Mirth Connect Upgrade Guide

This guide is for upgrading your installation of Mirth Connect 2.1 or later. There are a few steps that must be performed before upgrading to backup your database and configuration properties.

Please read this!
It is important to follow all of the steps below, to ensure that you do not lose any data. Be sure to check out the Channel updates section for version specific changes that aren't automatically performed during the upgrade process.

Message Migration
If you are upgrading to Mirth Connect 3.x from Mirth Connect 2.x, your existing messages will not be upgraded. This includes any queued messages that have not been sent, as well as any historical message records. After upgrading, Mirth Connect 2.x messages will not be available in the Mirth Connect Administrator, but will remain in the database in the OLD_MESSAGE table. Please see Other instructions for 3.0.0 for information about flushing out queued messages.

From here on, $MIRTH_HOME refers to the Mirth Connect installation directory.

If you are using a Mirth Appliance...

Congratulations! The upgrade process is done automatically when the appliance is upgraded.

1. Before you upgrade

Before upgrading, we highly recommend that you backup both your Mirth Connect Server Configuration and your Mirth Connect database.

1. **Backup server configuration**: Before proceeding, it's important that you backup your existing server configuration. This can be done through the Mirth Connect Administrator under Settings. The server configuration includes channels, code templates, alerts, and global scripts. User accounts and stored messages will not be backed up.
2. **Backup database**: If you are using the embedded Derby database, create a backup of the $MIRTH_HOME/mirthdb. If you are using an external database (ex. PostgreSQL, MySQL), create a backup of your database using your preferred database administration tool (ex. pg Admin, MySQL Administrator).

Changes to some custom configuration files may need to be manually copied over after the upgrade. If you've made changes to any of the following files, copy them to a location outside of your existing $MIRTH_HOME directory before running the new Mirth Connect installer. Once Mirth Connect is upgraded, you will need to update the newly installed files with your custom changes.

1. **Custom database drivers**: If you have added custom database drivers they will be preserved in custom-lib during the upgrade. However, you will need to back up your additions to $MIRTH_HOME/conf/dbdrivers.xml
2. **Java JVM parameters**: If you have added any Java JVM parameters or changed any of the default heap size settings, create a backup of any $MIRTH_HOME/*.vmoptions files
3. **Logger settings**: If you have made any changes to your logger settings, create a backup of $MIRTH_HOME/conf/log4j.properties
4. **CLI configuration**: If you have made any changes to the CLI configuration file, create a backup of $MIRTH_HOME/conf/mirth-cli-config.properties

Custom extensions may not work after the upgrade and you may see errors in the log. You will need to install an updated extension that is compatible with the new version of Mirth Connect.

2. Upgrading

1. Make sure that you have exited and stopped the Mirth Connect Administrator, Server Manager, and Server/Service before proceeding.
2. Run and complete the new Mirth Connect installer, choosing the update option using the same installation directory.

3. After the upgrade

1. **Custom database drivers**: Restore your changes to $MIRTH_HOME/conf/dbdrivers.xml
2. **Java JVM parameters**: Restore the changes in your backed up *.vmoptions files to the corresponding *.vmoptions files $MIRTH_HOME.

3. **Logger settings**: Set any properties you changed in your backed up $MIRTH_HOME/conf/log4j.properties manually.
4. **CLI configuration**: Restore the changes in your backed up $MIRTH_HOME/conf/mirth-cli-config.properties file.
5. **Start/Restart the server**
6. **Custom extensions**: Install new versions of any custom extensions

Channel updates

These are version specific changes to channels that are not done automatically during the upgrade process. Only channels with certain connectors or deprecated code references are affected by these updates. Versions that may require specific changes are listed below. For example, if you are upgrading from 2.1.0 to 2.2.0, you should look at any notes below for both 2.1.1 or 2.2.0.

2.2.0

The 'Email Sender' has been renamed to the 'SMTP Sender' and it no longer has the option to 'Use Server Settings' for the SMTP server configuration. If your channel has this configuration, your destination and/or channel will be disabled. You will need to manually enter your SMTP server settings and enable your destination/channel again after upgrading.

