Product Documentation

Peregrine

AssetCenter

Tuning

Part No. DAC-441-EN35

Peregrine SYSTEMS®

Build 17

If you need technical support for this product, or would like to request documentation for a product for which you are licensed, contact Peregrine Systems, Inc. Customer Support by email at support@peregrine.com.

If you have comments or suggestions about this documentation, contact Peregrine Systems, Inc. Technical Publications by email at doc_comments@peregrine.com.

This edition applies to version 4.4 of the licensed program

AssetCenter

Peregrine Systems, Inc.
3611 Valley Centre Drive San Diego, CA 92130
858.481.5000
Fax 858.481.1751
www.peregrine.com

PEREGRINE

Table of Contents

PEREGRINE

Introduction

What is the aim of this guide?

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

- the network,
- the database server,
- the AssetCenter client,
- the database engine

Who is this guide intended for?

This guide is mainly intended for AssetCenter 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.

Warning:

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 here are the only tuning options available. Also, Peregrine Systems, Inc is not responsible for any damages or data loss incurred as a result of improper use of the information provided herein. If you don't understand something, don't do it. As always, Peregrine recommends having a current backup before you make any major changes in your implementation.

Reminder concerning AssetCenter architecture

AssetCenter relies on a two-tier architecture. The AssetCenter client connects directly to the database and falls into the category of what is usually called 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 the key components of AssetCenter is AssetCenter Server. This server does not exchange any data with the client. It connects to the database separately, and perform all of the background tasks that require regular triggering and/or monitoring. For example, if you have an alarm set up on your contract end date to send you an email when the contract expires in less than 30 days, the AssetCenter Server is what sends that email out at the appropriate time. It also executes server-process workflows, calculates rents, and - most importantly - validates the authorization code in the database. Most of the tuning suggestions being made in this document will not apply to the AssetCenter Server.

Bottlenecks

There are basically four areas of AssetCenter 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 AssetCenter 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 because it takes a long time for the database to respond). Database server bottlenecks occur when the hardware that the database server is running on is insufficient for the implementation of AssetCenter. Typically this is not a problem area because most implementations analyze their requirements in advance and make sure 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.

Contents of this guide

Chapter Tuning a client

This chapter explains how to tune client computers.

Chapter Tuning the database

This chapter deals with tuning the database.

Conventions used in this guide

What follows is a list of the conventions that we use in this guide

Convention	Description
`Java Script Code`	Example of the code or command
`Fixed-width characters`	DOS command, function parameter or data format
...	Portion of omitted code or command.
Note: Extra information	Informative note
Important: Be careful...	Important information for the user
Tip: User tip	Tip to help you use the application
Warning: Exercise caution	Extremely important information for the user
Object	AssetCenter interface object: menu, menu entry, tab or button.

The following conventions are also used:

- The steps that we ask you to follow in a specific order are shown as a numbered list. For example:
  1. First step
  2. Second step
  3. Third and last step
- All the figures and the tables are numbered according to the chapter in which they are found, and the order in which the appear in the chapter. For example, the title of the fourth table in chapter two will be prefixed by **Table 2-4**.

CHAPTER 1 Tuning a client

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

Network latency

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

Tree view and table view

AssetCenter enables you to view records in two distinct modes:

- A tree view of the records. This view shows you the hierarchy of the table, 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)**, etc.

- 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 AssetCenter to **Yes**. (**Edit/ Options/ Lists/ Main lists** and **Edit/ Options/ Lists/ Other lists**).

List-loading mechanism

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

1. AssetCenter 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.FullNa
me 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 AssetCenter. 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.Tit
le, E1.Phone, E1.Fax, E1.FullName FROM amEmplDept E1 ORDER BY E1.lEmplDept
Id
```

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

Loading lists

Several options are available to define the number of records initially loaded in a list. By reducing or limiting the this number, you can improve 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 AssetCenter tries to populate the list. When this time is overrun, the operation is suspended and the list if 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 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 **ArrayFetchingSize** parameter in AssetCenter. This parameter specifies the number of records recovered by AssetCenter 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 **ArrayFetchingSize** parameter must be set to 201. This incrementation accounts for the NULL record, which is returned in the results of the query but not displayed by AssetCenter.

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

1 Load the **amdb.ini** file in a text editor,

2 In the section corresponding to the declaration of the database connection, find the **ArrayFetchingSize** parameter. The following is an example for the database **ACDemo432en**:

```
[ACDemo432en]
Engine=SQLAnywhere
Location=ACDemo432en
EngineLogin=itam
EnginePassword=7BC2423F1B1F6A42E2B3A27E396F52C3A9AD6828582AED88DAC2F
53E9A369BC03E6393911903124254200200
ReadOnly=0
CacheSize=5120000
ArrayFetchingSize=201
AmApiDll=C:\PROGRA~1\PEREGR~1\ASSETC~1\ASSETC~2\bin\aamapi43.dll
```

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

Note:

Changing this parameter may also prove useful when AssetCenter is used over a WAN (Wide Area Network).

Type-ahead

By default, AssetCenter 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 returns 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 expensive. 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 located 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

Access restrictions are part of an AssetCenter profile. It basically translates into a filter on the records of a table. For example they can be used to prevent a technician from seeing any portfolio items outside of their 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, lets take 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.lS
upervId AND amEmplDept_CurrentUser.lEmplDeptId = 25323 ORDER BY C1.lCos
tId
```

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 AssetCenter, 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, AssetCenter 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.

Important:

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 AssetCenter Database Administrator.

The display properties

The AssetCenter database objects, fields and links in particular, have properties that determine how they are displayed. These properties are the following:

- **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

AssetCenter 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 (i.e. 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 previously mentioned, the **Keep history** property may cause performance issues if a complex script (querying the database) is associated with it. Another impact to performance is directly linked to the fact that an insert statement is sent to the database for each historized object. In ideal cases, 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 those fields you absolutely have to. 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 AssetCenter 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 AssetCenter introduced prior to 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, etc.

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 AssetCenter 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 Peregrine Systems 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 with 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 AssetCenter client are due to poor list configuration choices. When a user opens a screen, AssetCenter executes a query to pull in all of the list data. The query is dynamically built based on several settings (the columns in the list, sorts, filters, etc.), read access restrictions, table/tree view, etc.

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, that creates a new join 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 AssetCenter. 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 (e.g. **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.

CHAPTER 2 Tuning the database

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

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.

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's 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 is able to 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 have 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 it's done. Ideally, you want the fastest hard disks you can get, typically SCSI drives, although IDE drives have come a long way in the past few years. You also may want to consider a RAID setup. RAID stands for Redundant Array of Independent Disks. It's 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 it's 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 AssetCenter queries involve reading 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 queries

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

1. AssetCenter builds the query and sends it to the database, but any parameters are not yet sent. 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 AssetCenter. 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**, then it will be able to determine an appropriate access plan (such as using the index on **lEmplDeptId**).
3. AssetCenter 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 AssetCenter.

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 suggestions 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. AssetCenter 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 regularly. This can be done by either your DBA as a regular database maintenance task, or by AssetCenter Server as one of its modules. Peregrine Systems recommends letting AssetCenter Server do this once a week. What calculating statistics does, in a basic sense, is gather information about the frequency that data is used. This often allows the optimizer to improve access to frequently used data.
- 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's not a lot of data being added or deleted, then you can probably do this once a quarter. Most likely, though, you should have the DBA perform this once a month a part of the regular database maintenance.
- If you're seeing poor performance overall, one thing you may want to look for is to see 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), then 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.

Warning:

This information is intended for experienced database administrators.

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 - 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 AssetCenter 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 2 - Backup

Backup the stored procedure **up_GetCounterVal**.

Step 3 - 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)  O
N [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 4 - 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 sup
ported when modified for non-locking optimization. Please edit the AMDB.IN
I file on your computer and add ImportCounterCache=1 in the current databa
se connexion description'
GO
CREATE PROCEDURE up_GetCounterVal @CounterName varchar(64), @CounterIncrem
ent 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', 'am
Ticket_TicketNo', 'amExpenseLine_ItemNo')
  BEGIN
    DECLARE amCounterLock CURSOR FOR SELECT lCounterId FROM amCounter WHER
E 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, @Coun
terIncrement, GetDate() FROM amCounter
    ELSE
    BEGIN
        OPEN amCounterLock
        UPDATE amCounter SET lValue = lValue + @CounterIncrement, dtLastMo
dif = 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 lu
serspid=@@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 lus
erspid=@@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) wher
e luserspid=@@SPID
    END
-- end of generation sql below
  END
END
GO
```

Tip:

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

