Asset Manager

Software Version: 9.61

Windows® and Linux® operating systems

Tuning

Document Release Date: September 2016

Software Release Date: September 2016

Hewlett Packard Enterprise

Legal Notices

Warranty

The only warranties for Hewlett Packard Enterprise products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from Hewlett Packard Enterprise required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

This product includes an interface of the 'zlib' general purpose compression library, which is Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to: https://softwaresupport.hpe.com/.

This site requires that you register for an HPE Passport and to sign in. To register for an HPE Passport ID, click **Register** on the HPE Software Support site or click **Create an Account** on the HPE Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HPE sales representative for details.

Support

Visit the HPE Software Support site at: https://softwaresupport.hpe.com.

This website provides contact information and details about the products, services, and support that HPE Software offers.

HPE Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support website to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HPE support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HPE Passport user and to sign in. Many also require a support contract. To register for an HPE Passport ID, click **Register** on the HPE Support site or click **Create an Account** on the HPE Passport login page.

To find more information about access levels, go to: https://softwaresupport.hpe.com/web/softwaresupport/access-levels.

HPE Software Solutions Now accesses the HPE Software Solution and Integration Portal website. This site enables you to explore HPE Product Solutions to meet your business needs, includes a full list of Integrations between HPE Products, as well as a listing of ITIL Processes. The URL for this website is http://h20230.www2.hp.com/sc/solutions/index.jsp.

Contents

Chapter 1: Introduction

What is the aim of this guide?

This guide describes tuning strategies for Asset Manager. In particular, it deals with certain techniques to reduce bottlenecks caused by:

- The network.
- The database server.
- The Asset Manager client.
- The database engine.

Who is this guide intended for?

This guide is mainly intended for Asset Manager administrators and database administrators. It contains information and procedures, which if misinterpreted, misapplied or implemented badly may provoke a range of problems from performance issues to data corruption.

> **Caution:** This document is intended to assist administrators in identifying and resolving performance-related issues within the software and database. However, it is not a comprehensive database-tuning guide, and although it is intended to contain as much information about tuning as possible, it is not guaranteed that the subjects mentioned herein are the only tuning options available. Also, HPE cannot be held liable for any damages or data loss incurred as a result of improper use of the information provided herein. If you do not fully understand a procedure or how a procedure will impact your installation or information system, you are advised not to execute the procedure. As always, HPE recommends that you perform a complete backup of your system and installation before making any major changes to it.

Reminder concerning Asset Manager architecture

Asset Manager relies on a two-tier architecture. The Asset Manager client connects directly to the database and falls into the category of what is usually named a "fat" client. A "fat" client resides completely on the client computer. The advantage of this kind of architecture is better performance on a fast LAN. The disadvantage is that the performance gets geometrically worse on a slow network. What this means is if your network latency is twice as bad as the recommended (10 milliseconds), your actual performance decrease will be approximately four times worse.

One of Asset Manager's key components is Asset Manager Automated Process Manager.
Asset Manager Automated Process Manager does not exchange any data with the client. It connects to the database separately, and performs all of the background tasks that require regular triggering and/or monitoring. For example: an alarm that is set for a contract's end date and that issues an email message 30 days before that date. Asset Manager Automated Process Manager also processes workflows, calculates rents, and validates the authorization code in the database. Most of the tuning suggestions made in this guide do not apply to Asset Manager Automated Process Manager.

Bottlenecks

There are basically four areas of Asset Manager where a performance bottleneck can occur: the network, the client, the database server, and the database engine.

Network bottlenecks usually occur when you are utilizing Asset Manager on a slower network than it was designed to utilize.

Client bottlenecks are often actually database or network bottlenecks. Usually what happens is that the queries take a long time to return data (either because of a slow network or a long database response time). Database server bottlenecks occur when the hardware that the database server is running on is insufficient for the implementation of Asset Manager. Typically this is not a problem area because most implementations analyze their requirements in advance and make sure that the hardware is good enough, but once in a while this can be a problem.

Database engine bottlenecks are the most common. What happens in this case is the database engine is not able to respond rapidly to the queries being sent to it. This can happen for a variety of different reasons; poor database tuning, complex queries, bad optimizer choices, or insufficient resources...

> **Note:** One thing to remember is just because you have a powerhouse of a server doesn't mean

> that the database is configured to take advantage of all of that power. Also, if you have several different applications sharing a single database server, that means the resources are going to be split among them, and that can affect performance.

Chapter 2: Tuning a client

This chapter explores the different performance-optimization strategies for Asset Manager. It discusses certain internal mechanisms of Asset Manager that can impact performance and provides troubleshooting tips.

Network latency

As mentioned above, the network has a direct impact on application performance. The recommended network latency for Asset Manager is 10 milliseconds or less.

Tree view and table view

Asset Manager enables you to view records in two distinct modes:

- A tree view of the records. This view shows you the hierarchy of the table, thus allowing you to unfold the different leaves and easily view the parent-child relationships being maintained in that table. Any table that has a **FullName** field can be viewed in tree view. Some examples are **Employees/Departments** (amEmplDept), **Portfolio Items** (amPortfolio), **Locations**

(amLocation), and so on.

- A list view of the records without any hierarchy information.

The way the tree view list is built is very costly in terms of performance. If you are on a fast network with a table that is not very large, tree view will work fine. But the larger the table and the slower the network, the worse that performance is going to get.

Two tuning methods are immediately available to you:

1. Viewing the tables in the list view if the network is slow and/or the table is of considerable size.
2. Modifying the default display parameters of hierarchic lists so that they are initially collapsed. This speeds up the initial screen load. To do this, set the **Initially collapsed** option in Asset Manager to **Yes**. (**Edit/ Options/ Lists/ Main lists** and **Edit/ Options/ Lists/ Other lists**).

List-loading mechanism

When the **Initially collapsed** option is enabled, Asset Manager uses the following mechanism to display the lists (the following example concerns the opening of the **Departments and Employees** table in tree view):

1. Asset Manager sends a query to recover all the records at the top of the hierarchy (those for which the **sLvl** field is set to **0**) and whose **Full name** (**FullName**) field is greater than the first record in the list. The query sent to the database is the following:

   ```
   SELECT E1.lIconId, E1.lEmplDeptId, E1.MrMrs, E1.Name, E1.FirstName, E1.Title,
   E1.Phone, E1.Fax, E1.FullName FROM amEmplDept E1 WHERE E1.sLvl=0 AND
   E1.FullName >='/Admin,,ADMIN/' ORDER BY E1.FullName
   ```

2. When a branch is unfolded, a new query is sent to the database. This query recovers the records directly underneath and whose full name contains the name of the parent. For example (in this case, the Taltek branch of the tree is unfolded):

   ```
   SELECT E1.lIconId, E1.lEmplDeptId, E1.MrMrs, E1.Name, E1.FirstName, E1.Title,
   E1.Phone, E1.Fax, E1.FullName FROM amEmplDept E1 WHERE E1.FullName LIKE
   '/Taltek/%' ESCAPE '\\' AND E1.sLvl=1 ORDER BY E1.FullName
   ```

   Each time a branch is unfolded, a query of this type is sent by Asset Manager. If the record you are looking for is lower down in the tree, the amount of queries sent to the database is increased.

On the other hand, the same table opened in list view results in one single query as follows:

```
SELECT E1.lIconId, E1.lEmplDeptId, E1.MrMrs, E1.Name, E1.FirstName, E1.Title,
E1.Phone, E1.Fax, E1.FullName FROM amEmplDept E1 ORDER BY E1.lEmplDeptId
```

All the records returned by the query are used to populate the list, the size of which is limited as defined in the Asset Manager options.

Loading lists

Several options are available to define the number of records initially loaded in a list. By reducing or limiting this number, you can improve the response time when displaying a list. The options in question are located in the **Edit/ Options/ Lists/ Main lists** and **Edit/ Options/ Lists/ Other lists** sections. The available options are:

- **Do not load for longer than**: This option defines the maximum time (in milliseconds) during which Asset Manager tries to populate the list. When this time is overrun, the operation is suspended and the list is populated with those records recovered. By default, this value is set to 5000 milliseconds.
- **Do not load more than**: This option defines the maximum number of records loaded in a list. By default, 200 records are loaded.

  > **Note:** This number also corresponds to the number of additional records that are loaded when clicking the + icon next to list.

The default values of these two options are appropriate for the vast majority of cases. However, in specific circumstances when the list configuration is complex, these values may not suit you in terms of performance. If performance issues are encountered displaying lists, you may wish to:

- Reduce the value of the **Do not load more than** option to 30 or 50, for example.
- Change the value of the **FetchingArraySize** parameter in Asset Manager. This parameter specifies the number of records recovered by Asset Manager in each network packet. For performance reasons, this value must be the same as that defined in the **Do not load more than** plus one. For example, if the value of the **Do not load more than** option is 200, the **FetchingArraySize** parameter must be set to 201. This increment accounts for the NULL record, which is returned in the results of the query but not displayed by Asset Manager.

  To modify the value of the **FetchingArraySize** parameter:

  a. Load the **amdb.ini** file in a text editor, the default path of **amdb.ini** is as follows.

