Best Practices: Upgrade Process

IdentityIQ Version: 7.2

This document describes SailPoint's recommended best practices for upgrading from one version of IdentityIQ to the next (or any subsequent) version.

Document Revision History

Revision Date	Written/Edited By	Comments
Sep 2017	Jennifer Mitchell	Split out from Deployment/Migration guide; updated for 7.2

Table of Contents

Introduction 4

Managing Upgrades 4

Upgrading Across Multiple Versions 4

Sandbox Upgrade 5

Dev-Int, UAT, and Production Upgrades 9

Appendix A: Sample Upgrade Procedure 13

ACME IdentityIQ Upgrade Procedures 13

Overview 13

Steps 13

Appendix B: Validating the Upgrade 15

Validation Steps 15

Appendix C: Troubleshooting 17

Appendix D: Upgrade Frequently Asked Questions 19

Introduction

This document explains how to apply an upgrade release (patch or new GA version) to an IdentityIQ installation using the recommended infrastructure described in the Deployment/Migration Best Practices guide. Namely, this is a multi-environment configuration with a version control system and a structured build process, where development occurs in the first environment level, and deployment of the developed artifacts to subsequent environments for testing and production use occurs through a structured, repeatable build process.

Managing Upgrades

An upgrade, whether a patch upgrade or an upgrade to a new major release, should follow the same migration procedures as the initial installation testing/migration process. Namely, the upgraded IdentityIQ application version and any custom configurations for the installation should be deployed and tested first in the sandbox environment, then in Dev-Int, and then in UAT; it should only be deployed to production when it has been successfully deployed to, and tested in, UAT. When this procedure is followed, the deployment to production occurs only after multiple successful deployments in less sensitive environments, minimizing the risk of problems in the production deployment that could result in production downtime or data errors.

NOTE: In addition to the instructions provided in this guide, always read the **Important Upgrade Considerations** section in the Release Notes for each major release and the **Release Notes** section of the README file included with any patch release. These discuss important information specific to the given release.

Upgrades are comprised of the following basic steps:

1. Backup the existing installation and data
2. Unpack the upgrade files
3. Run any database upgrade scripts
4. Execute the iiq upgrade or patch command
5. Evaluate the impact of changes made by the upgrade, merge any customizations with new files, and redeploy

Additional details on how to execute these basic steps in each environment are provided in the environment-specific upgrade sections that follow.

Upgrading Across Multiple Versions

Customers do not always upgrade to every release of IdentityIQ, so when it comes time to upgrade, the process may involve upgrading through several versions. The *Upgrade Path Through IdentityIQ Versions* document on Compass shows a matrix of all of the IdentityIQ releases and the upgrade paths available for them so customers can clearly see which versions to install when progressing from one release to any later release.

All interim GA releases between the currently installed release and the target version must be installed. However, it is not necessary to upgrade to the latest patch release of the currently installed version before upgrading to the next GA version, and it is not necessary to apply any patches on interim GA releases before upgrading to the next GA release. For example, a customer upgrading from 6.2p7 to 7.1p2 must run the 6.3GA upgrade, the 6.4GA upgrade, the 7.0GA upgrade, the 7.1GA upgrade, and the 7.1p2 patch upgrade. No further 6.2 patches and no 6.3, 6.4, or 7.0 patches are required. Since patches are cumulative, the 7.1p1 patch can be skipped as well.

Full testing of each interim release upgrade between the current version and the final target version is neither necessary nor recommended. Testing the full impact of each interim release and making changes to rules, workflows, etc. for each of those releases could result in unnecessary throw-away work if a subsequent release changes the requirements for those objects yet again. The recommended best practice is to complete the full set of upgrades before restarting the application and conducting any tests, especially in the production environment. In production, the upgrade process should only be stopped when a failure occurs. In pre-production environments, some customers choose to restart the application and verify that IdentityIQ is up and available after each interim upgrade is applied, and they occasionally do some minimal spot-checking of artifacts as well. However, test aggregations, certifications, provisioning requests, etc. are not recommended in any environment until the final target version's upgrade has been applied.

Each sequential environment should be upgraded through all interim versions to the target version before progressing to the next environment. In the example scenario of upgrading from 6.2p7 to 7.1p2, this means that the sandbox environment should first be updated to 6.3, 6.4, 7.0, and then 7.1 and 7.1p2, before repeating the process in the Dev-Int environment, then UAT, and finally the Production environment. Full testing should occur in each environment only during the final (7.1p2) upgrade process, and artifact migration to each sequential environment should only occur following successful test results in the preceding environment.

