Product Documentation

Peregrine

AssetCenter

Automatic software mechanisms

Part No. DAC-440-EN31

Peregrine SYSTEMS®

Build 26

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

List of Figures

PEREGRINE

List of Tables

PEREGRINE

Introduction

Who is this guide intended for?

This guide is intended for all enterprises using AssetCenter.

It is intended for engineers who require detailed information concerning the automatic data-processing mechanisms in AssetCenter:

- Database administrators.
- Those in charge of implementation or customization.

What does this guide offer?

This guide offers an overview of the different types of automatic mechanisms used in AssetCenter and gives an exhaustive listing of the different conditions governing these mechanisms. It also describes in detail the mechanisms associated with certain core tables in the database.

How to use this guide

Warning:

Broad and in-depth knowledge of AssetCenter is required to make proper use of this guide. In particular, mastery of the following areas is assumed: Database structure, Portfolio organization, Basic language and scripting.

Chapter Overview

This chapter explains the underlying principles related to automatic mechanisms in AssetCenter.

Read this chapter for an overview of automatic mechanisms.

Chapter Presentation of the automatic mechanisms

This chapter explains and categorizes all the automatic mechanisms operating in AssetCenter.

Chapter Automatic mechanisms in AssetCenter Server

This chapter presents the automatic mechanisms in AssetCenter Server.

Chapters Assets table (amAsset), Computers table (amComputer) and Portfolio Items table (amPortfolio)

These chapters detail every automatic mechanism in the tables concerned.

Chapter Glossary

This glossary contains the definitions of several key terms related to automatic mechanisms.

Appendix Extracting all the scripts from a database

This appendix explains how to extract all scripts from your database.

Appendix Determining the workflows used for a table

This appendix explains how to list all the workflows in a given table.

CHAPTER 1 Overview

AssetCenter uses a set of automatic mechanisms with three objectives in mind:

1. To maintain the structural and logical integrity of the data stored in the database. For example, integrity rules to maintain the relationship between the values of multiple fields.
2. To facilitate data entry. For example, scripted default values to populate certain fields automatically on creating a record.
3. To apply business rules globally or specifically. For example, workflows to trigger archival of past expense lines.

The use of the term **automatic mechanism** as used in this guide is large. It covers any sort of automatic modification to the database by a component of AssetCenter, triggered by an event (entering information in the user interface, updating a record, deletion of data by a workflow, etc.). All other external mechanisms outside of AssetCenter or its components, is not covered in this guide. This is the case, for example, of automatic mechanisms defined at the database level, such as triggers and stored procedures.

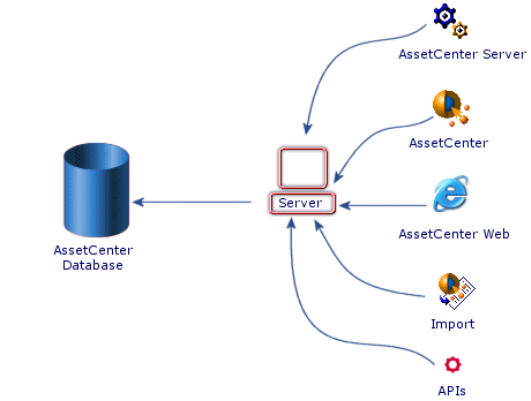
Concepts linked to automatic mechanisms

This section contains reminders of important general information concerning databases and specific information concerning the AssetCenter database.