     For Windows: *<System Drive>:\ProgramData\HPHPE\AssetManager\conf*

     For other operating systems, search for “Available .ini, .cfg and .res Files” in Asset Manager Help Center.

  b. In the section corresponding to the declaration of the database connection, find the **FetchingArraySize** parameter. The following is an example for the **AMDemo96en** database:

```
[AMDemo96en]
Engine=SQLAnywhere
Location=AMDemo96en
EngineLogin=itam

EnginePassword=7BC2423F1B1F6A42E2B3A27E396F52C3A9AD6828582AED88DAC2F53E9A369B
C03E6393911903124254200200
ReadOnly=0
CacheSize=5120000
FetchingArraySize=201
AmApiDll=C:\PROGRA~1\HPE\ASSETM~1\ASSETM~2\bin\aamapi96.dll
```

c. Modify the value of the parameter and save your changes.

> **Note:** Changing this parameter may also prove useful when Asset Manager is used over a WAN (Wide Area Network).

Another option to adjust the performance is **Lists/ Refresh all data when any record is modified**. By default, this option is enabled, and Asset Manager refreshes all list data whenever a record is modified. To improve the performance, you can disable this option and press F5 to manually refresh data.

Displaying error message

Asset Manager provides options to enable you to customize the display of the error messages (the result returned by the **Err.Raise (<Error number>, <Error message>, [Hide system error])** function).

For both Windows and Web client,

- If **[Hide system error]** is set to '1', the function hides the unwanted system error message (the 'error in Line number' information).
- If it is set to '0', the function returns full information.

Based on this implementation, the Web client goes a step further by providing another option **Error.Message.Display.FullText** in the **C:\Program Files\HPE\Asset Manager 9.61 xx\webtier\package.properties** file.

- If **Error.Message.Display.FullText** is set to **FALSE**, a **+/-** icon appears so that you can collapse/fold the error message.
- If it is set to **TRUE**, then all of the information defined by the **Err.Raise()** function is displayed.

Type-ahead

By default, Asset Manager uses type-ahead functionality to help users populate a link field. It works by waiting for a certain amount of time (which by default is after 500 milliseconds, or half a second) after a user stops typing in a link field, and then queries the database to find the first record that matches what the user has typed so far.

Most of the time this can be very handy; for instance, if you're typing in a person's name as a user of a portfolio item, it can help you populate the link after typing only a few letters of the person's last name. The bad side, however, is that once the type-ahead starts to execute the user will have to wait until the query finishes before they can continue to type. In situations where there is a slow network or where the table the link connects to is large, this can be very time-consuming. In these cases it is recommended to disable type-ahead. To do this:

1. Select the **Edit/ Options/ Navigation/ Selection of linked records/ Type-ahead after** menu,
2. Click the number next to this option.
3. Change it to a very large number (for example 1,000,000) to prevent type-ahead from ever being triggered,
4. Click **OK**.

Access restrictions

An access restriction is part of an Asset Manager user profile. It corresponds to a record filter on a table. For example, you can make sure that technicians only have access to the assets in their own department.

Because they are a designed to provide row-level security, access restrictions are commonly used, and rightly so. However, because they affect the query that gets sent to the database when a user opens a screen, a poorly written or poorly defined access restriction can cause all kinds of performance issues.

Access restrictions are always in the context of a specific table, and are broken down into two areas: Read and Write. Read access restrictions prevent users from seeing a particular set of records. By corollary, they also prevent users from modifying certain sets of records. Write access restrictions prevent users from modifying certain records. They do not affect their ability to read any data in the record, they simply receive an error message whenever they try to make a change to that record.

Access restriction mechanism

- Read restrictions are applied via a WHERE clause added to the query sent to the database. For example, you have an access restriction that a user may only view cost centers for which they are the supervisor. The query sent is as follows:

  ```
  SELECT
    C1.lCostId,
    C1.Title,
    C1.AcctNo
  FROM
    amCostCenter C1,
    amEmplDept amEmplDept_CurrentUser
  WHERE
    amEmplDept_CurrentUser.lEmplDeptId   = C1.lSupervId
  AND amEmplDept_CurrentUser.lEmplDeptId = 25323
  ORDER BY C1.lCostId
  ```

  This typical and simple case is not very costly in performance terms. But if the query is more complex, and in particular if it contains one or more sub-queries, the database engine will take more time in processing and sending the result to Asset Manager, which will result in a slower display. Once more, the size of the table concerned by the query plays an important role.

- Write restrictions work in a slightly different way. When a user tries to modify a record in a table with one or more write access restrictions, Asset Manager sends a separate SELECT query to the database specifying the access restriction in the WHERE clause. If the record that the user is trying to modify is not returned, a message informs the user that they do not have the required permissions and the changes are cancelled at the database level. If the record is returned, an UPDATE statement is sent to the database and the modifications are validated. The network traffic is potentially twice as high (a test query plus a modification query). In addition, as for read restrictions, the complexity of the queries and the size of the tables concerned also plays a role in reduced performance.

> **Note:** It is therefore crucial to carefully analyze how you require access restrictions to be applied and to implemented them in the most succinct and rational manner possible.

Properties of the objects

The properties of database objects (fields, links…) may have a significant impact on performance.

> **Note:** These properties can been seen via the **Configure object** shortcut menu or by using Asset Manager Application Designer.

The display properties

The Asset Manager database objects, fields and links in particular, have properties that determine how they are displayed. These properties are as follows:

- **Mandatory**: The label of this field is shown in red and the field must be populated in order to be able to validate the record.
- **Read only**: The field is grayed out.
- **Irrelevant**: The field is not displayed.

These properties may be set to the following values:

- Yes
- No
- Script

  In this last case, an interpreted Basic script determines the final value of the property.

A value of Yes or No for a property does not impact performance because it is not required to be evaluated. This is not the case for a script. When a script is loaded or updated, all the scripts associated with the record are executed. Simple scripts that work on local data are relatively transparent in terms of performance. On the other hand, scripts that query the database (for example, when using `AmDbGetLong()` type functions) will generate queries and have a direct impact on performance. It is the same when the value of a field or a remote link is referenced in a script, for example:

```
[CostCenter.Supervisor.Location.City]
```

To resume:

- Analyze clearly the need to script a property,
- Bear in mind that all scripts that perform database queries may impact performance,
- Optimize your scripts for efficiency.

History

Asset Manager makes it possible to keep history of modifications made to database objects. This functionality is controlled by the **Keep history** property of fields and links. The history mechanism is as follows:

1. A user modifies a record. If any object concerned by this modification has a **Keep history** properties that evaluates to 1 (that is, it is set to **Yes** or a script returns the value 1), the history agent is triggered.
2. The history agent creates a record in the History table (**amHistory**) storing both the old and new values of the object.

As for the **Mandatory**, **Read only** and **Irrelevant** properties mentioned previously, the **Keep history** property may cause performance issues if a complex script (querying the database) is associated with it. Another impact on performance is directly linked to the fact that an insert statement is sent to the database for each historized object. Ideally, if the number of historized objects is limited this will not cause any particular problems. This is not the case when dozens of historized objects are modified. Another factor in reduced performance is linked to the fact that the amount of time it takes for the query to complete is directly related to the size of the table. The more records there are in the table, the longer it takes. And typically if someone is keeping history on several fields, the amHistory table can get very, very large, in some cases upwards of 10 million rows.

We therefore recommend:

- For best performance, only keep history on essential fields. Do not historize fields systematically.
- Choose an amount of time after which you can delete history. You can write a workflow to delete history after 6 months, for example.

Validity check

Validity scripts are used by Asset Manager to check certain conditions of records before inserting them into the database. For example, this type of script is used to check that an end date in a contract does not come before the start date. If this occurs, the record is not inserted and an error is raised.

Scripts are subject to the same performance constraints as previously mentioned for the properties of objects.

Features

Features are a historical functionality of Asset Manager introduced before version 4.0.0. They provided the ability to define additional information for records when it was not possible for users to extend the database schema. Despite their power, features are often costly in performance terms. Taking the Locations table as an example, three tables are impacted when using the features, as illustrated below:

The Locations table (amLocation) does not actually hold any information about features. It is simply the target of the Feature Values (amFVLocation) table, which holds the values of the features associated with the Locations table. The Features (amFeature) table itself contains the physical data on the feature, such as its SQL name and Label, and so on.

Each feature you assign to a record causes at least one extra join to be executed to pull in that data. The impact on performance may be significant, especially if the feature is displayed in the list.

Since Asset Manager 4.0.0, it is now possible to modify the database schema directly. Using features is less justifiable. However, you may still wish to do so in the following two cases:

1. When the piece of information is going to be on very few records (1% or less, but may vary depending on implementations), you may wish to use a feature rather than create a new field in the table. A feature only takes up space when it is populated (the only notable exception being when the **Force display** property is on). An additional field takes up space regardless. Under these circumstances, using a field may use up a large amount of storage space unnecessarily.
2. Modifying the database structure is relatively straightforward but can be risky. If something happens during the structure update that causes it to stop, your database will be in an inconsistent state and you will have to restore from a backup. For this reason, HPE recommends:
   - Backing up the database before modifying the structure,
   - Carefully planning changes and having them performed by personnel with the required competences.