Sandbox Upgrade

The deployment to the sandbox environment is generally the most complex and time-intensive because the implications of changes to the data model and core product components should be discovered and managed in this environment. Upgrades, especially major (GA) release upgrades, may introduce changes to object structures that require updates to the managed XML object versions in the source code repository. Additionally, changes to file system components or system functionality may render custom classes or UI customizations obsolete or may otherwise require changes to those components. Evaluating the impact of and responding to the various changes in the new release may be a laborious manual and investigative task, depending on the extent and types of customizations implemented for the installation. However, once this is done in the sandbox environment, deployment of the modified managed artifacts to other environments is a straightforward, scriptable process.

The recommended best practice procedures for upgrading the sandbox environment and preparing the artifacts for deployment to subsequent environments are described below.

1) Before applying any upgrade, stop all processes and shut down IdentityIQ. This is usually done by stopping the application server but could also be accomplished by shutting down the application from the application server's administrative interface.

NOTE: The purpose of the next four steps is to delete the old contents of the sandbox environment and prepare it with a clean copy of the production system version and all managed artifacts for the system; this ensures that the upgrade is applied to the correct system version and that it updates all managed artifacts for the installation. When upgrading across multiple releases, these steps are performed only once, prior to deployment of the first upgrade version.

2) **OPTIONAL**: If desired, back up the sandbox database and zip a copy of the IdentityIQ installation directory tree (the deployment directory, e.g.: /tomcat/identityiq/*). Because old data in a sandbox environment is usually considered throw-away, it is often *not* backed up before beginning an upgrade deployment. A backup of the new clean deployment will be captured in step 6 (below) before running any upgrade processes on the system, so most customers choose to skip this preliminary backup.
3) Certain files in the installation directory tree provide important configuration settings for communicating with the IdentityIQ database and for configuring the table columns as required for the installation. These will be overwritten on upgrade to a new GA version and possibly on patch upgrades as well, so they must be merged back into each version after the upgrade war/jar is deployed. Copy these files to a temporary location outside of the installation directory tree to save them for merging in later:
   - WEB-INF/classes/iiq.properties
   - WEB-INF/classes/sailpoint/object/*Extended.hbm.xml
   - JDBC drivers in WEB-INF/lib (The JDBC drivers that ship with IdentityIQ may be older than the current database version in use, and SailPoint strongly recommends replacing those drivers with the latest ones available from the customer's database vendor.)

   These files should also be saved in the **web** directory of the build structure and in the VCS so they are redeployed automatically when the build is used to deploy the application, but they will have to be manually applied when the upgrade versions are deployed without the build as part of the upgrade process. The iiq.properties file version stored in the build usually contains target substitution variables to allow different values to be set automatically by the build process for each environment.
4) Delete all contents of the IdentityIQ installation directory tree in the sandbox environment to prevent the possibility of previous versions' jar files, previously-applied customizations, or efixes interfering with the upgraded system.
5) Build and deploy a clean copy of the current production IdentityIQ version in the sandbox environment like this:
   a. Checkout the build environment from the version control system. This should include the current IdentityIQ version deployed in production, any applied patches or efixes, and up-to-date copies of all managed artifacts for the installation.
   b. Perform a clean build of the current system version with the managed artifacts and deploy it to the sandbox environment.
   c. Drop the database and recreate it using the Create scripts in the installation's WEB-INF/database directory. This deletes all artifacts in the system, providing a clean slate for the upgrade version. **NOTE**: If extended attributes have been added to the *Extended.hbm.xml files, the correct Create scripts must be generated by running the `iiq schema` command; the default scripts create only the standard number of extended attributes for each object type. (The `iiq schema` command is described in more detail in the IdentityIQ Installation Guide included with any GA release.)

d. From the IdentityIQ console, import the init.xml file to add the core product objects into the database.
e. If Lifecycle Manager (LCM) is licensed and installed for the installation, import init-lcm.xml as well.
f. Import the full set of managed XML objects into the database using the sp.init-custom.xml script generated by the build process (found in the [IdentityIQ Installation Directory]/WEB-INF/config directory). In most cases, the build process edits the init.xml file to also run this script, but if any of the managed artifacts are overrides of LCM objects, they must be re-imported after running init-lcm.xml to supersede the default objects imported by that script. This ensures that all managed XML objects (including customizations of the core product objects) are in the pre-upgrade environment so the upgrade process can apply any object structure changes to them all.