```
select '''cnt_'+Identifier+''', ' as sqlquery from amCounter where lCounte
rid<>0
```

and

```
select '  IF @CounterName =''cnt_'+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 5 - 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 6 - 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
insert into LastId (Value)  values(@@SPID)
return 1+(SELECT max(IdSeed) FROM lastid  WITH (NOLOCK) where value=@@SPID
)*30
/*
create procedure up_GetId as   if (select count(*) from LastId WITH (READU
NCOMMITTED) )>20 delete from LastId  insert into LastId (Value) values(@@S
PID)  return 1+@@IDENTITY*30
*/
GO
```

Step 7 - 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 f
rom 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 th
e 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 recommend
ed).
--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 8 - Edit the amdb.ini file

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

```
ImportCounterCache=1
```

3. Repeat this procedure on each client computer.

Important:

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 AssetCenter via APIs. Specifically, it will speed handling of large transactions involving the insertion of records/objects which need default values taken of 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 != Nu
ll;
```

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 AssetCenter (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 SQLPlus as the AssetCenter 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 AM
COUNTER where identifier IS NOT NULL;

Spool off
```

This yields a result similar to the following:

```
CREATE SEQUENCE amEmplDept_BarCode INCREMENT BY 1 NOMAXVALUE MINVALUE 6
0155 NOCYCLE CACHE 20 ORDER;  CREATE SEQUENCE amLocation_BarCode INCREM
ENT BY 1 NOMAXVALUE MINVALUE 31 NOCYCLE CACHE 20 ORDER;  CREATE SEQUENC
E amCategory_BarCode INCREMENT BY 1 NOMAXVALUE MINVALUE 101 NOCYCLE CAC
HE 20 ORDER;  CREATE SEQUENCE amProduct_CatalogRef INCREMENT BY 1 NOMAX
VALUE MINVALUE 34076 NOCYCLE CACHE 20 ORDER; ...
CREATE SEQUENCE amOutputEvent_EventNo INCREMENT BY 1 NOMAXVALUE MINVALU
E 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 DUA
L;
ELSIF CounterName = 'amEmplDept_BarCode' THEN  SELECT amEmplDept_BarCod
e.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amLocation_BarCode' THEN  SELECT amLocation_BarCod
e.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amCategory_BarCode' THEN  SELECT amCategory_BarCod
e.NEXTVAL INTO CounterValue FROM DUAL;
...
```

2 Make the following modifications to the generated statement:
   1 Change the generated statement to properly handle IF Blocks.
   2 Remove the line with that deals with the empty counter:

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

3 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 VARCHAR
2, CounterIncrement IN NUMBER, CounterValue OUT NUMBER) AS
CounterIdent NUMBER;
BEGIN
IF CounterIncrement <> 1 THEN
RAISE_APPLICATION_ERROR (-20004, 'Error in counter increment with th
e 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_Bar
Code.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amLocation_BarCode' THEN  SELECT amLocation_Bar
Code.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amCategory_BarCode' THEN  SELECT amCategory_Bar
Code.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amProduct_CatalogRef' THEN  SELECT amProduct_Ca
talogRef.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amProduct_BarCode' THEN  SELECT amProduct_BarCo
de.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amProdCompo_ItemNo' THEN  SELECT amProdCompo_It
emNo.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_C
nxNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amCompany_Code' THEN  SELECT amCompany_Code.NEX
TVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amBudget_BudgetNo' THEN  SELECT amBudget_Budget
No.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amContract_Ref' THEN  SELECT amContract_Ref.NEX
TVAL 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.NEX
TVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amTicket_TicketNo' THEN  SELECT amTicket_Ticket
No.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 I
NTO CounterValue FROM DUAL;
ELSIF CounterName = 'amKnowlBase_Code' THEN  SELECT amKnowlBase_Code
.NEXTVAL INTO CounterValue FROM DUAL;
```

```
ELSIF CounterName = 'amDecisionTree_Code' THEN  SELECT amDecisionTre
e_Code.NEXTVAL INTO CounterValue FROM DUAL
ELSIF CounterName = 'amDownTimePeriod_Code' THEN  SELECT amDownTimeP
eriod_Code.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amRequest_ReqNumber' THEN  SELECT amRequest_Req
Number.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_It
emNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amPOrder_PONumber' THEN  SELECT amPOrder_PONumb
er.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amPOrdLine_ItemNo' THEN  SELECT amPOrdLine_Item
No.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amDeliv_DelivNumber' THEN  SELECT amDeliv_Deliv
Number.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amItemReceived_ItemNo' THEN  SELECT amItemRecei
ved_ItemNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amItemReturned_ItemNo' THEN  SELECT amItemRetur
ned_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 amFieldAdjus
t_ItemNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amFieldAdjustModel_ItemNo' THEN  SELECT amField
AdjustModel_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 I
NTO CounterValue FROM DUAL;
ELSIF CounterName = 'amCostCenter_Code' THEN  SELECT amCostCenter_Co
de.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amExpenseLine_ItemNo' THEN  SELECT amExpenseLin
e_ItemNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amWfScheme_Ref' THEN  SELECT amWfScheme_Ref.NEX
TVAL INTO CounterValue FROM DUAL;
ELSIF CounterName = 'amWfOrgRole_Ref' THEN  SELECT amWfOrgRole_Ref.N
EXTVAL 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.NEXTV
AL 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_R
ef.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 amOutputEve
nt_EventNo.NEXTVAL INTO CounterValue FROM DUAL;
ELSE
SELECT MAX(lCounterId) INTO CounterIdent FROM amCounter WHERE Identi
fier = 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, Counter
Increment, SYSDATE);
ELSE
UPDATE amCounter SET lValue = lValue + CounterIncrement, dtLastModif
= SYSDATE WHERE lCounterId = CounterIdent;
END IF;
SELECT lValue INTO CounterValue FROM amCounter WHERE Identifier = Co
unterName;
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 8 - Edit the `amdb.ini` file

1 Open the **`amdb.ini`** file of AssetCenter

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