3.0.0

The API in Mirth Connect 3.0 has changed dramatically. All old JavaScript references and variables from the Reference List should still work properly, but they may log messages stating the code you are using has been deprecated, and recommend code changes that you should complete. Any deprecated methods may be removed in an upcoming version.

Calls to any unsupported internal Mirth Connect code may have changed and may no longer work. This includes any JavaScript that has to import or reference anything in the com.mirth.connect.* package. If you are using any unsupported internal Mirth Connect classes to perform actions like starting and stopping channels, please take a look at Mirth Connect 3.0's new ChannelUtil API.

3.0.2

The Database Reader no longer prepends table name to the column name or alias regardless of how many tables are used in the query. Each element name in the generated XML will be the alias if one exists, otherwise it will be the column name. Users connecting to Postgres or Oracle databases with the Database Reader connector should not be required to make any changes. Users connecting to other databases should update their channels accordingly if needed. It is recommended to always use a unique alias.

Code templates in 3.0.0 and 3.0.1 were not being restricted from certain JavaScript scopes based on their context. Users upgrading from these versions to 3.0.2+ should check their code templates to ensure their contexts are correctly set. See MIRTH-3150 for more information.

Other instructions

These are version specific instructions that are not done automatically during the upgrade process. Versions that may require special instructions are listed below. For example, if you are upgrading from 2.1.0 to 2.2.0, you should look at any notes below for both 2.1.1 or 2.2.0.

3.0.0

Any messages that were queued at the time of upgrade from 2.x will not be processed. If you would like to ensure these messages are processed, we recommend flushing out all queued messages before upgrading:

For non-polling channels:

- Pause the channel
- Wait for all queued messages to finish processing

For polling channels:

1. Edit the channel, export the source connector (to have a backup)
2. Change the source connector type to Channel Reader

3. Redeploy the channel
4. Wait for all queued messages to finish processing
5. Edit the channel again, restore the exported source connector, but do not redeploy

Do the above for all channels until there are no more queued messages. If any queued messages cannot be processed because the external system is down, you can search for the queued messages and export them as XML or Plain Text for manual processing later.

To simply remove all queued messages you can remove them from the Message Browser, which should also remove them from the file system, or simply delete the queuestore folder (or individual channel folders inside of queuestore) in your appdata directory.

3.1.0

A new index has been added to the message metadata table for each channel to improve queue performance . Channels created in 3.1.0 will automatically create this index, but existing channels will need to have the index created manually:

1. Stop all channels that require the index
2. Navigate to Settings > Database Tasks
3. Select the "Add Metadata Index" task
4. Click Run Task

If you do not see the "Add Metadata Index" task, then the index already exists for all of your channels and there is no action needed. If the task fails to add the index for all of your channels, ensure that the remaining affected channels are stopped or undeployed and run the task again.

Ensure you have enough disk space before adding the new index. The index will require roughly 45 megabytes of disk space per connector for every million messages currently stored in the database. It may take some time depending on how many messages are currently stored, so it is recommended to schedule some downtime for adding the index.

3.1.1

Due to the recent POODLE vulnerability, we have disabled SSLv3 for all of our relevant connectors and connections that use SSL. In case you are connecting to some legacy systems that require SSLv3, it is possible to re-enable SSLv3. In your mirth.properties file, you will find the following two properties:

- https.client.protocols = TLSv1.2,TLSv1.1,TLSv1
- https.server.protocols = TLSv1.2,TLSv1.1,TLSv1,SSLv2Hello

If you need to use SSLv3, simply add "SSLv3" into the comma separated list for the client or server protocols and restart your server. These two properties also allow you to configure exactly which SSL protocols you wish to use. Similarly, if you need to add or remove cipher suites, you can update the https.ciphersuites property in the mirth.properties file.

Please note that if you are making https connections programmatically in JavaScript, you will still need to update your code accordingly in order to disable SSLv3.