   During the planning stage, you may want to temporarily use feature before permanently creating a field to link to perform the same role.

List configuration

A majority of performance problems experienced when using the Asset Manager client are due to poor list configuration choices. When a user opens a screen, Asset Manager executes a query to pull in all

list data. The query is dynamically built based on several settings (the columns in the list, sorts, filters, and so on), read access restrictions, table/tree view, and so on.

The choice of columns in the list is particularly important as they are used in the SELECT clause of the query. Performance is optimal when querying one single table only. However, if you add a column that is a link to another table, a new join is created in the query.

> **Note:** This is also true for features, for the reasons already mentioned.

Before adding the following items to a list configuration, we also recommend thinking through your needs:

- A calculated field: Basic-type calculated fields also slow things down because they must be evaluated once for every record in the list.
- The **self** description string: For certain tables, the description is complex (for example, the Portfolio Items table) and may require multiple outer joins.

Sorts

When a list is sorted on the value of a field or link, the database engine sorts the results that are sent to Asset Manager. This operation is not expensive when the object is indexed in the database. The following list resumes the sort strategies that can be applied to lists starting with the least expensive in performance terms:

1. Sort on an indexed field in the table
2. Sort on an indexed field on a linked table one link distant (such as **User.BarCode** from the list of Portfolio Items)
3. Sort on an non-indexed field in the table
4. Sort on multiple indexed fields in the table
5. Sort on multiple non-indexed fields in the table
6. Sort on multiple fields (indexed or not) in a linked table one link distant
7. Sort on the description string of a linked table
8. Sort on fields (indexed or not) in a linked table more than one link distant

Filters

Filters translate directly to the WHERE clause of the query that populates the list on a table. As such, you should consider how they are going to be treated by the database when you create them. Filters that involve linked tables are necessarily going to be slower than filters that only involve fields on the main table. Complex filters can, if improperly designed, make the list open extremely slowly. Some rules of thumb for writing filters:

- Whenever possible, try to use AND instead of OR
- Try to avoid using LIKE and NOT LIKE. If you must use them, try to only use a trailing wildcard and use an indexed field. For example, it is better to write:

  ```
  FullName LIKE '/Taltek/%'
  ```

  than:

  ```
  FullName LIKE '%/Taltek/%'
  ```

  Wildcards at the start of the string for comparison force a complex search through the whole database, which can prove very costly.
- It is better to use IN and NOT IN clauses with constants. For example:

  ```
  Name IN ('Thomas', 'James')
  ```

  Using these clauses with a sub-queries is very expensive.

Performance log

You can enable the performance log to monitor the response times that the web service uses to handle the web tier requests.

To enable the performance log, follow these steps:

1. Open `http://<Name or IP address of the Asset Manager Web Service server>:<Asset Manager Web Service port>/AssetManagerWebService/diagnostics/logConfig` in a web browser.
2. On the **Log4j Configuration** page, type `PERF` in the **Debug** column, and then click **Update**.

   > **Note:** The web service generates the information of the response time of each request from

> the web tier. This information is stored in the log file of the web service. For example, if you use Tomcat as the application server, by default, this information is stored in the `<tomcat home>\logs\catalina.<date>.log` file.

3. On the **Log4j Configuration** page, type `com.peregrine.ac.ws.axis.ApiLogger` in the **Debug** column, and then click **Update**.

> **Note:** The web service generates the detailed response time of each JNI call in the requests from the web tier. This information is also stored in the log file of the web service.

The output of the performance log resembles the following.

```
18:46:17,423 DEBUG - PERF - ws call 'contain soap serialize time' start //Request
starts
18:46:17,426 DEBUG - com.peregrine.ac.ws.axis.ApiLogger - [ApiLogger]ws call api
isMultiTenantSystem() [duration=0]
18:46:17,426 DEBUG - com.peregrine.ac.ws.axis.ApiLogger - [ApiLogger]ws call api
startReadTransaction("admin") [duration=0]
18:46:17,443 DEBUG - com.peregrine.ac.ws.axis.ApiLogger - [ApiLogger]ws call api
getAccessibleItems("admin",8) [duration=17]
18:46:17,526 DEBUG - PERF - [ACWSRPCProvider]ws call
[com.peregrine.ac.ws.api.ACBeanListApi.retrieveFirst] directly without soap
duration=100 //Request handle method
18:46:17,557 DEBUG - PERF - [ACWSServlet]ws call 'contain soap serialize time' end
[duration=133] //Request ends
```

Tuning Websphere

When using Websphere, you can follow these steps to improve the performance.

1. Locate the following directory:

   `<driver>:\ibm\WebSphere\AppServer\profiles\[App Server name]\properties\`

2. Create an empty file named "commons-logging.properties" with the following content:

   ```
   priority=1

   org.apache.commons.logging.LogFactory=org.apache.commons.logging.impl.LogFactor
   yImpl
   ```

3. Restart Websphere and test the performance.

Chapter 3: Tuning the database

This chapter deals with tuning the database used by Asset Manager. It is not intended to replace specialized books on the subject but does describe several tuning strategies for Asset Manager. Whatever the case, it is highly recommended to leave this sort of operation to a qualified database administrator.

> **Note:** After you tune the database by editing stored procedures, if you perform any major Application Designer operation such as upgrading the database, the customized stored procedures may be dropped and overwritten by the OOB version. Therefore, we recommend that you always back up the stored procedures so that you can recover the changes after such Application Designer operations.
>
> The same rule applies to customized indices.

About hardware

The levels of database server hardware you need for your database is one of the hardest things to quantify. There are so many different factors involved, such as the number of users, the number of databases, the database engine, other applications running on the server, the quantity of data, and so on.

This section includes:

RAM

As a general rule of thumb, RAM is going to have the greatest impact on database performance. RAM is where the DBMS caches queries and data, so the more of it there is the more likely you are to get a cache hit when you execute a query. It is also where information for each user connection to the database is stored, so the more users you have the more RAM you need.

CPUs

The number and type of CPUs is also going to have an effect. If your DBMS can make use of multiple processors, then it is beneficial to have several, which allows the DMBS to do calculations in parallel. And of course, in general, the faster the CPU the better.

Hard drives

Hard disk I/O will also has a major effect on database performance. If you have a blazingly fast CPU with tons of RAM but your hard disk is slow, then any time the DBMS has to go to disk to retrieve data, everything else has to wait until is done. Ideally, you want the fastest hard disks you can get, typically SCSI drives. You also may want to consider a RAID setup. RAID stands for Redundant Array of Independent Disks. It is essentially a way to combine multiple hard disks into one system for enhancing certain aspects of performance:

- If you want to set up a bare minimum RAID system, then you should consider RAID 1. RAID 1 is mirroring. That means that every disk in the array has another identical disk that duplicates its data exactly. This allows a significant increase in Read I/O performance because both disks can be read in parallel (since they both contain the same information). As the majority of Asset Manager queries involve read operations only, this will probably give you good performance. It also has the advantage of fault tolerance; if one disk crashes, causing data loss, you still have another disk with exactly the same data on it, so none of the data is actually lost. You just slip a new hard disk into the array and let it mirror the data again. Unfortunately there is no benefit to writing with this system. The other disadvantage is that you must have an even number of disks, so each one can be mirrored.
- If you have the money and the hardware, however, a RAID 10 system is better. This combines both mirroring and striping into one system. Striping is when the system writes data across several

disks. This allows very fast write performance because each disk has a section of the data written to it simultaneously.

Network

As mentioned on several occasions, network performance is of extreme importance. The faster the network is, the greater the bandwidth and a greater number of concurrent users can be handled.

About statistics

Asset Manager uses the number zero instead of "null" values. This design relies on database statistics a lot. The performance will get impacted once the statistics are not up-to-date.