NOTE: When upgrading across multiple releases, the next 3 steps (steps 6-8) are performed iteratively for all interim releases and the final target release before proceeding to the final steps in this upgrade process.

6) Back up the database and zip a copy of the IdentityIQ installation directory tree; this provides a clean rollback snapshot if errors are encountered in the upgrade process.
7) Deploy the upgrade version to your sandbox file system like this (Do not use the build process for this step.):
   a. For a GA upgrade, delete the entire contents of the IdentityIQ installation directory tree. Then download the war file and unjar it in the installation directory.
   b. For a patch upgrade, download the jar file and unjar it in the installation directory, overlaying it into the GA installation. If any efixes are installed, delete the efix from the WEB-INF/classes directory tree. (Efixes are built for a specific version and do not apply when a new version is installed.)
   c. Copy the configuration files saved in step 3 above back into the installation:
      - iiq.properties -> WEB-INF/classes
      - *Extended.hbm.xml -> WEB-INF/classes/sailpoint/object
      - JDBC drivers -> WEB-INF/lib

      NOTE: If any JDBC drivers are replaced, the corresponding RDBMS drivers that are included with IdentityIQ must be deleted.

   NOTE: This deployment will not recompile any of the custom java classes, and the GA upgrade process will not include any custom file system files (xhtml, css, etc.) for the installation. This is okay because the only purpose of this deployment is to prepare to upgrade the XML objects in the database. The entire environment will be rebuilt and deployed with all of the custom artifacts later, in step 10.
8) Run the SQL upgrade scripts (DDL) and the iiq upgrade or iiq patch command, as described in the upgrade chapter of in the installation guide included with any GA release or in any patch version's readme file. These upgrade processes make any required structural modifications to the installation's managed XML artifacts (as well as any other system objects). If the post_upgrade_identityiq_tables DDL script has any contents, run that script after the iiq upgrade/patch command has been run to complete the post-upgrade database clean up tasks.

   NOTE: The DDLs are templates that may require editing per installation. Each customer's DBA should examine the DDLs and modify them as required before running them. Common modifications include changes to the database name, indexes, column lengths, etc.

NOTE: An upgrade can take minutes to hours and can generate many messages to stdout. Ensure that it is run from a terminal that is configured so the process will continue to run even if the terminal's connection to the server is lost. The upgrade processes for later versions of IdentityIQ (6.0+) automatically log the stdout messages to a log file, which can be reviewed for errors once the upgrade process is finished. Earlier versions may need their upgrade output redirected to a file to capture this same data for review.

NOTE: When upgrading across multiple releases, repeat steps 6-8 for each interim release and the final target release before proceed to step 9. If desired, the application server can be restarted to verify that IdentityIQ can be accessed between each upgrade, but full testing should only be conducted on the final target version (as described in step 11 below). See the *Upgrading Across Multiple Versions* section above for details.

9) When the final target version upgrade has been performed, export copies of the managed XML artifacts from the upgraded system and copy them into the **config** folder of the build environment structure. This ensures that all object modifications made by the upgrade process are reflected in the build's set of managed XML artifacts.
   NOTE: Objects managed through the change-merge process (e.g. Configuration, UIConfig) will be exported in their entirety (i.e. as the whole XML object) but should not be saved into the build structure that way. Instead, the entries in the merge import XML files should be manually compared to the corresponding entries in the new core configuration objects, and the merge import files should be updated to reflect any required changes made by the upgrade process. The primary reason for managing these artifacts this way is for supportability; it is much easier to identify what has been customized in these artifacts when the import-merge XML is maintained, which speeds the troubleshooting process with SailPoint Support when problems occur and makes future modifications easier to manage as well.
10) Use the build process to build and deploy the final target upgrade version in the sandbox environment as described below. This recompiles any custom java classes against the target core product version, includes any custom file system artifacts (css, xhtml, etc.), and redeploys the managed XML artifacts.
    a. Start with the build environment structure used in previous steps.
    b. When just upgrading to a new patch version of the currently deployed GA version, copy the patch's jar file into the base/patch directory of the build directory structure.
    c. When upgrading to a new GA version, copy the target version's zip file into the base/ga directory of the build directory structure.
    d. When upgrading to a patch version of a new GA version, copy the target GA version's zip file into the base/ga directory and the target patch's jar file into the base/patch directory of the build directory structure.
    e. Remove any efixes from the /base/efix directory of the build directory structure. Efixes are built for a specific version and must be deleted when a new version is installed so they do not override other functional improvements made in future releases.
    f. Assess whether any of the custom Java classes should be removed or further modified and whether any file system managed artifacts need to be edited. This is a manual, investigative task that

includes reading the release documentation as well as examining the relevant system components in the new version to determine if new product features render custom content obsolete.