When upgrading to Mirth Connect 3.1.1+ OR Java 8, some users have experienced issues connecting to the Administrator or external SSL endpoints which previously worked fine. This is likely because some servers do not implement forward compatibility correctly and refuse to talk to TLS 1.1 or TLS 1.2 clients. TLSv1.2 and TLSv1.1 are enabled by default in Java 8. As such we have also decided to enabled them by default beginning with 3.1.1 regardless of Java version. If you experience these issues, you should first try disabling TLSv1.2 and TLSv1.1 by updating the https.client.protocols property and restarting the server

- https.client.protocols = TLSv1

If you are still unable to connect to the Administrator, you can try also disabling TLSv1.2 and TLSv1.1 on the server protocols. For security reasons, It is recommended to leave the default settings unless you are explicitly running into issues connecting to SSL endpoints.

3.2.0

JRE 1.6 (Java 6) is no longer be supported as of 3.2.0. If you are still using JRE 1.6, please ensure you update to at least JRE 1.7 before upgrading to Mirth Connect 3.2.0+. However future releases in the 2.x, 3.0.x, and 3.1.x branches will still continue to support JRE 1.6 unless otherwise stated.

The default connection pool used for message processing has been updated to HikariCP. We've seen that HikariCP is generally faster and more reliable than DBCP although this change should be imperceptible to most users. Support for DBCP still exists and users can revert back to it if necessary by adding the following line to mirth.properties and restarting the server.

- database.pool = dbcp

A bug in the configuration map was fixed where certain special characters were not being handled correctly (MIRTH-3542). Since this bug affected both the saving and loading of the configuration map, there is a chance your configuration map values will be loaded differently after

upgrading Mirth Connect. Please verify your configuration map values after upgrading.

3.4.0

Mirth Connect plugins that implement custom client <-> server communication will need to be updated to use the new HTTP servlet architecture introduced in version 3.4.0.

Specifically, the invoke() method has been removed from the ServicePlugin interface as well as the ClientPlugin abstract class. Prior to 3.4.0, calls to ClientPlugin.invoke() on the client were routed to ServicePlugin.invoke() on the server.

Plugins that previously used these methods must now provide a separate servlet interface on the server side using JAX-RS and Swagger annotations:

ServerLogInterface.java

```
@Path("/extensions/serverlog")
@Api("Extension Services")
@Consumes(MediaType.APPLICATION_XML)
@Produces(MediaType.APPLICATION_XML)
public interface ServerLogServletInterface extends BaseServletInterface {
    public static final String PLUGIN_POINT = "Server Log";
    public static final String PERMISSION_VIEW = "View Server Log";

    @GET
    @Path("/")
    @ApiOperation("Retrieves all server log entries.")
    @MirthOperation(name = "getMirthServerLogs", display = "View Server Log",
permission = PERMISSION_VIEW, type = ExecuteType.ASYNC, auditable = false)
    public LinkedList<String[]> getServerLogs() throws ClientException;
```

To invoke servlet methods from the client side, use the getServlet() method instead of the previous invokePluginMethod* methods.

3.3.x and earlier:

ServerLogClient.java

```
try {
    serverLogReceived = (LinkedList<String[]>)
PlatformUI.MIRTH_FRAME.mirthClient.invokePluginMethodAsync(SERVER_LOG_SERVICE_PLUGINPO
INT, GET_SERVER_LOGS, null);
} catch (ClientException e) {
```

3.4.0 and later:

ServerLogClient.java

```
try {
    serverLogReceived =
PlatformUI.MIRTH_FRAME.mirthClient.getServlet(ServerLogServletInterface.class).getServ
erLogs();
} catch (ClientException e) {
```

The servlet interface must also be specified in the plugin's plugin.xml file:

```
<pluginMetaData path="serverlog">
 ...
 <apiProvider type="SERVLET_INTERFACE"
name="com.mirth.connect.plugins.serverlog.ServerLogServletInterface"/>
 ...
</pluginMetaData>
```

3.5.0

Mirth Connect 3.5 and above now require a minimum version of Java Runtime Environment (JRE) 8. Please ensure that you have Java 8 (or later) installed first before attempting to upgrade.

