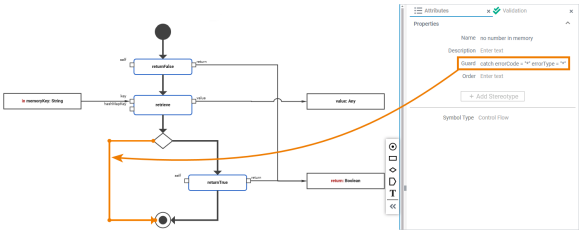
Catching Errors

If an error occurs, the default behavior is that the service execution is stopped at this point, and an error message is sent back. This behavior can be changed by catching the error in your activity diagram.

Fatal errors, however, cannot be caught. Depending on where the fatal error occurs, these errors are not sent back to the client - they are written to the log file of the xUML service. For example, if an xUML service starts up and cannot find needed add-ons, this error is logged and the service shuts down. A list of log errors can be found at Log Errors.

You can catch non-fatal errors by using domain and code values. These values are listed on System Errors.
You can catch system error in the outgoing flow of a decision. Define a catch statement as a guard expression, and supply error domain and code as listed on System Errors. Besides using exact code values like 12, it is also possible to use a wildcard (`{*}`) for all errors of the specified type. Such an example is shown in the following activity diagram.

In the example below, the operation checks if there has been data stored to memory at all (`errorCode="*"`). If not (error case), the operation returns false, else the operation returns true.



It is also possible to use variables and provide the error code dynamically.

Where Errors Need to be Caught

An error can be caught

- either immediately after the action node that throws the error (for instance a MongoDB adapter action),
- or directly after the operation call that calls a sub-activity containing the erroneous action node.

In the latter case, within the activity diagram that implements the operation, any activity step after the error has occurred is **not** executed. As a rule, if the error is not caught in the activity diagram, the Runtime passes the error to the caller (the calling sub-activity). However, in order to catch the error, the decision needs to be placed directly after the operation call.

On this Page:

Related Pages:

- getError()
- getErrors()
- Log Errors
- System Errors

Accessing Error Objects

Errors that are caught can be accessed by using the error functions getError() and getErrors(). Both functions return one resp. multiple object(s) of type **Error**.

```
{
        "callstack": ["Callstack"],
        "category": "String",
        "code": "String",
        "description": "String".
        "detail": "Any",
        "domain": "String",
        "response": "Blob",
        "timestamp": "DateTime"
}
```

The error object contains information on the **callstack**, some more error **details**, **response** information (e.g. from a REST call), and an error **timestamp**.