g. If core file system files have been modified, save a copy of the new core files before deploying with the customized old file versions. This makes it possible to restore the core files and reapply the customizations to those new file versions if the old versions cause errors.
h. Build a war file for the target upgrade version and deploy it to the sandbox environment. Custom java files may sometimes incur errors during recompilation against the new version (performed as part of the build process) due to API changes. Correct these errors so those classes successfully compile before deploying the upgraded version.
i. XML objects do not need to be re-imported into the sandbox database since they are already in the database in their upgraded state.

11) Restart the application (or application server) and fully test the upgraded installation in the sandbox environment. Any required modifications of file system artifacts or custom java files discovered during the testing process should be applied in the build structure and rebuilt/redeployed. Any modifications to managed XML objects can be made directly in the IdentityIQ installation, but ensure that they are re-exported and captured in the build directory structure when the system has been fully tested and verified.
12) Commit the updated build folder structure to the version control system.

NOTE: See the troubleshooting tips in *Appendix C* if any problems are encountered in the upgrade process, and refer to the FAQ in *Appendix D* for additional information that might be helpful when executing an upgrade.

Dev-Int, UAT, and Production Upgrades

The sandbox upgrade process leaves the build environment ready to produce a deployable WAR file that includes fully-updated copies of the managed XML objects and tested versions of any custom classes or file system artifacts. In all subsequent environments, the upgrade process should be managed through a scripted set of steps, with the same set of steps being performed in each environment sequentially. Those steps still include running the DDL scripts and upgrade process for each version on the upgrade path in each environment, but no additional artifact management is required after the sandbox upgrade. Since the production upgrade should only occur after the upgrade has been successfully deployed in other, less sensitive environments, this iterative, scripted approach minimizes the risk of problems occurring in the production upgrade.

The recommended best practice is to take a backup of the production system’s database and installation directory and restore them in one or more of the migration environments before starting the upgrade procedures in those environments. This serves as verification that the production backups are usable for recovery while also setting up a test bed that fully represents the data volumes and types in the production environment. When a test environment matches production, the scripted upgrade process can be fully validated and the time duration required for the production upgrade can be determined in advance of running the production upgrade procedure. This can be critical for organizations operating in a tight time window for production upgrades.

These steps should first be performed in the Dev-Int environment, then in UAT when the upgrade has been fully tested and approved in Dev-Int, and finally in Production with the UAT upgrade has been tested and approved.

NOTE: These steps are high-level guidelines for the upgrade process that should be used as a template as each customer develops and documents their specific upgrade procedures based on their own environment. An example customer-specific upgrade procedure is shown in *Appendix A*, illustrating how these steps should be translated into a script for a specific installation's upgrade.

1) Stop all processes and ensure that all scheduled tasks have been deleted. This is usually done by stopping the application server but could also be accomplished by shutting down the application from the application server's administrative interface.

NOTE: When upgrading across multiple releases, the next 3 steps are conducted iteratively for all interim releases and the final target release before proceeding to the final steps in this upgrade process.

2) Back up the environment's database and zip a copy of the IdentityIQ installation directory tree in the environment; this provides a clean rollback snapshot if errors are encountered in the upgrade process. **NOTE**: Many customers choose to do this step only once in a multi-release upgrade – before running the first upgrade. Some customers choose to take a backup after each interim upgrade as a rollback safeguard against errors in a later version's upgrade process; without interim backups, an unrecoverable error in a later version's upgrade would require starting the upgrade process over at the beginning.
3) As described in the sandbox upgrade steps, these files in the installation directory tree must be saved to a temporary location outside of the installation directory tree for merging back in following a native (non-build) deployment of an upgrade version:
    - WEB-INF/classes/iiq.properties
    - WEB-INF/classes/sailpoint/object/*Extended.hbm.xml
    - JDBC drivers from WEB-INF/lib