Database access

The automatic mechanisms apply to all types of database access. The following diagram summarizes the different components that access that database:

Figure 1.1. Database access



Sequence of modification

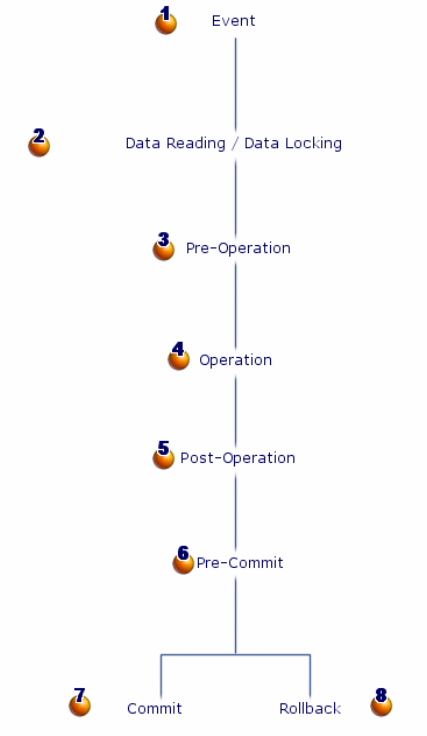
The modification of data in the database, whether it be an elementary operation (update, insert, delete) or a series of elementary operations, always follows the same sequence within a transaction.

Note:

A transaction may be made up of several SQL queries. A database manipulation involving read and write operations may be consistent once finished but pass through intermediate stages that are not.

The typical sequence of modification is as follows:

Figure 1.2. Sequence within a transaction



- 1: The event is not part of the transaction. It is at the origin of it. An event therefore means a manipulation that will potentially lead to a modification of the data in the database.
- 2: In order to maintain the consistency of the transaction, a data-locking mechanism is used. In practice, the first transaction to use a data item locks it. The other transactions in progress may therefore not use it until it is unlocked.
- 3, 4 and 5 constitute the sequential steps followed in all database operations be they INSERT, DELETE or UPDATE.

Note:

There may be several operations and therefore several Pre-Operation / Operation / Post-Operation cycles within the same transaction.

-  represents an interim step: The operations have been performed but the modifications have not yet been committed to the database.
-  : The modifications have been committed to the database.
-  : All the modifications have been cancelled. The database has not been modified by the transaction.

 Note:

Each database engine has its individual characteristics, in particular with regard to **Rollback** operations. Refer to the documentation provided with your DBMS for more information.

CHAPTER 2 Presentation of the automatic mechanisms

Several automatic mechanisms are used in AssetCenter:

- Scripts
- Integrity rules
- Agents
- Synchronous and asynchronous workflows
- Automatic mechanisms handled by AssetCenter Server

The objective of this chapter is to provide you with the most exhaustive list possible of these automatic mechanisms.

Categories of automatic mechanisms

As a convention, we have chosen to classify the different automatic mechanisms in AssetCenter using three major groups. The categories depend on the persistence of the automatic mechanisms:

1. Permanent automatic mechanisms, as their name suggests, are permanently activated for all database access methods (Windows client, APIs, etc.) as well as at the transaction level. Scripts and integrity rules enter into this category.

2 Synchronous automatic mechanisms, which are only triggered as the result of an event (modification of a record, specific step in a transaction, etc.). Agents and asynchronous workflows enter into this category.

3 Asynchronous automatic mechanisms, which are triggered in an uncorrelated manner with reference to events. This category includes automatic mechanisms managed by AssetCenter Server and asynchronous workflows, which are not dealt with in detail in this chapter. For a complete description of workflows, refer to the **Advanced use** guide, **Workflow** chapter.

Definition of automatic mechanisms

Basic scripts

In AssetCenter, Basic scripts are used to define and control automatic behavior. AssetCenter ships with a standard set of predefined scripts (and automatic mechanisms). The administrators and users may create their own scripts.

Scripts work:

- at the record level, or
- at the field and link level

The following table summarizes the different types of scripts.

Script name	Field of application	Definition
Validity	Record	This script applies to all records in a table and makes it possible to define conditions for validating new or modified records. For example, you can define an automatic mechanism to forbid the creation of numeric type features if the maximum value is less than the minimum value.
Historized	Field or Link	This script enables you to define conditions for historizing modifications made to a field or link.