```
ImportCounterCache=1
```

3 Repeat this procedure on each client computer.

Important:

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 AssetCenter 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 AssetCenter (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 AssetCenter 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 FR
OM AMCOUNTER where identifier IS NOT NULL;
```

This will generate a statement like the following:

```
CREATE SEQUENCE amInputEvent_EventNo INCREMENT BY 1 NOMAXVALUE START WI
TH 1001 NOCYCLE CACHE 20 ORDER
CREATE SEQUENCE amOutputEvent_EventNo INCREMENT BY 1 NOMAXVALUE START W
ITH 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 1
005 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 1
029 NOCYCLE CACHE 20 ORDER
CREATE SEQUENCE amWfOrgRole_Ref INCREMENT BY 1 NOMAXVALUE START WITH 10
01 NOCYCLE CACHE 20 ORDER
CREATE SEQUENCE amWfScheme_Ref INCREMENT BY 1 NOMAXVALUE START WITH 101
0 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 1
029 NOCYCLE CACHE 20 ORDER
CREATE SEQUENCE amCable_CableTag INCREMENT BY 1 NOMAXVALUE START WITH 1
001 NOCYCLE CACHE 20 ORDER
CREATE SEQUENCE amCableLink_BarCode INCREMENT BY 1 NOMAXVALUE START WIT
H 1001 NOCYCLE CACHE 20 ORDER
CREATE SEQUENCE amColorCode_Code INCREMENT BY 1 NOMAXVALUE START WITH 1
001 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 AMC
OUNTER
```