Due to the recent SWEET32 vulnerability, we've disabled cipher suites using Triple DES. If for any reason you need to use one of these cipher suites to communicate with a legacy system, you can re-enable it in mirth.properties with the "https.ciphersuites" setting. If you're using the SSL Manager extension, then you can choose to enable these cipher suites on a specific channel / connector rather than for the entire server. If you have any connectors using the SSL Manager that arenot using the server default cipher suites, then you will want to update these connectors and uncheck any cipher suites containing "3DES".

To address the secondary "nation state resources" theoretical use-case of the Logjam vulnerability, in 3.5 we're now setting the ephemeral Diffie-Hellman key size to 2048. If for any reason you need to change this, you can do so in mirth.properties with the "https.ephemeraldhkeysize" setting.

Custom attachment handlers that extend MirthAttachmentHandlerProvider will need to provide a constructor that matches and calls this super constructor:

```
public MirthAttachmentHandlerProvider(MessageController messageController)
```

Custom attachment handlers also **may** need to update method signatures if any of the following methods have been overridden (updated signatures on the second line of each group):

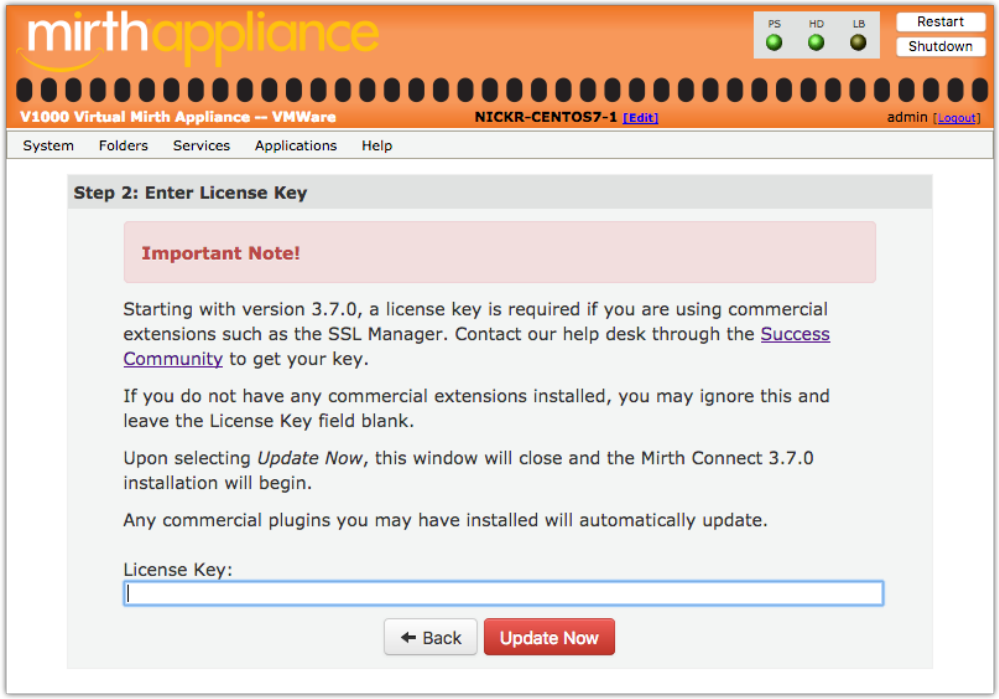
```
public byte[] reAttachMessage(String raw, ConnectorMessage connectorMessage, String
charsetEncoding, boolean binary)
public byte[] reAttachMessage(String raw, ConnectorMessage connectorMessage, String
charsetEncoding, boolean binary, boolean reattach)

public String reAttachMessage(ConnectorMessage message)
public String reAttachMessage(ConnectorMessage message, boolean reattach)

public String reAttachMessage(ImmutableConnectorMessage message)
public String reAttachMessage(ImmutableConnectorMessage message, boolean reattach)

public String reAttachMessage(String raw, ConnectorMessage message)
public String reAttachMessage(String raw, ConnectorMessage message, boolean reattach)

public String reAttachMessage(String raw, ImmutableConnectorMessage message)
public String reAttachMessage(String raw, ImmutableConnectorMessage message, boolean
reattach)

public byte[] reAttachMessage(String raw, ImmutableConnectorMessage connectorMessage,
String charsetEncoding, boolean binary)
public byte[] reAttachMessage(String raw, ImmutableConnectorMessage connectorMessage,
String charsetEncoding, boolean binary, boolean reattach)
```