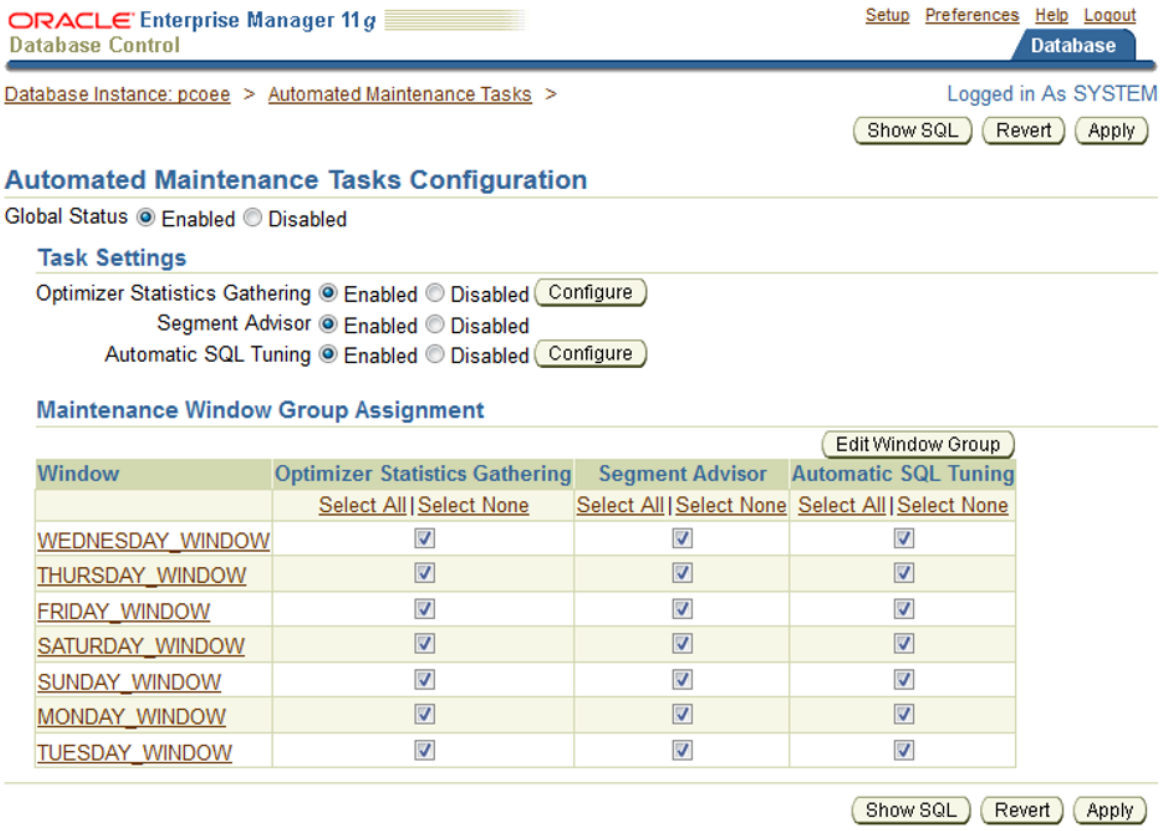
From Asset Manager 9.50, the "Stats" module is removed from APM because all supported databases like Oracle and SQL Server are able to provide better background statistics maintenance with their out-of-box tools. We recommend that you gather the database statistics with these tools on a daily or weekly basis.

Oracle

Choose either way to enable the automatic statistics gathering.

GUI

Log on to the Enterprise Manager, navigate to **Server** -> **Automated maintenance Tasks**, click the **Configure** button, select **Enabled**.



SQL Plus

```
dbms_auto_task_admin.enable ();
```

SQL Server

GUI

Right -click **Database** -> **Properties** -> **Options**, set **Auto_Update_Statistics** and **Auto_Update_Statistics_Asynchronously** to **True**.

SQL

```
ALTER DATABASE dbName SET AUTO_UPDATE_STATISTICS ON
ALTER DATABASE dbName SET AUTO_UPDATE_STATISTICS_ASYNC ON
```

DB2

SQL

```
CONNECT TO <db alias>
UPDATE DB CONFIG USING AUTO_MAINT ON AUTO_TBL_MAINT ON AUTO_RUNSTATS ON
```

About queries

To better understand how query transactions impact performance, it is worthwhile detailing how a query is sent by Asset Manager:

1. Asset Manager builds the query and sends it to the database, but no parameters are sent yet. The parameters are the hard-coded values of the query that are needed to specify what is returned. For example, in the following WHERE clause:

   ```
   WHERE lEmplDeptId=12345
   ```

   12345 is the parameter.

2. The database's query optimizer compiles an **access plan** for the data and returns it to Asset Manager. The value is irrelevant at this point. Using the above example, if the optimizer knows that it is going to get a value for **lEmplDeptId**, it will be able to determine an appropriate access plan (such as using the index on **lEmplDeptId**).
3. Asset Manager binds the parameters to the query, then sends it back to the database.
4. The database executes the query based on the access plan and retrieves the data, then returns it to Asset Manager.

> **Note:** On some RDBMSs, steps 1 and 2 are not used. In that case, the query is just sent directly to the database with the parameters bound and without getting the access plan in advance. The impact on performance may be significant.

About database engines

This section contains a few recommendations that are valid for all database engines, although they will have differing levels of effectiveness depending on your DBMS.

- First off, when possible, keep indexes and data in separate tablespaces. Asset Manager uses a lot of indexes out of the box, and performance will typically be improved by having the indexes separated from the data. This is especially true when you can put the two tablespaces on separate disks, where the database can do simultaneous reads of both.

- Another recommendation is to calculate statistics for tables regularly. This can be performed by your database administrator as a regular database maintenance task. Table statistics are used to determine how often data is used. This lets the optimizer improve access to frequently used data. You can enable the automatic statistics for tables on the database level to achieve better performance.
- Rebuild your indexes regularly, as well, especially if you do a lot of inserts/deletes. After a while, the indexes become less efficient because of holes in their data and size. If you mostly do updates and there is not a lot of data being added or deleted, you can probably do this once a quarter. Most likely, though, you should have the DBA perform this once a month as part of the regular database maintenance.
- If you are seeing poor performance overall, you may want to check how much server swapping is going on. This happens when you have virtual memory specified and the operating system has to swap data out of RAM and onto the disk (in the pagefile) to make space for other data, or it needs to swap the data from the disk back into physical RAM in order to act on it. This can happen if there is insufficient physical RAM on the machine, or there is not enough physical RAM allocated to the RDBMS. If you see this happening (which you can see by looking at the Task Manager in Windows), you either need to allocate more physical memory to the RDBMS (if possible) or add more physical memory to the server.

Eliminating locks and deadlocks

When several users are doing similar operations with quite large transactions involving counters, there is a significant probability that some concurrent transactions may end with deadlock on amCounter. This section deals with troubleshooting the deadlocks on several RDBMs.

> **Caution:** This information is intended for experienced database administrators.

This section includes:

Microsoft SQL Server

The proposed solution is to replace the amCounter table for most used counters and deport them to other tables. Each counter will have its own table to minimize locking.

Step 1 - Alter the SQLServer Schema isolation level

Execute the following command on the database, replacing **<AMSchema>** with the real schema name:

```
ALTER DATABASE <AMSchema> SET READ_COMMITTED_SNAPSHOT ON
ALTER DATABASE <AMSchema> SET ALLOW_SNAPSHOT_ISOLATION ON
GO
```

> **Note:** If the execution takes too long, you may need to disconnect all connections to the database. One possible way is to restart the database service, then execute the command. If you restart the SQL Server, and an Integration Point has already been created in UCMDB, you should restart the UCMDB Probe as well to avoid the issue of dead connections.

Step 2 - Alter Asset Manager database options

1. Open the Asset Manager Client and connect to the appropriate database schema.
2. Navigate on the top menus to **Administration** > **Database options**.
3. For option **'Sql Server specifics'|'Isolation command before starting a write transaction'**, change the current value to **set transaction isolation level snapshot**.

Step 3 - Identify the deadlocking counters

Most likely, the counters with the highest numbers are the problem areas, but there might be others if some specific operations are leading to locks. Begin by identifying the counters and counter names. To do this:

1. Open MS SQL Server Query Analyzer
2. Run the following query against the Asset Manager database:

   ```
   select * from amcounter
   ```

   This query returns the current value of the counters stored in this table.

> **Note:** For the steps to follow, the following counters will be used: **amItemReceived_ItemNo**, **amAsset_AssetTag**, **amTicket_TicketNo** and **amExpenseLine_ItemNo**.

Step 4 - Backup

Backup the stored procedure **up_GetCounterVal**.

Step 5 - Create tables for the counters

Run a CREATE TABLE script for each of the counters you are deporting using the following format:

```
CREATE TABLE [itam].[<NewTableName>] (
     [lValue] [int] IDENTITY (<IdentityStart>, 1) NOT NULL ,
     [luserspid] [int] NOT NULL
) ON [PRIMARY]
GO
```

In this script:

- **NewTableName** is the name of the table to create, typically in the format **cnt_<Original counter name>**.
- **IdentityStart** is the current value of the original counter table.

A typical SQL query that can be used to generate these CREATION STATEMENTS on the database would be:

```
select 'CREATE TABLE [itam].[cnt_'+Identifier+'] ([lValue] [int] IDENTITY ('+ltrim
(rtrim(str(lValue)))+',1) NOT NULL ,[luserspid] [int] NOT NULL) ON [PRIMARY]' as
sqlquery from amCounter where lCounterid<>0
```

> **Note:** In this example the owner of the tables will be specified as **itam**, be sure it matches your database and modify the script if necessary.