This will generate a statement like the following:

```
ELSEIF CounterName = 'amInputEvent_EventNo' THEN Values NEXTVAL FOR amI
nputEvent_EventNo INTO CounterValue;
ELSEIF CounterName = 'amOutputEvent_EventNo' THEN Values NEXTVAL FOR am
OutputEvent_EventNo INTO CounterValue;
ELSEIF CounterName = 'amWfActivAlarm_Ref' THEN Values NEXTVAL FOR amWfA
ctivAlarm_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfActivity_Ref' THEN Values NEXTVAL FOR amWfAct
ivity_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfEvent_Ref' THEN Values NEXTVAL FOR amWfEvent_
Ref INTO CounterValue;
ELSEIF CounterName = 'amWfInstance_Ref' THEN Values NEXTVAL FOR amWfIns
tance_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfOrgRole_Ref' THEN Values NEXTVAL FOR amWfOrgR
ole_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfScheme_Ref' THEN Values NEXTVAL FOR amWfSchem
e_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfTransition_Ref' THEN Values NEXTVAL FOR amWfT
ransition_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfWorkItem_Ref' THEN Values NEXTVAL FOR amWfWor
kItem_Ref INTO CounterValue;
ELSEIF CounterName = 'amCable_CableTag' THEN Values NEXTVAL FOR amCable
_CableTag INTO CounterValue;
ELSEIF CounterName = 'amCableLink_BarCode' THEN Values NEXTVAL FOR amCa
```

```
bleLink_BarCode INTO CounterValue;
ELSEIF CounterName = 'amColorCode_Code' THEN Values NEXTVAL FOR amColor
Code_Code INTO CounterValue;
ELSEIF CounterName = 'amTraceHistory_BarCode' THEN Values NEXTVAL FOR a
mTraceHistory_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 amContrac
t_Ref INTO CounterValue;
.....
```

2 Make the following modifications to the generated statement:
   1 Change the generated statement to properly handle IF Blocks.
   2 Remove the line with that deals with the empty counter:
   3 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 Co
unterIncrement 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 lockin
g, You must use a counter increment of 1, please add the importcount
ercache=1 in the amdb.ini file, in the parameters of your current da
tabase 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 am
WfActivAlarm_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfActivity_Ref' THEN Values NEXTVAL FOR amWf
Activity_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfEvent_Ref' THEN Values NEXTVAL FOR amWfEve
nt_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfInstance_Ref' THEN Values NEXTVAL FOR amWf
Instance_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfOrgRole_Ref' THEN Values NEXTVAL FOR amWfO
rgRole_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfScheme_Ref' THEN Values NEXTVAL FOR amWfSc
heme_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfTransition_Ref' THEN Values NEXTVAL FOR am
WfTransition_Ref INTO CounterValue;
ELSEIF CounterName = 'amWfWorkItem_Ref' THEN Values NEXTVAL FOR amWf
WorkItem_Ref INTO CounterValue;
ELSEIF CounterName = 'amCable_CableTag' THEN Values NEXTVAL FOR amCa
ble_CableTag INTO CounterValue;
ELSEIF CounterName = 'amCableLink_BarCode' THEN Values NEXTVAL FOR a
mCableLink_BarCode INTO CounterValue;
```

```
ELSEIF CounterName = 'amColorCode_Code' THEN Values NEXTVAL FOR amCo
lorCode_Code INTO CounterValue;
ELSEIF CounterName = 'amTraceHistory_BarCode' THEN Values NEXTVAL FO
R amTraceHistory_BarCode INTO CounterValue;
ELSEIF CounterName = 'amCatalog_Code' THEN Values NEXTVAL FOR amCata
log_Code INTO CounterValue;
ELSEIF CounterName = 'InternalRef' THEN Values NEXTVAL FOR InternalR
ef INTO CounterValue;
ELSEIF CounterName = 'amContract_Ref' THEN Values NEXTVAL FOR amCont
ract_Ref INTO CounterValue;
ELSE SELECT MAX(lCounterId) INTO CounterIdent FROM amCounter WHERE I
dentifier = CounterName;
IF CounterIdent IS NULL THEN SELECT MAX(lCounterId)+1 INTO CounterId
ent FROM amCounter;
INSERT INTO amCounter (lCounterId, Identifier, Description, lValue,
dtLastModif) VALUES (CounterIdent, CounterName, CounterName, Counter
Increment, CURRENT TIMESTAMP);
ELSE UPDATE amCounter SET lValue = lValue + CounterIncrement, dtLast
Modif = CURRENT TIMESTAMP WHERE lCounterId =CounterIdent;
END IF;
SELECT lValue INTO CounterValue FROM amCounter WHERE Identifier = Co
unterName;
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 8 - Edit the `amdb.ini` file

1 Open the **`amdb.ini`** file of AssetCenter

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

```
ImportCounterCache=1
```

3 Repeat this procedure on each client computer.

Important:

This step is absolutely mandatory.


Peregrine
SYSTEMS