    These files are also saved in the web directory of the build structure and in the VCS so they are redeployed automatically when the build is used to deploy the application, but they will have to be manually applied when the upgrade versions are deployed without the build as part of the upgrade process.
4) Deploy the upgrade IdentityIQ version to the IdentityIQ installation directory in the file system following these procedures. (Do not use the build process for this step.)
    a. For a GA upgrade, delete the entire contents of the IdentityIQ installation directory tree. Then download the war file and unjar it in the installation directory. (If the application server requires the application to be deployed as a war file, run the upgrade from a temporary directory; only the new war file built in step 7 will replace the old deployed war file.)
    b. For a patch upgrade, download the jar file and unjar it in the installation directory, overlaying it into the GA installation (or follow the patch upgrade instructions to update a war file deployment). If any efixes are installed, delete them from the WEB-INF/classes directory tree.
    c. Copy the configuration files saved in step 3 above back into the installation:
        - iiq.properties -> WEB-INF/classes
        - *Extended.hbm.xml -> WEB-INF/classes/sailpoint/object
        - JDBC drivers -> WEB-INF/lib

NOTE: If any JDBC drivers are copied into WEB-INF/lib, the corresponding RDBMS drivers that are included with IdentityIQ must be deleted.

NOTE: The purpose of this deployment is to prepare to upgrade the database structure and the XML objects in the database. It will not recompile any of the custom java classes, and a GA upgrade deployment will not even include any custom classes or file system artifacts (e.g. xhtml, css files) for the installation. This is okay because the already-upgraded artifacts from the sandbox environment will be included in the environment in the rebuild and redeploy in step 7.

5) Run the SQL upgrade scripts (DDL) and the iiq upgrade or iiq patch command, as described in the upgrade chapter of in the installation guide included with any GA release or in any patch version's readme file. These upgrade processes make any required structural modifications to the installation's managed XML artifacts (as well as any other system objects). If the post_upgrade_identityiq_tables DDL script has any contents, run that script after the iiq upgrade/patch command has been run to complete the post-upgrade database clean up tasks.
   NOTE: The DDLs are templates that may require editing per installation. Each customer's DBAs should examine the DDLs and modify them as required before running them. Common modifications include changes to the database name and indexes.
   NOTE: An upgrade can take minutes to hours and can generate many messages to stdout. Ensure that it is run from a terminal that is configured so the process will continue to run even if the terminal's connection to the server is lost. The upgrade processes for later versions of IdentityIQ (6.0+) automatically log the stdout messages to a log file, which can be reviewed for errors once the upgrade process is finished. Earlier versions may need their upgrade output redirected to a file to capture this same data for review.

NOTE: When upgrading across multiple releases, repeat steps 2-5 for each interim release and the final target release before proceeding to step 6. If desired, the application server can be restarted in pre-production environments to verify that IdentityIQ can be accessed between each upgrade, but full testing should only be conducted on the final target version (as described in step 10 below). The full upgrade process should be conducted without interruption in the production environment. See the *Upgrading Across Multiple Versions* section above for details.

6) Delete all contents of the IdentityIQ installation directory tree in the target environment to prevent the possibility of previously-applied customizations or efixes interfering with the upgraded system. To start with a clean environment for testing, drop and recreate the database; this is often done in Dev-Int but rarely in UAT and never in production.
   NOTE: If the database is dropped and recreated, customers who have added extended attributes to the *Extended.hbm.xml files must run the `iiq schema` command to generate updated create scripts, and those scripts should be used instead of the default create scripts to recreate the database. (`The iiq schema` command is described in more detail in the IdentityIQ Installation Guide included with any GA release.)
7) Using the updated build environment, build and deploy the upgraded system version to the target environment. This may require some changes to the build properties files – new target specification, different token substitutions, etc.; see the SSB User Guide for details on those properties.

8) If the database was dropped and recreated in step 6, import the init.xml and init-lcm.xml (if LCM is licensed) files through the IdentityIQ console.
9) Import the full set of managed XML objects into the database using the sp.init-custom.xml script generated by the build process (found in the [IdentityIQ Installation Directory]/WEB-INF/config directory). These objects were upgraded to the target version in the sandbox environment, and any structural changes to the database or system required to support them will have been made by the SQL upgrade scripts and upgrade/patch commands.
10) Restart IdentityIQ (or the application server) and test the upgraded installation in the environment. Follow appropriate testing/acceptance processes for each environment, as defined by the organization's policies and software development lifecycle. Any problems encountered may require a return to the sandbox environment for development work (depending on the organization's policies) along with rebuild/redeployment of the updated version. Ensure that any modifications to managed artifacts are captured in the build directory structure and the version control system.