Script name	Field of application	Definition
Read only	Field or Link	This script enables you to define the conditions under which a field or link can be modified.
Mandatory	Field or Link	This script enables you to define the conditions making a field or link mandatory.
Default	Field or Link	This script enables you to define the value that is automatically proposed for a field or a link when a new record is created.
Irrelevance	Field or Link	This script conditions whether a field or a link is displayed.

Integrity rules

AssetCenter permanently checks the consistency between certain field before authorizing insert or update operations in the database.

In practice, an integrity rule is made up of three elements:

1. The list of monitored objects (fields and links)
2. The rule concerning the objects monitored to be verified
3. The list of objects (fields and links) that can be modified in order to check the rule

Warning:

An integrity rule constantly checks the rule for which it is created. It sometimes has to perform arbitrations and modify values to maintain database integrity.

The integrity rules work recursively. For example, if an integrity rule, A, triggered by the modification of a field, C, modifies a field D, which in turn is monitored by a second integrity rule, B, then integrity rule B will execute when field D is modified without waiting for rule A to finish working.

Agents

An agent is an automatic mechanism that is triggered at the same time as a transaction. This can be before (**Pre**), during or after (**Post**) one of the following operations:

- Insert
- Update
- Delete

Note:

An agent can also be triggered before the database **Commit** operation.

An agent is made up of three elements:

1. The list of objects (fields and links) monitored by the agent with for each object the step of the transaction during which it is monitored.
2. The list of operations performed by the agent.
3. The list of objects (fields and links) updated by the process.

Agents work in cooperative mode. The are triggered once only and declare beforehand which objects are going to be modified by the process, thus allowing other agents to work.

Synchronous workflows

A synchronous workflow is a specific type of workflow used to implement behaviors that do not exist by default in AssetCenter. Unlike agents and integrity roles, workflows can be created and modified by the user. They are particularly suited to the needs of implementers who require company-specific or line-of-business-specific automatic mechanisms. In this type of workflow, events are processed immediately and the appropriate transitions are activated by AssetCenter Server.

For example, a synchronous workflow may be used to automatically propagate a changed cost center at the location level to the sub-locations.

There is no major functional difference between a synchronous workflow and an agent. Only their nature differs: An agent is hard coded in AssetCenter and cannot be modified, a synchronous workflow is part of the data in the database and may not be modified at will. Additionally, synchronous workflows are only executed after one of the operations previously mentioned (Insert, Update, Delete).

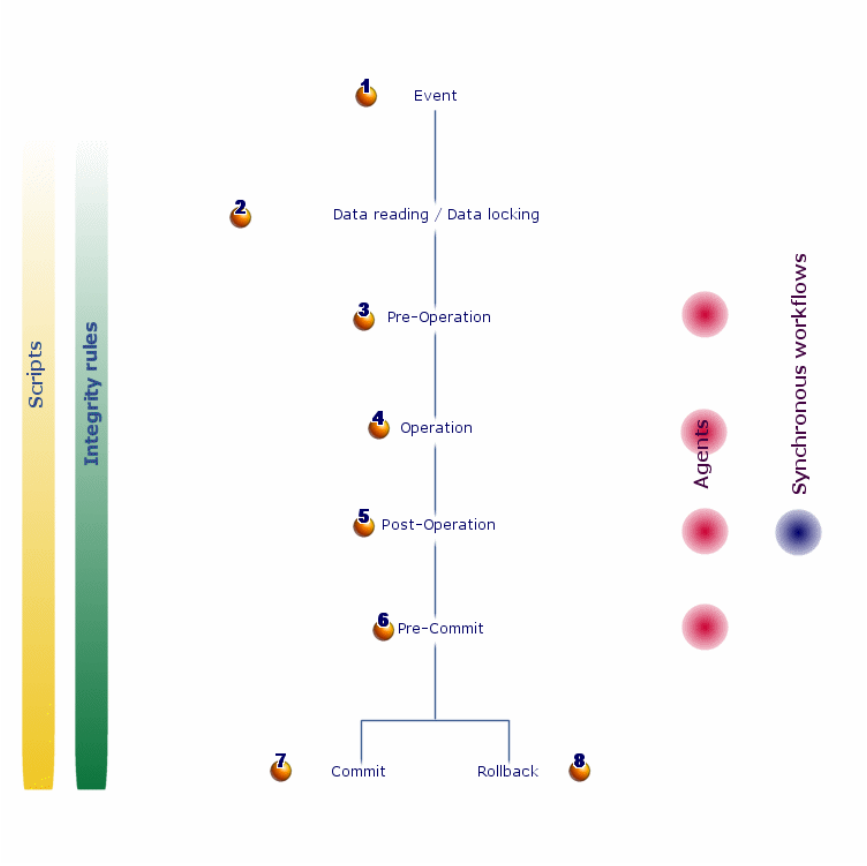
Note:

We invite you to read the documentation on workflows, the **Advanced use** guide, **Workflow** chapter.

Overview

The following diagram gives an overview of how the different mechanisms fit together to modify data.

Figure 2.1. Positioning of automatic mechanisms



CHAPTER 3 Automatic mechanisms in AssetCenter Server

This chapter includes reminders of the automatic mechanisms processed by AssetCenter Server.

Note:

For further information, refer to the **Administration** guide, AssetCenter Server chapter.

Overview of AssetCenter Server

AssetCenter includes a system to monitor deadlines and automatically trigger actions: This program, called AssetCenter Server, operates independently of AssetCenter and automatically monitors all designated database deadlines. In particular:

- Alarms (end of term dates of contracts for example).
- Purchase request approvals.
- Stock line reorder levels.
- Rent calculations at the asset and the contract level.
- Lease contract loss value calculations.

- Expense line split operations associated with cost centers.
- Verification of history lines.
- Workflow deadlines.
- Searches for new workflow execution groups.
- Execution of workflow rules.
- Verification of time zones.

If justified to do so by the deadlines, AssetCenter Server performs actions, such as issuing reminder messages in the AssetCenter database via the internal messaging system. If necessary, it calculates contract rent, lease contract loss-values, etc.

Each automatic mechanism carried out by AssetCenter Server is defined as a module.

AssetCenter Server modules

Add the computers listed in the NT domain to the database module (AddCpu)

AssetCenter Server enables you to program the recovery of those computers declared in the NT domain.

The domain to analyze is specified at the Connect-It **addcpu.scn** scenario.

Add NT users to the database module (AddUser)

AssetCenter Server enables you to program the recovery of the users declared on the NT domain.

This is essentially used to populate the **Departments and employees** table with the information useful for connecting to an AssetCenter database that uses integrated NT security.

The domain to analyze is specified at the Connect-It **adduser.scn** scenario.

Calculate rents module (Rent)

AssetCenter Server monitors periodic rent payments for contracts and assets. It calculates and/recalculates the amounts involved.

The **Calculate rents** module defines:

- Certain parameters concerning the generation of costs for contracts and asset-level rent payments.
- The frequency of updates.

Overview

AssetCenter Server verifies at regular intervals whether it needs to generate expense lines. If this is so, it generates them.

After checking and generating the expense lines relative to a periodic rent, AssetCenter Server stores the date of the last expense line (past or present) in the **Recalculation effective from** field (SQL name: dRecalcul).

- If the contract-level rent is distributed to the assets, AssetCenter Server modifies the **Recalculation effective from** field that is found in the rent sub-tabs of the **Acquis.** tab of the assets detail.
- If the contract-level rent is not distributed to asset level, AssetCenter Server modifies the **Recalculation effective from** field, which is found in the rent sub-tabs of the **Rents** tab of the contract detail.

AssetCenter Server does not recalculate every single expense line each time.

- Projected expense lines associated with a periodic rent are always recalculated.
- The **Recalculation effective from** field, proper to each rent, sets the date from which past and present expense lines associated with a periodic rent are recalculated.

