Genesys Intelligent Automation Reference Guide

Call Processing

12/14/2025

Contents

Call Processing

This page describes the logic applied at the end of calls. It also describes associated backlog processing, if necessary.

End-of-call logic

When a call ends, Genesys Intelligent Automation writes the call records to the configured database. A call ends under the following conditions:

- Intelligent Automation receives a hang-up event from MCP.
- There are no more blocks to process in the callflow.
- The call reaches the **End Call** block (can be configured either to disconnect or return to strategy).
- The call reaches a **Transfer** block.
- Session timeout due to no MCP requests during the configured timeout period.

  The **<session-timeout>** parameter value, found in the application **web.xml** file, defines this timeout period in seconds. In the example below, the session timeout period is set at 30 seconds:

```
<session-config>
 <session-timeout>30</session-timeout>
 </session-config>
```

> **Important**
>
> The **<session-timeout>** parameter resets to the default value each time you upgrade Intelligent Automation. Therefore, you must update this value after each upgrade if you do not want to use the default value.

Backlog processing

> **Important**
>
> Backlog processing of calls only occurs if the **Backlog.Processor.Enabled** setting is set to **true** and the hive-off process is not running. Backlog processing is paused during the hive-off process.

When a call ends, Intelligent Automation attempts to write the call to the database. If this process fails, Intelligent Automation sends the call to backlog processing.

The table below, **Handled Database Write Errors**, describes reasons for why the write might fail. If the write fails for another reason that is not described in this table, the call is unrecoverable and its data is lost.

Handled Database Write Errors

Database timeout
Connection pool exhaustion
The SQL Exception from the JDBC driver matches one of the following: • 08 – Connection Error • 66 – Driver Error • HY – Operation Cancelled • S0001 – SQL Login Failed

Configuring backlog processing

You can configure the settings below to periodically check the the backlog folder size:

- **Backlog.DiskSpaceMonitor.CheckIntervalMillis** - Specify, in milliseconds, how often to check the size of the backlog folder.
- **Backlog.DiskSpaceMonitor.WarningUsedMB** - Specify, in megabytes, the size that the backlog folder must exceed before a WARNING-level SMTP alert is sent.
- **Backlog.DiskSpaceMonitor.MaxUsedMB** - Specify, in megabytes, the size that the backlog folder must exceed before a MAJOR-level SMTP alert is sent. This is the size that, once reached, data is lost.

As of the 3.5.100.04 release, you must configure the following settings to specify which call-records errors are sent to the backlog:

- **Backlog.TreatAllFailuresAsBackloggable** - If `true`, all possible errors are sent to the backlog. If `false`, you can specify which errors are backlogged by configuring the following settings:
  - **Backlog.BackLoggableErrorsList.ExactMatch** - Specify an error code that must be matched. For example, `S0001`.
  - **Backlog.BackLoggableErrorsList.StartsWith** - Specify a comma-separated list of prefixes for accepted error codes. For example, `08,66,HY`.

> ### Important
> If you set **Backlog.TreatAllFailuresAsBackloggable** to `true`, Intelligent Automation sends all records to the backlog, even if these records cannot be backlogged. This movement consumes bandwidth until the records are manually removed from the backlog directory on the disk.

Sending backlog items to the database

If processing continues, the call data is serialized to XML, saved in an XML file on disk, and added to the internal file backlog list. You can use the setting **Backlog.Processor.MinItemAgeBeforeProcessingMillis** to specify, in milliseconds, how long Intelligent Automation must wait before trying to re-insert an item from the backlog back into the database.

Multiple backlog processor threads process any items that are added to the backlog. The **Backlog.Processor.ThreadCount.CallHistory** server setting defines the number of threads to use. Consequently, this value is also the maximum number of concurrent database writes from the backlog, regardless of database pool availability.

> ### Important
> The backlog shares the connection pool with the main application. Therefore, pool exhaustion might result if the **Backlog.Processor.ThreadCount.CallHistory** server setting is set too high.

Scenario

The following provides an example of how backlog processing functions.

Consider the following scenario:

- Maximum Pool Size = **100**
  - Main Application Connection Usage = **95**
  - Backlog Thread Count = **10**

The total number of items is **105**, which is five more than the maximum pool size. The excess items are sent to the backlog.

> ### Important
> Intelligent Automation does not guarantee whether excess items come from the backlog or the main application. Therefore, you must provide some headroom on your maximum pool size configuration. You must also account for the number of VUI servers and gauge whether the database server can cope with the number of total connections configured per server.

Once items are in the backlog, each backlog processing thread tries to grab an item from the backlog and re-insert it into the database. If no backlog items exist, the processing threads sleep for 20 seconds. This sleep value is not configurable.

Once a thread finds an item to process:

1. It attempts to move the file into the processing folder.
2. It de-serializes the file contents back into call data and deletes the XML file.

> **Important**
>
> If an error occurs in one of the first two steps, the file moves permanently to the *failed* folder and is no longer accessible to the backlog processing threads. This is the only scenario by which a file is moved to the *failed* folder and abandoned. In other words, the *failed* folder is not used for calls that cannot be re-saved. In that case, a backlog thread continues to put the call into the work queue without limit on failed attempts (unless the error does not match one of the handled errors referenced above).

3. It attempts to re-save the call data. One of the following occurs:
   - If the re-save is successful, the thread moves onto the next item in backlog.
   - If the save fails but it is a handled error, the thread adds a new backlog item for the call data. The thread that tried to process the item sleeps for 60 seconds (this value is non-configurable). The new item is not processed by another thread until the value of **Backlog.Processor.MinItemAgeBeforeProcessingMillis** has passed.
   - If the save fails and it is an unhandled error (not part of the list of handled errors, call data is lost.

Notes

- There is no limit on the the number of times a backlog thread can fail to process an item. Each time it fails, the item is added to the backlog again.
- The same code executes each time a backlog thread attempts to re-save an item. Therefore, a thread in the reporting connection pool is used each time an attempt is made.
- The only server setting that you can change at runtime is **Backlog.Processor.Enabled**. All other settings require a restart of the server.