Appendix A: Sample Upgrade Procedure

Every customer should document their own upgrade procedure with a detailed, step-by-step plan. The plan should include:

- Version details – note the specific set of versions to be installed on the upgrade path
- Managed artifact types – identify whether there are any custom classes or file-system artifacts to be managed; all installations will have some managed XML artifacts
- Ordered steps for each environment, based on the general procedures outlined in this document

The example below is for a fictional company, ACME, upgrading from IdentityIQ version 6.4p5 to version 7.1p2. This is the upgrade script for the Dev-Int, UAT, and Production environments. It assumes the Sandbox upgrade has already occurred and all managed artifacts have therefore been upgraded and tested in that environment.

ACME IdentityIQ Upgrade Procedures

Overview

Current Version: 6.4p5
Target Version: 7.1p2
Required Upgrade Path: 7.0GA -> 7.1GA -> 7.1p2

Managed Artifacts

- XML Artifacts: yes
- Custom Java Classes: no
- File System Artifacts: yes

Steps

These steps must be performed in each sequential environment (Dev-Int, then UAT, then Production).

1. Download identityiq-7.0.zip, identityiq-7.1.zip, and identityiq-7.1p2.jar from Compass.
2. Stop the IdentityIQ application.
3. Take a backup of the DB and jar the IdentityIQ installation directory.
4. Copy WEB-INF/classes/iiq.properties to /tmp.
5. Delete all from /tomcat/webapps/identityiq directory. Unzip identityiq-7.0.zip and copy identityiq.war to /tomcat/webapps/identityiq. jar –xvf identityiq.war
6. Copy /tmp/iiq.properties to WEB-INF/classes.
7. Run upgrade DDL: log into mysql; source upgrade_identityiq_tables.mysql.
8. In WEB-INF/bin directory, chmod +x iiq. Run ./iiq upgrade 2>&1 | tee upgrade64-70.out..
9. Delete all from /tomcat/webapps/identityiq directory. Unzip identityiq-7.1.zip and copy identityiq.war to /tomcat/webapps/identityiq. jar –xvf identityiq.war
10. Copy /tmp/iiq.properties to WEB-INF/classes.
11. Run upgrade DDL: log into mysql; source upgrade_identityiq_tables.mysql.
12. In WEB-INF/bin directory, chmod +x iiq. Run ./iiq upgrade 2>&1 | tee upgrade70-71.out.
13. Copy identityiq-7.1p2.jar to /tomcat/webapps/identityiq. jar –xvf identityiq-7.1p2.jar.
14. Copy /tmp/iiq.properties to WEB-INF/classes.
15. Run upgrade DDL: log into mysql; source upgrade_identityiq_tables.mysql.

16. In WEB-INF/bin directory, run ./iiq patch 2>&1 | tee patch71p2.out.
17. Build acme-iiq71p2.war file from the SSB environment, pointing servers.properties to [Environment]. (This assumes [environment].iiq.properties exists with correct properties and that [environment].target.properties contains the appropriate token properties for token substitutions. It also assumes that the 7.1GA zip and the 7.1p2 jar file were dropped into the build in the sandbox upgrade preparation process.)
18. Delete all from /tomcat/webapps/identityiq directory. Copy acme-iiq71p2.war into /tomcat/webapps/identityiq and jar-xvf acme-iiq71p2.war.
19. Copy /tmp/iiq.properties to WEB-INF/classes.
20. Launch the console (from WEB-INF/bin: ./iiq console). Import sp.init-custom.xml. (This reloads all upgraded managed XML objects.)
21. Restart the application.
22. Perform all prescribed test cases (according to documented test plan).

Appendix B: Validating the Upgrade

This section describes some suggested ways for validating that the upgrade of a pre-production (e.g. DEV-INT or UAT) environment has been successful before performing the production upgrade. These validation steps assume the customer has followed the recommended best practice and completed these key steps:

- Restored a production backup to the pre-production environment prior to applying the upgrade
- Followed the upgrade steps outlined in the Managing Upgrades: Dev-Int, UAT, and Production Upgrades section of this document to apply the upgrade to the pre-production environment

These steps should only be performed on the final target version, not on interim version along the upgrade path.