  The lessee may directly modify the recalculation date of the non-projected expense lines by directly modifying the **Recalculation effective from** field. This flexibility enables you to recalculate erroneous expense lines in case of a change in tax rates, for example.

Parameters

The **User data item** field is used to set the rent generation parameters. The syntax of this field is as follows:

```
<Duration>j
```

This duration set the number of days for which the calculation is made. For example, if you want to calculate rent over a period of 90 days, enter the following value:

```
90d
```

Note:

The maximum number of rent calculations made per transaction is specified by the `UserData` entry in the **`Amsrv.cfg`** configuration file.

Location of this file: ▸ **AssetCenter - Installation** guide, chapter **.ini and .cfg files**.

Projected rent

The **User data item** field enables you to specify the number of days for which you calculate project rent payments.

AssetCenter Server generates the projected expense lines for the specified period. In order to not generate any, you just need to set this field to **0**.

Example

Let's consider the following configuration:

- The contract is effective from July 1, 2001 through July 1, 2004.
- The rent is payable monthly on the first day of the month.
- AssetCenter Server verifies rent payments every 2 months and generates projected rent payments for the next 12 months.

On July 1, 2002, AssetCenter Server is launched for the first time: it generates:

- Past rents from July 1, 2001 through July 1, 2002.
- The present rent on July 1, 2002.
- The projected rents from August 1, 2002 through July 1, 2003.

Following these calculations, the Recalculation effective from field indicates the date of the last non-projected expense line, i.e. July 1, 2002.

AssetCenter Server runs in the background: 2 months later on September 1, 2002, it generates:

- The projected rents from October 1, 2002 through September 1, 2003.
- Past or present rents for which the payment date is later than that contained in the Recalculation effective from field, i.e. the rents from August 1, 2002 through October 1, 2002.

Calculate stipulated loss values module (LostVal)

AssetCenter Server recalculates, at regular intervals, the loss values for lease contracts whose calculation method is set to **Calculate for all periods** (**Calculation** field (SQL name: seLossValCalcMode) in **Leasing** tab of the lease contract detail). In this way, loss values pertaining to any loss value rules that have been modified since the last time AssetCenter Server accessed the database are updated.

Create assets, consumables, etc. corresponding to items received module (Delivery)

Prerequisites

This module cannot be executed unless you have already done the following:

- Execute AssetCenter.
- Select the **Administration/ Database options** menu.
- Select the **Procurement/ Let AssetCenter Server create the items received in the portfolio** option.
- Set this option to **Yes**.

Task performed by the module

This module is used to process the records from the **Items received** table in order to create received items (assets, consumptions, etc.) in their respective tables.

Utility of this mode

Assigning this task to AssetCenter Server rather than the AssetCenter application can increase the performances of those users receiving orders.

Frequency of execution

We recommend that you execute this module several times a day if you want the users to be able to quickly access the items received in their respective tables.

Execute workflow rules for execution group modules

Once a workflow execution group (Example: **ADMIN**) is detected, AssetCenter Server executes the appropriate workflow rules.

Monitoring of workflow execution groups

AssetCenter Server monitors the deadlines specific to workflow instances associated with the execution group.

Deadlines to be monitored by AssetCenter Server as soon as the activity is triggered are defined in the **Alarms** tab of the detail of the workflow activity.

These deadlines are defined by the time limits defined for the set tasks to be carried out.

Note:

In the case of deadlines specific to workflow, business calendars specified in the **Time limit** tab in the activity detail are taken into account. When calculating deadlines, these time limits are converted to business hours.

Processing of Periodical type events

According to the frequency defined in the **Parameters** tab in the detail of a **Periodical** type event, AssetCenter Server triggers the event if the activation conditions are met.

Then, the role of AssetCenter Server depends on the event's processing mode as indicated in the **General** tab of the event detail:

- **Log event and process by server**: As soon as the event occurs, AssetCenter Server saves it to the table with SQL name "amWfOccurEvent".