Using the examples of counters defined above, the statements sent will be the following:

```
CREATE TABLE [itam].[cnt_amAsset_AssetTag] (
     [lValue] [int] IDENTITY (17697, 1) NOT NULL ,
     [luserspid] [int] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [itam].[cnt_amExpenseLine_ItemNo] (
     [lValue] [int] IDENTITY (10735, 1) NOT NULL ,
     [luserspid] [int] NOT NULL
) ON [PRIMARY]
```

```
GO

CREATE TABLE [itam].[cnt_amItemReceived_ItemNo] (
     [lValue] [int] IDENTITY (4, 1) NOT NULL ,
     [luserspid] [int] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [itam].[cnt_amticket_ticketno] (
     [lValue] [int] IDENTITY (140000, 1) NOT NULL ,
     [luserspid] [int] NOT NULL
) ON [PRIMARY]
GO
```

Step 6 - Create a stored procedure

The next step is to create the stored procedure to handle counters. We suggest that you use the following example, replace the counters name in the stored procedure and add/remove as many procedure blocks as needed. Also, make sure the counter name appears in the NOT IN () statement used to distinguish counters using their own table from those still using **amCounter**.

```
sp_dropmessage 50895
GO
sp_addmessage 50895, 16,'up_GetCounterVal increment is %d which is not supported
when modified for non-locking optimization. Please edit the AMDB.INI file on your
computer and add ImportCounterCache=1 in the current database connexion
description'
GO
CREATE PROCEDURE up_GetCounterVal @CounterName varchar(64), @CounterIncrement int
AS
BEGIN
  DECLARE @CounterIdent int
  'Change the NOT IN() list in this next line to reflect the names of the new
counters in your database.
  IF @CounterName NOT IN ('amItemReceived_ItemNo', 'amAsset_AssetTag', 'amTicket_
TicketNo', 'amExpenseLine_ItemNo')
  BEGIN
    DECLARE amCounterLock CURSOR FOR SELECT lCounterId FROM amCounter WHERE
Identifier = @CounterName FOR UPDATE
    SELECT @CounterIdent = MAX(lCounterId) FROM amCounter WHERE Identifier =
@CounterName
    IF @CounterIdent IS NULL
      INSERT INTO amCounter (lCounterId, Identifier, Description, lValue,
dtLastModif) SELECT MAX(lCounterId) + 1, @CounterName, @CounterName,
@CounterIncrement, GetDate() FROM amCounter
    ELSE
    BEGIN
```

```
        OPEN amCounterLock
        UPDATE amCounter SET lValue = lValue + @CounterIncrement, dtLastModif =
GetDate() WHERE lCounterId = @CounterIdent
    END
    SELECT lValue FROM amCounter WHERE Identifier = @CounterName
    IF @CounterIdent IS NOT NULL
      CLOSE amCounterLock
    /* END IF*/
    DEALLOCATE amCounterLock
  END
  ELSE
  BEGIN
  IF @CounterIncrement <> 1
    RAISERROR (50895, 17, @CounterIncrement)
  /* END IF */
  -- See generation sql below
  --Add an IF block (next 5 lines) for EACH new counter table. Insert the original
counter name in all the bold faced areas.
  IF @CounterName = 'amTicket_TicketNo'
    BEGIN
      insert into cnt_amTicket_TicketNo (luserspid) values(@@SPID)
      SELECT max(lValue) FROM cnt_amTicket_TicketNo WITH (NOLOCK) where
luserspid=@@SPID
    END
  IF @CounterName = 'amAsset_AssetTag'
    BEGIN
      insert into cnt_amAsset_AssetTag (luserspid) values(@@SPID)
      SELECT max(lValue) FROM cnt_amAsset_AssetTag WITH (NOLOCK) where
luserspid=@@SPID
    END
  IF @CounterName = 'amExpenseLine_ItemNo'
    BEGIN
      insert into cnt_amExpenseLine_ItemNo (luserspid) values(@@SPID)
      SELECT max(lValue) FROM cnt_amExpenseLine_ItemNo WITH (NOLOCK) where
luserspid=@@SPID
    END
  IF @CounterName = 'amItemReceived_ItemNo'
    BEGIN
      insert into cnt_amItemReceived_ItemNo (luserspid) values(@@SPID)
      SELECT max(lValue) FROM cnt_amItemReceived_ItemNo WITH (NOLOCK) where
luserspid=@@SPID
    END
  -- end of generation sql below
  END
END
GO
```

The content of the NOT IN() statement and the IF block statements can be respectively generated with the following queries:

```
select ''''+Identifier+''', ' as sqlquery from amCounter where
lCounterid<>0
```

and

```
select '  IF @CounterName ='''+Identifier+ '''
    BEGIN
      insert into  cnt_'+Identifier+' (luserspid) values(@@SPID)
      SELECT max(lValue) FROM  cnt_'+Identifier+' WITH (NOLOCK) where
luserspid=@@SPID
    END' as sqlquery from amCounter where lCounterid<>0
```

Step 7 - Check the stored procedure

Run the following script:

```
exec up_getcounterval 'amAsset_AssetTag',1
SELECT COUNT(*) from  cnt_amAsset_AssetTag
exec up_getcounterval 'amItemReceived_ItemNo',1
SELECT COUNT(*) from  cnt_amItemReceived_ItemNo
exec up_getcounterval 'amTicket_TicketNo',1
SELECT COUNT(*) from  cnt_amticket_ticketno
exec up_getcounterval 'amExpenseLine_ItemNo',1
SELECT COUNT(*) from  cnt_amExpenseLine_ItemNo
```

Each and every new counter table should theoretically have one record.

Step 8 - Modify the stored procedure UP_GETID

Locking on ID generation may also occur sometimes. To minimize locking, modify the stored procedure **UP_GETID** as follows :

```
DROP procedure UP_GETID
GO
CREATE procedure UP_GETID as
declare @id int
set @id = 0
DECLARE @result int;
declare @NewId as table(Id int)

begin tran tran1
    EXEC @result = sp_getapplock @Resource = 'up_getid', @LockMode = 'Exclusive',
@LockTimeout = '0';
    if (@result >= 0) begin
        select @id = Min (IdSeed) from LastId where lRemain > 0 and lInUse = 0
        if (@id > 0) update LastId set lInUse = 1 where IdSeed = @id
        else begin
            insert into LastId(Value, lRemain, lInUse) output Inserted.IdSeed INTO
```

```
@NewId values(@@SPID,32, 1)
            select @id = Id from @NewId
        end
    end
    else begin
        insert into LastId(Value, lRemain, lInUse) output Inserted.IdSeed INTO
@NewId values(@@SPID,32, 1)
        select @id = Id from @NewId
    end
commit tran tran1
return @id
GO
```

Step 9 - Modify the stored procedure UP_GETINDEPID

Locking on independent ID generation may also occur sometimes. To minimize locking, modify the stored procedure **UP_GETINDEPID** as follows :

```
DROP procedure UP_GETINDEPID
GO
CREATE procedure UP_GETINDEPID as
declare @id int
set @id = 0
DECLARE @result int;
declare @NewId as table(Id int)

begin tran tran1
    EXEC @result = sp_getapplock @Resource = 'up_GetIndepId', @LockMode =
'Exclusive', @LockTimeout = '0';
    if (@result >= 0) begin
        select @id = Min (IdSeed) from IndependentLastId where lRemain > 0 and
lInUse = 0
        if (@id > 0) update IndependentLastId set lInUse = 1 where IdSeed = @id
        else begin
            insert into IndependentLastId(Value, lRemain, lInUse) output
Inserted.IdSeed INTO @NewId values(@@SPID,32, 1)
            select @id = Id from @NewId
        end
        EXEC @result = sp_releaseapplock @Resource = 'up_GetIndepId'
    end
    else begin
        insert into IndependentLastId(Value, lRemain, lInUse) output
Inserted.IdSeed INTO @NewId values(@@SPID,32, 1)
        select @id = Id from @NewId
    end
commit tran tran1
return @id
GO
```

Step 10 - Create a clean-up stored procedure

The purpose of this step is to build a stored procedure that will wipe out each counter table periodically when no inserts are pending.

1. Get the object ID for each counter table. Run the following query:

```
select id, name from sysobjects where name like 'cnt%'
```

2. You should get a result set like this:

```
id          name
----------  ------------------------
1793441463  cnt_amAsset_AssetTag
2001442204  cnt_amExpenseLine_ItemNo
1745441292  cnt_amItemReceived_ItemNo
1505440437  cnt_amticket_ticketno
```

3. Generate the stored procedure for wiping out specific counter tables:

```
To generate all the main code block below for all deletes, run the sql query
select 'delete from [cnt_'+Identifier+'] WITH (TABLOCKX)' as sqlquery from
amCounter where lCounterid<>0

DROP PROCEDURE AM_Cleanspeccounters
GO
CREATE PROCEDURE AM_Cleanspeccounters AS
BEGIN
     declare @locknb int
     set nocount on
     --while 1=1

     'Insert a code block (Next 1 line) for EACH new counter. Insert the
appropriate ID or counter name in the bold faced areas.
     delete from cnt_amAsset_AssetTag  with (TABLOCKX)
     delete from cnt_amExpenseLine_ItemNo  with (TABLOCKX)
     delete from cnt_amItemReceived_ItemNo with (TABLOCKX)
     delete from cnt_amticket_ticketno with (TABLOCKX)
...

--Next line is to be commented when you define a new SQL Agent Task to kick off
the procedure each time on the server (it is in fact recommended).
--waitfor delay '000:05:00'

 --     'here the stored procedure is fired each 5 minutes

 --     CONTINUE
--    end
end
```

4. Define a new SQL Agent Task to kick off the procedure each hour or so on the server.
5. Verify that it works. After the programmed delay all specific counter tables should be empty.

Step 11 - Edit the **amdb.ini** file

1. Open the **amdb.ini** file of Asset Manager.
2. Find the section named **[Connexion]** and add this line:

   ```
   ImportCounterCache=1
   ```

3. Repeat this procedure on each client computer.

> **Note:** This step is absolutely mandatory.