3.7.0

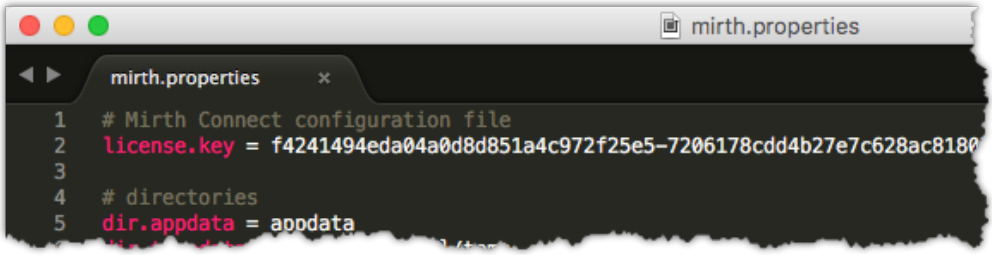
Commercial License Key

> As of 3.7 commercial extensions now require a license key to be set in mirth.properties. The machine running Connect also **must have outbound internet access** so that our licensing server can be reached. Before upgrading, contact the help desk through our Success Community to get your license key.

- **Using Mirth Appliance**
  - When installing the 3.7.0 update, a wizard will prompt you to enter the license key:



- **Standalone Installation**
  - Edit conf/mirth.properties.
  - Add a setting with key "**license.key**" and your license key for the value.



  - Restart the Connect service.

Database Connection Pools

In previous versions, there was a connection pool dedicated to the Donkey engine (message processing), and a separate pool dedicated to everything else, like Administrator or API actions.

In 3.7, this has been changed somewhat. You have two options:

- **Option 1: Use two pools, one read/write and one read-only**
  - This is the default in 3.7. To enable, set this in mirth.properties:
    - **database.enable-read-write-split = true**
  - When this setting is enabled, the database connection pool is split into two: A **read-only** pool, and a **read/write** pool. The read-only pool is used for many of the Administrator API calls that only fetch data. The read/write pool is used for all backend message processing, as well as any operation that creates/modifies/deletes data.

When the connection pools are split, you can separately configure settings for the read-only pool, with the "**database-readonly**" options. By default none of the "database-readonly" options are set, meaning that the read-only pool will default to the same configuration as the main read/write pool.

For example, if "**database-readonly.max-connections**" is not set, it defaults to the "**database.max-connections**" setting. So if you have your max connections set to 20 for the read/write pool, the read-only pool will also have up to 20 connections, meaning the total number database connections will be at most 40.

These settings also allow you to point the read-only connection pool to a completely different database instance. You may want to do this if you have a master DB and a horizontally scaling cluster of read-replica DBs. You can point the main read/write pool to the master DB, and point the read-only pool to the read-replica cluster instead. By doing this you can potentially reduce the traffic and strain on your master database.

When using read replicas however, the concept of "replica lag" should be taken into account. That refers to the amount of time a read replica DB is behind the master DB. If your replica lag is sufficiently large, all the selects done by the read-only pool may return out-of-date information, which can lead to unintended results in the Administrator.

The server keeps internal caches for channels, channel groups, code templates, and code template libraries. By default these caches use the read-only pool. But if replica lag is a concern, a separate option, "**database.write-pool-cache**", allows you to switch the caches over to using the read/write pool instead.