Validation Steps

1. Before performing any test cases, configure the pre-production environment to redirect emails to a file to avoid accidentally sending emails during testing. If provisioning (other than manual work items) has been implemented, disable provisioning (perhaps using the Simulated Provisioning Connector available on Compass) before proceeding with active testing in Step 3 below.
2. To verify all configuration artifacts are present, run the iiq console "list" command on all configuration object types in both the production and pre-production environments and confirm that the same objects are present in both environments.
3. Validate that any differences between the configuration artifacts in the two environments are expected and justified. Export all configuration objects in both the production and pre-production environments using the iiq console "export –clean" command (or use the objects in the build structure for the pre-production environment, since they were freshly imported as part of the upgrade process). Compare the objects from both environments using a diff utility to identify any differences, and confirm that the differences are appropriate (e.g. part of new functionality in the new release.)
4. Test the functionality of the configuration artifacts, as suggested below, to verify that they work the same in both environments.
    - **Aggregation**: Run a complete aggregation/refresh cycle in both the production and pre-production environments at the same day/time (or as close as possible) from the same production data so that the data aggregated should be identical. Verify that the access aggregated in both systems is the same by running the Identity Effective Access report in both environments, exporting them as CSV files, and using a diff utility to compare the two versions; they should be identical or their differences should be explainable by a difference in timing of the aggregation runs.
    - **Entitlement Catalog** (if used): Run the Export command from the Application -> Entitlement Catalog window from both environments. Use a diff utility to compare the 2 CSV files and confirm that they are identical or that any differences can be explained by a difference in timing of the aggregation runs. (Only applies when both the original version and the target version include the Entitlement Catalog.)
    - **Custom reports** (if used): Run any custom reports in both environments and export their results as CSV files. Use a diff utility to compare the 2 CSV files and confirm that they are identical or any differences can be explained by a difference in timing of the aggregation runs.

- **LCM Workflows and Lifecycle Event Triggers** (if used): Run through a set of regression tests for LCM workflows and event triggers (e.g., onboarding, termination, request access) *in the pre-production environment only* to confirm that they operate correctly. These regression tests should be a subset of those performed during initial implementation. This should be done last as it will change data in the pre-production environment and impact the comparisons.

NOTE: New functionality supported in a new version should not be implemented during the upgrade process unless it is required by the new version. For example, connectors previously used only for reading data should not be used to implement write functionality (provisioning) during an upgrade even if the connector gains that ability as a result of the upgrade. Functional changes should be implemented (through a separate change-control and testing cycle) only after the upgrade has been successfully completed in the production environment.

Appendix C: Troubleshooting

Customers may occasionally experience problems with the upgrade process caused by specific system customizations or by unexpected system failures during the upgrade process. This section suggests steps to take to resolve the issues.

Problem: The core system objects (e.g. non-managed objects or configuration objects) have not been correctly updated by the upgrade process.

Solution: Re-run the import of the init.xml and (if LCM is installed) init-lcm.xml files. This updates all the core objects to their default state for the installed version. Then, re-run the import of all managed XML artifacts to ensure that customizations are reapplied as intended for the installation.

Problem: **Lifecycle Manager (LCM) components were not upgraded in sync with compliance system.**

Solution: If LCM is enabled when the upgrade is run, this problem will not occur. On rare occasion, however, a customer may have initially deployed LCM and then disabled it. If an upgrade is run against a system with LCM disabled, the upgrade process will skip updating of any LCM components and the versions could get out of sync. Once LCM has been installed, it should always be enabled prior to running the upgrade process and can then be disabled again after the upgrade is complete, if desired. To enable LCM, set the "lcmEnabled" entry in the SystemConfiguration Configuration object to "true." Alternatively, LCM can be brought back in sync by importing the init-lcm.xml file for the installed version; this also automatically sets to lcmEnabled entry to true.

Problem: **System displays the error message: Schema version [x] does not match required version [y].**

Solution: This occurs when "iiq upgrade" or "iiq patch" is run before running the DDL script to update the database. Do not manually change the version information stored in the database; that would skip the important step of making the required database changes. Instead run the DDL, which will make the necessary changes and update the version information in the database, and then run the iiq upgrade or iiq patch command.

Problem: **System displays the error message: System version [x] does not match database version [y].**

Solution: This occurs when the application is restarted after the DDL scripts have been run but before the "iiq upgrade" or "iiq patch" command has been executed. To solve this problem, simply run the iiq upgrade or iiq patch command and then restart the application / application server.