Oracle

The proposed solution is to modify the **UP_GETCOUNTERVALUE** stored procedure.

This procedure will improve performance in wizards or external applications calling Asset Manager via APIs. Specifically, it will speed up handling of large transactions involving the insertion of records/objects which need default values taken from counters. On large transactions with several inserts needing default values from counters, counters may lock or deadlock (especially if several transactions are played at the same time, all needing a new value for the same counter). Current lock contention on counters can be monitored through a SQL query on the **v$session** view (when connected as an administrator) with the following statement:

```
Select username, machine, sid, serial# from v$session where lockwait != Null;
```

If this returns a large number of locks, then you may benefit from the workaround listed below. The purpose of the workaround is to replace counters whenever possible by sequences (which are non-locking mechanisms).

> **Note:** This solution eliminates your ability to query counter values through Asset Manager (the Tools/ Administration/ Counters menu item will display a screen with out-of-date values).

Step 1 - Preparation

1. Before you begin, make sure no one is using the database (or at least no one is doing inserts using

counters). This is very important, otherwise the sequences you generate will not have the correct numbers.

2. Before altering the database, make sure you have a backup and tested copy of the database.

Step 2 - Replace the counters

1. Connect to SQLPlus as the Asset Manager database owner.
2. Identify the counters to be replaced by sequences with the following query:

   ```
   Select * from amcounter;
   ```

3. Move the counters into sequences:

   ```
   SET CHARWIDTH 200;
   SET PAGESIZE 9999;
   SET HEADING OFF;
   SET ECHO OFF;
   Spool seqgen.sql
   SELECT 'CREATE SEQUENCE ' || IDENTIFIER || ' INCREMENT BY 1 NOMAXVALUE MINVALUE
   '|| (LVALUE + 1) || ' NOCYCLE CACHE 20 ORDER;' AS TEXT FROM AMCOUNTER where
   identifier IS NOT NULL;
   Spool off
   ```

   This yields a result similar to the following:

   ```
   CREATE SEQUENCE amEmplDept_BarCode INCREMENT BY 1 NOMAXVALUE MINVALUE 60155
   NOCYCLE CACHE 20 ORDER;  CREATE SEQUENCE amLocation_BarCode INCREMENT BY 1
   NOMAXVALUE MINVALUE 31 NOCYCLE CACHE 20 ORDER;  CREATE SEQUENCE amCategory_
   BarCode INCREMENT BY 1 NOMAXVALUE MINVALUE 101 NOCYCLE CACHE 20 ORDER;  CREATE
   SEQUENCE amProduct_CatalogRef INCREMENT BY 1 NOMAXVALUE MINVALUE 34076 NOCYCLE
   CACHE 20 ORDER; ...
   CREATE SEQUENCE amOutputEvent_EventNo INCREMENT BY 1 NOMAXVALUE MINVALUE 11
   NOCYCLE CACHE 20 ORDER;
   49 rows selected.
   ```

4. The **seqgen.sql** file can then be then edited to keep only the CREATE SEQUENCE statements.

Step 3 - Update the stored procedure

Replace the stored procedure **UP_GETCOUNTERVAL** to take advantage of sequences where possible.

1. Execute a basic statement to generate most of the sequence calls into the stored procedure. For example:

```
spool UP_GETCOUNTERVAL_gen.SQL
SET CHARWIDTH 200;
SELECT 'ELSIF CounterName = '||CHR(39)|| IDENTIFIER ||CHR(39)||' THEN '
 || ' SELECT '||IDENTIFIER||'.NEXTVAL INTO CounterValue FROM DUAL;'
FROM AMCOUNTER;
spool off;
```

This will generate a statement like the following:

```
ELSIF CounterName = '' THEN  SELECT .NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amEmplDept_BarCode' THEN  SELECT amEmplDept_
BarCode.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amLocation_BarCode' THEN  SELECT amLocation_
BarCode.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amCategory_BarCode' THEN  SELECT amCategory_
BarCode.NEXTVAL INTO CounterValue FROM DUAL;
...
```

2. Make the following modifications to the generated statement:

   a. Change the generated statement to properly handle IF Blocks.

   b. Remove the line with that deals with the empty counter:

   ```
   ELSIF CounterName = '' THEN  SELECT .NEXTVAL INTO CounterValue FROM DUAL;
   ```

   c. Add the declare and end part of the UP_GETCOUNTERVAL stored procedure.

   Your final statement should look like the following example:

   ```
   CREATE OR REPLACE PROCEDURE UP_GETCOUNTERVAL (CounterName IN VARCHAR2,
   CounterIncrement IN NUMBER, CounterValue OUT NUMBER) AS
     CounterIdent NUMBER;
   BEGIN
   IF CounterIncrement <> 1 THEN
     RAISE_APPLICATION_ERROR (-20004, 'Error in counter increment with the Up_
   getcounterval stored procedure altered to minimize locking, You must use a
   counter increment of 1, please add the importcountercache=1 in the amdb.ini
   file, in the parameters of your current database connection');
   ELSIF CounterName = 'amEmplDept_BarCode' THEN  SELECT amEmplDept_
   BarCode.NEXTVAL INTO CounterValue FROM DUAL;
   ELSIF CounterName = 'amLocation_BarCode' THEN  SELECT amLocation_
   BarCode.NEXTVAL INTO CounterValue FROM DUAL;
   ELSIF CounterName = 'amCategory_BarCode' THEN  SELECT amCategory_
   BarCode.NEXTVAL INTO CounterValue FROM DUAL;
   ELSIF CounterName = 'amProduct_CatalogRef' THEN  SELECT amProduct_
   CatalogRef.NEXTVAL INTO CounterValue FROM DUAL;
   ELSIF CounterName = 'amProduct_BarCode' THEN  SELECT amProduct_
   BarCode.NEXTVAL INTO CounterValue FROM DUAL;
   ELSIF CounterName = 'amProdCompo_ItemNo' THEN  SELECT amProdCompo_
   ItemNo.NEXTVAL INTO CounterValue FROM DUAL;
   ```

```
ELSIF CounterName = 'amAsset_AssetTag' THEN  SELECT amAsset_AssetTag.NEXTVAL
INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amConnection_CnxNo' THEN  SELECT amConnection_
CnxNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amCompany_Code' THEN  SELECT amCompany_Code.NEXTVAL
INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amBudget_BudgetNo' THEN  SELECT amBudget_
BudgetNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amContract_Ref' THEN  SELECT amContract_Ref.NEXTVAL
INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amConsUse_ItemNo' THEN  SELECT amConsUse_ItemNo.NEXTVAL
INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amStock_Code' THEN  SELECT amStock_Code.NEXTVAL INTO
CounterValue FROM DUAL;
ELSIF CounterName = 'amProdReserv_ItemNo' THEN  SELECT amProdReserv_
ItemNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amProject_Code' THEN  SELECT amProject_Code.NEXTVAL
INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amTicket_TicketNo' THEN  SELECT amTicket_
TicketNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amWorkOrder_WONo' THEN  SELECT amWorkOrder_WONo.NEXTVAL
INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amNews_Name' THEN  SELECT amNews_Name.NEXTVAL INTO
CounterValue FROM DUAL;
ELSIF CounterName = 'amKnowlBase_Code' THEN  SELECT amKnowlBase_Code.NEXTVAL
INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amDecisionTree_Code' THEN  SELECT amDecisionTree_
Code.NEXTVAL INTO CounterValue FROM DUAL
ELSIF CounterName = 'amDownTimePeriod_Code' THEN  SELECT amDownTimePeriod_
Code.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amRequest_ReqNumber' THEN  SELECT amRequest_
ReqNumber.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amReqLine_ItemNo' THEN  SELECT amReqLine_ItemNo.NEXTVAL
INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amEstimate_EstimNumber' THEN  SELECT amEstimate_
EstimNumber.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amEstimLine_ItemNo' THEN  SELECT amEstimLine_
ItemNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amPOrder_PONumber' THEN  SELECT amPOrder_
PONumber.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amPOrdLine_ItemNo' THEN  SELECT amPOrdLine_
ItemNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amDeliv_DelivNumber' THEN  SELECT amDeliv_
DelivNumber.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amItemReceived_ItemNo' THEN  SELECT amItemReceived_
ItemNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amItemReturned_ItemNo' THEN  SELECT amItemReturned_
```

```
ItemNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amInvoice_InvoiceNumber' THEN  SELECT amInvoice_
InvoiceNumber.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amAdjustment_ItemNo' THEN  SELECT amAdjustment_
ItemNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amFieldAdjust_ItemNo' THEN  SELECT amFieldAdjust_
ItemNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amFieldAdjustModel_ItemNo' THEN  SELECT
amFieldAdjustModel_ItemNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amReturnEnv_Code' THEN  SELECT amReturnEnv_Code.NEXTVAL
INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amLoan_Code' THEN  SELECT amLoan_Code.NEXTVAL INTO
CounterValue FROM DUAL;
ELSIF CounterName = 'amCostCenter_Code' THEN  SELECT amCostCenter_
Code.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amExpenseLine_ItemNo' THEN  SELECT amExpenseLine_
ItemNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amWfScheme_Ref' THEN  SELECT amWfScheme_Ref.NEXTVAL
INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amWfOrgRole_Ref' THEN  SELECT amWfOrgRole_Ref.NEXTVAL
INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amWfActivity_Ref' THEN  SELECT amWfActivity_Ref.NEXTVAL
INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amWfActivAlarm_Ref' THEN  SELECT amWfActivAlarm_
Ref.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amWfEvent_Ref' THEN  SELECT amWfEvent_Ref.NEXTVAL INTO
CounterValue FROM DUAL;
ELSIF CounterName = 'amWfTransition_Ref' THEN  SELECT amWfTransition_
Ref.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amWfWorkItem_Ref' THEN  SELECT amWfWorkItem_Ref.NEXTVAL
INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amWfSyncPoint_Ref' THEN  SELECT amWfSyncPoint_
Ref.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amWfInstance_Ref' THEN  SELECT amWfInstance_Ref.NEXTVAL
INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amInputEvent_EventNo' THEN  SELECT amInputEvent_
EventNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amOutputEvent_EventNo' THEN  SELECT amOutputEvent_
EventNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSE
       SELECT MAX(lCounterId) INTO CounterIdent FROM amCounter WHERE Identifier =
CounterName;
       IF CounterIdent IS NULL THEN
              SELECT MAX(lCounterId)+1 INTO CounterIdent FROM amCounter;
              INSERT INTO amCounter (lCounterId, Identifier, Description, lValue,
dtLastModif) VALUES (CounterIdent, CounterName, CounterName,
CounterIncrement, SYSDATE);
```

```
        ELSE
                UPDATE amCounter SET lValue = lValue + CounterIncrement, dtLastModif =
    SYSDATE WHERE lCounterId = CounterIdent;
        END IF;
        SELECT lValue INTO CounterValue FROM amCounter WHERE Identifier =
    CounterName;

    Commit;
    END IF;
    END;
    /
```