- **Option 2: Use one connection pool for everything**
  - To combine all database connections used by Connect into a single pool, set this in mirth.properties:
    - **database.enable-read-write-split = false**
  - All of the "**database-readonly**" options will be ignored in this case. If you switch to this mode, note that to may want to increase "**database.max-connections**" to compensate for the "**database-readonly.max-connections**" value that will no longer be used.

Keystore Passwords

The default keystore storepass and keypass is "81uWxplDtB". However when Connect starts up for the first time and the keystore file hasn't yet been created, Connect will automatically replace the default passwords with new auto-generated passwords.

If you are upgrading from a previous version nothing will change, your keystore and passwords will remain the same. This is only done for completely new servers where the keystore hasn't yet been created and the passwords are still set to the default values.

3.8.0

In 3.8 we upgraded the MySQL JDBC driver to version 8.0.16, so that the latest MySQL 8.x can be used. However the new driver doesn't support versions of MySQL older than 5.6. Since these older versions are over 10 years old and are already listed as End-Of-Life by Oracle, we've made the decision to adjust our system requirements to reflect that 5.6+ is now the minimum supported version for MySQL.

If you still need to connect to an older version of MySQL, don't worry you still can!

Connecting via a Database Reader/Writer or in JavaScript

- Download the appropriate MySQL JDBC Driver JAR for the version of MySQL you need to connect to.
- Place this JAR into a folder of your choice on the system running Mirth Connect.
- Create a new library resource (Settings -> Resources tab) pointing to that folder.
- Link your connector/channel to your new resource (Summary tab -> Set Dependencies).
- Make sure you're using the **old** driver class name, "com.mysql.jdbc.Driver".

Using an old version of MySQL for your Connect backend database

- Download the appropriate MySQL JDBC Driver JAR for the version of MySQL you need to connect to.
- Swap this JAR with the MySQL 8.x driver in the "server-lib/database" folder in your installation directory.
- Set "database.driver = com.mysql.jdbc.Driver" in mirth.properties.
- Make sure that any Database Reader/Writer connectors that are connecting to the old version of MySQL use the **old** driver class name, "com.mysql.jdbc.Driver".

If you need to use an old version of MySQL for the backend database and also need to connect to newer MySQL 8.x+ databases as well, you can follow the steps in the section above to include the 8.x JDBC driver as a custom resource.

3.9.0

SMB Versions in a File Reader/Writer

We added support for SMB versions 2 and 3 when using a File Reader/Writer. With "smb" as the selected method, click the wrench icon to view the SMB settings. There you can select the minimum and maximum supported SMB versions.

When upgrading Connect, existing channels will use "SMB v1" as the minimum version and "SMB v3.1.1" as the maximum version, in order to preserve functionality. However, SMB v1 is outdated and poses security risks, so we recommend updating your channels to use later versions of SMB. This may require also updating your SMB servers accordingly.

DICOM Sender Storage Commitment

In 3.9.0 we changed the functionality of the DICOM Sender option "Request Storage Commitment". Before 3.9.0, if a DICOM Sender with this option set to "Yes" failed to get a response to its storage commitment request, the sender would log an error to the server log but the message would still be considered successful. Now, in 3.9.0, if a DICOM Sender with this option set to "Yes" fails to get a response to its storage commitment request, the message will get the Error state. This means there's a potential for DICOM Senders to start encountering errored messages after the update but we feel that this is the proper behavior. You must either ensure the system you are sending to supports storage commitment or you can set "Request Storage Commitment" to "No".

3.10.0

Advanced Clustering Sync Intervals

In 3.10.0 we've split many of the clustering tasks off from the Cluster Sync Interval into their own intervals. Here's a list of the new intervals and what they control:

- Dashboard Status Sync Interval: The frequency in milliseconds to query the cluster to update channel dashboard statuses.
- Cluster Log Sync Interval: The frequency in milliseconds to query the cluster to update alert logs, connection logs, and server logs.
- Cluster State Sync Interval: The frequency in milliseconds to query the cluster to update alert states, connection statuses, and the global map.

Cluster Sync Interval continues to handle the frequency in milliseconds to query the cluster for new tasks to execute. These tasks handle membership changes, message recovery, role changes, and server name changes.