Problem: **System displays the error message: Error creating bean with name 'versionChecker' defined in class path resource [configBeans.xml]: Initialization of bean failed; nested exception is java.lang.RuntimeException: Unable to check IdentityIQ database version: Cannot create PoolableConnectionFactory (Access denied for user 'identityiq'@'localhost' (using password: YES)).**

Solution: An invalid username/password combination has been provided in the /WEB-INF/classes/iiq.properties file. Correct the username and password in that file and try again.

Problem: **System displays the error message: Error creating bean with name 'versionChecker' defined in class path resource [configBeans.xml]: Initialization of bean failed; nested exception is java.lang.RuntimeException: Unable to check iiq database version: Cannot create PoolableConnectionFactory (Unknown database 'help1').**

Solution: An incorrect JDBC URL has been provided in the /WEB-INF/classes/iiq.properties file. Correct the database URL in that file and try again.

Problem: **System displays the error message: java.lang.OutOfMemoryError: Java heap space.**

Solution: Insufficient memory has been allocated to the iiq console for performing the upgrade. Allocate more memory to the console. A wiki article providing instructions for doing this can be found on Compass.

Problem: **The patch or upgrade command displays messages indicating that columns, tables, or indexes already exist or that key names are duplicates.**

Solution: This just means that a previous patch has been applied that already made the changes that the upgrade is trying to make. Since patches are cumulative and upgrades do not require all interim patches to be applied first, they are designed to be redundant. Consequently, these messages do not actually indicate a problem and can be ignored. Ensure that they system is configured to allow the script to continue running despite these messages so other changes made by the script can be applied successfully.

Problem: System displays the error message: ./iiq: Permission denied.

Solution: This means that the user does not have execute permission on the iiq application. This is only an issue for UNIX installations. From the /WEB-INF/bin directory, enter: chmod +x iiq to make the iiq application executable by all users.

Appendix D: Upgrade Frequently Asked Questions

This section addresses some common questions that may come up as customers prepare for an upgrade or follow the process outlined in this document.

Q: **When stopping the application server prior to performing an upgrade, do we need to ensure that all users are logged off and all tasks are completed first?**

A: Ideally, yes. The task executors and database should cleanly complete all currently running transactions prior to termination, thus preventing data corruption. However, if an aggregation task, for example, has processed half of an application's records, it will stop and leave the other half unaggregated; the task would have to be rerun to complete the full aggregation. An upgrade is best run at a time when users will not be accessing the system, and all task and request schedulers should be disabled while the upgrade is running.

Q: How do we disable a task or request scheduler?

A: In version 6.2+, the list of servers to use as task and request hosts should be managed through the Task and Request ServiceDefinition objects. By default, tasks and requests run on all hosts, as indicated by the attribute hosts="global" on those objects; tasks and requests can be directed to specific hosts by specifying a comma-separated list of host machine names in that hosts attribute. Change the hosts values in the Task and Request ServiceDefinition objects to any non-existent server name(s) to disable these schedulers.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE ServiceDefinition PUBLIC "sailpoint.dtd" "sailpoint.dtd">
<ServiceDefinition hosts="XXXX" name="Task"…>
  <Description>
Service definition for the Task scheduler service.
    </Description>
</ServiceDefinition>
```

In versions prior to 6.2, the iiq.properties file controls the list of task and request hosts, so disabling all task request hosts is done by modifying the iiq.properties file and pointing the taskSchedulerHosts and requestSchedulerHosts to a non-existent server name.

```
environment.taskSchedulerHosts=XXXX
environment.requestSchedulerHosts=XXXX
```

This is particularly important if you plan to restart the application server during the upgrade process to verify upgrade status along the way; if tasks are scheduled to run during the time when the upgrade is performed, the taskScheduler will attempt to run them when the server is restarted, which could cause unintended consequences. The requestScheduler will attempt to process any pending requests when the server is restarted. Preventing these two from running at all keeps this from being a problem without requiring you to delete (and later recreate) all scheduled tasks and pending requests.

NOTE: For backward compatibility, 6.2+ versions still support the iiq.properties method of specifying task and request schedulers. When the ServiceDefinition method is used with newer versions, which is the recommended approach, the environment.taskSchedulerHosts and environment.requestSchedulerHosts values should be omitted from the iiq.properties file to avoid confusion.