3. Make a backup of the original **UP_GETCOUNTERVAL** stored procedure.
4. Update the stored procedure using the script previously generated.

Step 4 - Modify the stored procedure UP_GETID

Locking on ID generation may also occur sometimes. To minimize locking, modify the stored procedure **UP_GETID** as follows:

```
create or replace procedure UP_GETID(lSeedId OUT NUMBER) AS PRAGMA AUTONOMOUS_
TRANSACTION;
v_id NUMBER DEFAULT 0;
v_count INT DEFAULT 0;
BEGIN
Lock table LastId IN EXCLUSIVE MODE;
select count(*) into v_count from LastId where lRemain = 0;
IF(v_count > 20) THEN
       delete from LastId where lRemain = 0;
END IF;
select Min(IdSeed) into v_id from LastId where lRemain > 0 and lInUse = 0;
if (v_id is null or v_id = 0) then
       select LastId_autoid.nextval into lSeedId from dual;
       insert into LastId(IdSeed, value, lRemain, lInUse) values(lSeedId, 1, 32, 1);
else
       update LastId set lInUse = 1 where IdSeed = v_id;
       lSeedId := v_id;
end if;
commit;
END;
```

Step 5 - Modify the stored procedure UP_GETINDEPID

Locking on independent ID generation may also occur sometimes. To minimize locking, modify the stored procedure **UP_GETINDEPID** as follows :

```
create or replace procedure UP_GETINDEPID(lSeedId OUT NUMBER) AS PRAGMA AUTONOMOUS_
TRANSACTION;
v_id NUMBER DEFAULT 0;
v_count INT DEFAULT 0;
BEGIN
Lock table IndependentLastId IN EXCLUSIVE MODE;
select count(*) into v_count from IndependentLastId where lRemain = 0;
IF(v_count > 20) THEN
       delete from IndependentLastId where lRemain = 0;
END IF;
select Min(IdSeed) into v_id from IndependentLastId where lRemain > 0 and lInUse =
0;
if (v_id is null or v_id = 0) then
       select IndependentLastId_autoid.nextval into lSeedId from dual;
       insert into IndependentLastId(IdSeed, value, lRemain, lInUse) values(lSeedId, 1,
32, 1);
else
       update IndependentLastId set lInUse = 1 where IdSeed = v_id;
       lSeedId := v_id;
end if;
commit;
END;
```

Step 6 - Edit the **amdb.ini** file

1. Open the **amdb.ini** file of Asset Manager.
2. Find the section named **[Connexion]** and add this line:

   ```
   ImportCounterCache=1
   ```

3. Repeat this procedure on each client computer.

> **Note:** This step is absolutely mandatory.

DB2

The proposed solution is to modify the **UP_GETCOUNTERVALUE** stored procedure.

This procedure will improve performance in wizards or external applications calling Asset Manager via APIs. Specifically, it will speed up handling of large transactions involving the insertion of records/objects that need default values from counters.

If this returns a large number of locks, then you may benefit from the workaround listed below. The purpose of the workaround is to replace counters whenever possible by sequences (which are non-locking mechanisms).

> **Note:** This solution eliminates your ability to query counter values through Asset Manager (the Tools/ Administration/ Counters menu item will display a screen with out-of-date values).

Step 1 - Preparation

1. Before you begin, make sure no one is using the database (or at least is doing inserts using counters). This is very important, otherwise the sequences you generate will not have the correct numbers.
2. Before altering the database, make sure you have a backup and tested copy of the database.

Step 2 - Replace the counters

1. Connect to CLP as the Asset Manager database owner.
2. Identify the counters to be replaced by sequences with the following query:

   ```
   Select * from amcounter;
   ```

3. Move the counters into sequences:

   ```
   SELECT 'CREATE SEQUENCE ' || IDENTIFIER || ' INCREMENT BY 1 NOMAXVALUE START
   WITH '|| CHAR(LVALUE + 1) || ' NOCYCLE CACHE 20 ORDER' AS TEXT FROM AMCOUNTER
   where identifier IS NOT NULL;
   ```

   This will generate a statement like the following:

   ```
   CREATE SEQUENCE amInputEvent_EventNo INCREMENT BY 1 NOMAXVALUE START WITH 1001
   NOCYCLE CACHE 20 ORDER
   CREATE SEQUENCE amOutputEvent_EventNo INCREMENT BY 1 NOMAXVALUE START WITH 1001
   NOCYCLE CACHE 20 ORDER
   CREATE SEQUENCE amWfActivAlarm_Ref INCREMENT BY 1 NOMAXVALUE START WITH 1001
   NOCYCLE CACHE 20 ORDER
   CREATE SEQUENCE amWfActivity_Ref INCREMENT BY 1 NOMAXVALUE START WITH 1005
   NOCYCLE CACHE 20 ORDER
   CREATE SEQUENCE amWfEvent_Ref INCREMENT BY 1 NOMAXVALUE START WITH 1011 NOCYCLE
   CACHE 20 ORDER
   CREATE SEQUENCE amWfInstance_Ref INCREMENT BY 1 NOMAXVALUE START WITH 1029
   NOCYCLE CACHE 20 ORDER
   CREATE SEQUENCE amWfOrgRole_Ref INCREMENT BY 1 NOMAXVALUE START WITH 1001
   NOCYCLE CACHE 20 ORDER
   CREATE SEQUENCE amWfScheme_Ref INCREMENT BY 1 NOMAXVALUE START WITH 1010
   ```

```
NOCYCLE CACHE 20 ORDER
CREATE SEQUENCE amWfTransition_Ref INCREMENT BY 1 NOMAXVALUE START WITH 1009
NOCYCLE CACHE 20 ORDER
CREATE SEQUENCE amWfWorkItem_Ref INCREMENT BY 1 NOMAXVALUE START WITH 1029
NOCYCLE CACHE 20 ORDER
CREATE SEQUENCE amCable_CableTag INCREMENT BY 1 NOMAXVALUE START WITH 1001
NOCYCLE CACHE 20 ORDER
CREATE SEQUENCE amCableLink_BarCode INCREMENT BY 1 NOMAXVALUE START WITH 1001
NOCYCLE CACHE 20 ORDER
CREATE SEQUENCE amColorCode_Code INCREMENT BY 1 NOMAXVALUE START WITH 1001
NOCYCLE CACHE 20 ORDER
CREATE SEQUENCE amTraceHistory_BarCode INCREMENT BY 1 NOMAXVALUE START WITH
1001 NOCYCLE CACHE 20 ORDER
......
73      rows selected.
```

4. The **seqgen.sql** file can then be then edited to keep only the CREATE SEQUENCE statements.

Step 3 - Update the stored procedure

Replace the stored procedure **UP_GETCOUNTERVAL** to take advantage of sequences where possible.

1. Execute a basic statement to generate most of the sequence calls into the stored procedure. For example:

   ```
   db2 => SELECT 'ELSEIF CounterName = '||CHR(39)|| IDENTIFIER ||CHR(39)||' THEN
   Values NEXTVAL FOR '||IDENTIFIER||' INTO CounterValue ' FROM AMCOUNTER
   ```

   This will generate a statement like the following:

   ```
   ELSEIF CounterName = 'amInputEvent_EventNo' THEN Values NEXTVAL FOR
   amInputEvent_EventNo INTO CounterValue;
   ELSEIF CounterName = 'amOutputEvent_EventNo' THEN Values NEXTVAL FOR
   amOutputEvent_EventNo INTO CounterValue;
   ELSEIF CounterName = 'amWfActivAlarm_Ref' THEN Values NEXTVAL FOR
   amWfActivAlarm_Ref INTO CounterValue;
   ELSEIF CounterName = 'amWfActivity_Ref' THEN Values NEXTVAL FOR amWfActivity_
   Ref INTO CounterValue;
   ELSEIF CounterName = 'amWfEvent_Ref' THEN Values NEXTVAL FOR amWfEvent_Ref INTO
   CounterValue;
   ELSEIF CounterName = 'amWfInstance_Ref' THEN Values NEXTVAL FOR amWfInstance_
   Ref INTO CounterValue;
   ELSEIF CounterName = 'amWfOrgRole_Ref' THEN Values NEXTVAL FOR amWfOrgRole_Ref
   INTO CounterValue;
   ELSEIF CounterName = 'amWfScheme_Ref' THEN Values NEXTVAL FOR amWfScheme_Ref
   INTO CounterValue;
   ```

```
ELSEIF CounterName = 'amWfTransition_Ref' THEN Values NEXTVAL FOR
amWfTransition_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfWorkItem_Ref' THEN Values NEXTVAL FOR amWfWorkItem_
Ref INTO CounterValue;
ELSEIF CounterName = 'amCable_CableTag' THEN Values NEXTVAL FOR amCable_
CableTag INTO CounterValue;
ELSEIF CounterName = 'amCableLink_BarCode' THEN Values NEXTVAL FOR amCableLink_
BarCode INTO CounterValue;
ELSEIF CounterName = 'amColorCode_Code' THEN Values NEXTVAL FOR amColorCode_
Code INTO CounterValue;
ELSEIF CounterName = 'amTraceHistory_BarCode' THEN Values NEXTVAL FOR
amTraceHistory_BarCode INTO CounterValue;
ELSEIF CounterName = 'amCatalog_Code' THEN Values NEXTVAL FOR amCatalog_Code
INTO CounterValue;
ELSEIF CounterName = 'InternalRef' THEN Values NEXTVAL FOR InternalRef INTO
CounterValue;
ELSEIF CounterName = 'amContract_Ref' THEN Values NEXTVAL FOR amContract_Ref
INTO CounterValue;
       .....
```

2. Make the following modifications to the generated statement:

   a. Change the generated statement to properly handle IF Blocks.

   b. Remove the line with that deals with the empty counter:

   c. Add the declare and end part of the UP_GETCOUNTERVAL stored procedure.

      Your final statement should look like the following example:

```
CREATE PROCEDURE UP_GETCOUNTERVAL (IN CounterName VARCHAR(64), IN
CounterIncrement INTEGER, OUT CounterValue INTEGER) LANGUAGE SQL BEGIN
DECLARE CounterIdent INTEGER; DECLARE ERR_MSG VARCHAR(255);
IF CounterIncrement <> 1 THEN
   SELECT 'Up_getcounterval stored procedure altered to minimize locking,
You must use a counter increment of 1, please add the importcountercache=1
in the amdb.ini file, in the parameters of your current database connection'
INTO ERR_MSG FROM SYSIBM.SYSDUMMY1;
   SIGNAL SQLSTATE '75002' SET MESSAGE_TEXT = ERR_MSG;
ELSEIF CounterName = 'amInputEvent_EventNo' THEN Values NEXTVAL FOR
amInputEvent_EventNo INTO CounterValue;
ELSEIF CounterName = 'amOutputEvent_EventNo' THEN Values NEXTVAL FOR
amOutputEvent_EventNo INTO CounterValue;
ELSEIF CounterName = 'amWfActivAlarm_Ref' THEN Values NEXTVAL FOR
amWfActivAlarm_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfActivity_Ref' THEN Values NEXTVAL FOR
amWfActivity_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfEvent_Ref' THEN Values NEXTVAL FOR amWfEvent_Ref
INTO CounterValue;
ELSEIF CounterName = 'amWfInstance_Ref' THEN Values NEXTVAL FOR
```

```
amWfInstance_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfOrgRole_Ref' THEN Values NEXTVAL FOR amWfOrgRole_
Ref INTO CounterValue;
ELSEIF CounterName = 'amWfScheme_Ref' THEN Values NEXTVAL FOR amWfScheme_Ref
INTO CounterValue;
ELSEIF CounterName = 'amWfTransition_Ref' THEN Values NEXTVAL FOR
amWfTransition_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfWorkItem_Ref' THEN Values NEXTVAL FOR
amWfWorkItem_Ref INTO CounterValue;
ELSEIF CounterName = 'amCable_CableTag' THEN Values NEXTVAL FOR amCable_
CableTag INTO CounterValue;
ELSEIF CounterName = 'amCableLink_BarCode' THEN Values NEXTVAL FOR
amCableLink_BarCode INTO CounterValue;
ELSEIF CounterName = 'amColorCode_Code' THEN Values NEXTVAL FOR amColorCode_
Code INTO CounterValue;
ELSEIF CounterName = 'amTraceHistory_BarCode' THEN Values NEXTVAL FOR
amTraceHistory_BarCode INTO CounterValue;
ELSEIF CounterName = 'amCatalog_Code' THEN Values NEXTVAL FOR amCatalog_Code
INTO CounterValue;
ELSEIF CounterName = 'InternalRef' THEN Values NEXTVAL FOR InternalRef INTO
CounterValue;
ELSEIF CounterName = 'amContract_Ref' THEN Values NEXTVAL FOR amContract_Ref
INTO CounterValue;
 ELSE SELECT MAX(lCounterId) INTO CounterIdent FROM amCounter WHERE
Identifier = CounterName;
 IF CounterIdent IS NULL THEN SELECT MAX(lCounterId)+1 INTO CounterIdent
FROM amCounter;
 INSERT INTO amCounter (lCounterId, Identifier, Description, lValue,
dtLastModif) VALUES (CounterIdent, CounterName, CounterName,
CounterIncrement, CURRENT TIMESTAMP);
 ELSE UPDATE amCounter SET lValue = lValue + CounterIncrement, dtLastModif =
CURRENT TIMESTAMP WHERE lCounterId =CounterIdent;
 END IF;
 SELECT lValue INTO CounterValue FROM amCounter WHERE Identifier =
CounterName;
 COMMIT;
 END IF;
 END
GO
```

3. Make a backup of the original **UP_GETCOUNTERVAL** stored procedure.
4. Update the stored procedure using the script previously generated.

Step 4 - Modify the stored procedure UP_GETID

Locking on ID generation may also occur sometimes. To minimize locking, modify the stored procedure

UP_GETID as follows:

```
drop procedure UP_GETID;
create procedure UP_GETID (OUT lSeedId INTEGER)  LANGUAGE SQL BEGIN
  DECLARE v_id INT DEFAULT 0;
  DECLARE v_count INT DEFAULT 0;
  LOCK TABLE LastId IN EXCLUSIVE MODE;
  select count(*) into v_count from LastId where lRemain = 0;
  IF(v_count > 20) THEN delete from LastId where lRemain = 0; END IF;
  select Min(IdSeed) into v_id from LastId where lRemain > 0 and lInUse = 0;
  if (v_id is null or v_id = 0) then
    insert into LastId(value, lRemain, lInUse) values(1, 32, 1);
    select Max(IdSeed) into v_id from LastId; set lSeedId = v_id;
  else
    update LastId set lInUse = 1 where IdSeed = v_id;
    set lSeedId = v_id;
  end if;
end;
```

Step 5 - Modify the stored procedure UP_GETINDEPID

Locking on independent ID generation may also occur sometimes. To minimize locking, modify the stored procedure **UP_GETINDEPID** as follows :

```
drop procedure UP_GETINDEPID;
create procedure UP_GETINDEPID (OUT lSeedId INTEGER)  LANGUAGE SQL BEGIN
  DECLARE v_id INT DEFAULT 0;
  DECLARE v_count INT DEFAULT 0;
  LOCK TABLE IndependentLastId IN EXCLUSIVE MODE;
  select count(*) into v_count from IndependentLastId where lRemain = 0;
  IF(v_count > 20) THEN delete from IndependentLastId where lRemain = 0; END IF;
  select Min(IdSeed) into v_id from IndependentLastId where lRemain > 0 and lInUse
= 0;
  if (v_id is null or v_id = 0) then
    insert into IndependentLastId(value, lRemain, lInUse) values(1, 32, 1);
    select Max(IdSeed) into v_id from IndependentLastId;
    set lSeedId = v_id;
  else
    update IndependentLastId set lInUse = 1 where IdSeed = v_id;
    set lSeedId = v_id;
  end if;
end;
```

Step 4 - Edit the **amdb.ini** file

1. Open the **amdb.ini** file of Asset Manager.

2. Find the section named **[Connexion]** and add this line:

   ```
   ImportCounterCache=1
   ```

3. Repeat this procedure on each client computer.

> **Note:** This step is absolutely mandatory.

Send documentation feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Tuning (Asset Manager 9.61)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to ovdoc-ITSM@hpe.com.

We appreciate your feedback!