  Then, AssetCenter Server activates the transition according to the frequency of verification as defined in the configuration screen of AssetCenter Server.
- **Log event and process immediately**: As soon as the event occurs, AssetCenter Server saves it to the table with SQL name "amWfOccurEvent", and activates the transition.
- **Process event immediately without logging**: As soon as the event occurs, AssetCenter Server activates the transition.

Activation of transitions

AssetCenter Server activates the transitions for events according to the frequency defined in the configuration screen. The following events are concerned:

- **System** events.
- **Database** and **Periodical** type events whose processing mode is set to **Log event and process by server**.

Execution of tasks

AssetCenter Server executes tasks resulting from **Automatic action** or **Test / script** type activities, except in the possible case of tasks resulting from activities whose **Execute actions immediately** (SQL name: bExecImmediately) box is selected.

The frequency with which AssetCenter Server verifies and performs the tasks it has to carry out is indicated in the configuration screen of AssetCenter Server.

In the case of a task originating from an **Automatic action** or **Test / script** type activity whose **Execute actions immediately** box (SQL name: bExecImmediately) is checked:

- This task is executed by AssetCenter Server if it is AssetCenter Server that activates the transition creating the task. In this case, AssetCenter Server performs the task as soon as the transition it creates is activated.
- Otherwise, the AssetCenter client machine executes the task.

Update the database with the results of the scanners module (IddAc)

AssetCenter Server enables you to program the recovery of the **`.fsf`** files produced by Desktop Inventory (these files store the results of the machine scan).

The folder containing the **`.fsf`** files is specified at the level of the Connect-It **`iddac.scn`** scenario.

 Note:

This module is based on the assumption that the machine scan has already been performed.

Update the database with the results of the scanners module (PdiAc)

This module enables you to program the recovery of **.xml.gz** files by Desktop Inventory. These files store the results of each computer scan.

The folder that contains the **.xml.gz** files is specified at the level of the Connect-It scenario **addcpu.scn**.

Update statistics for tables module (Stats)

This module updates the database statistics.

These statistics are used by all the DBMSs supported by AssetCenter to optimize SQL query plans.

If these statistics are not updated, the DBMS will not know which indexes are the most efficient.

We recommend that you execute this module once a week, or every night if your database is heavily modified.

Purge the input-events table module (PurgeEventInTable)

This module deletes the records from the **Input events** table according to the information in the:

- **Status** field (seStatus) of the **Input events** table (amInputEvent).
- **Deletion** field (seStatus) of the **Input events** table (amInputEvent).
- Expiration time defined by the **Events management/ Expiration time for input events (hours)**, accessible via the **Administration/ Database options** menu in the AssetCenter application.

Purge the outgoing-events table module (PurgeEventOutTable)

This module deletes the records from the **Input events** table according to the information in the:

- **Status** field (seStatus) of the **Output events** table (amOutputEvent).
- **Deletion** field (seStatus) of the **Output events** table (amOutputEvent).
- Expiration time defined by the **Events management/ Expiration time for output events (hours)**, accessible via the **Administration/ Database options** menu in the AssetCenter application.

Search for new workflow execution groups module (WorkflowFinder)

AssetCenter Server monitors the creation of new workflow execution groups.

As soon as AssetCenter Server detects a new workflow execution group **G**, it creates a new monitoring module **Execution of workflow rules for execution group G**.

This mechanism has the following advantages:

- It enables you to define verification timetables for each workflow execution group.
- Different workflow execution groups can be monitored by different instances of AssetCenter Server.

Signal presence of database server module (UpdateToken)

AssetCenter Server regularly sends a signal to the database server in order to indicate that it is functioning.

If the database server does not receive a signal from AssetCenter Server for over one hour, a message is displayed when a user connects to the database in AssetCenter.

This message indicates that AssetCenter Server has not been launched on this database for over one hour and that without this process, monitoring functions will be interrupted.

If the database server goes without receiving a signal from AssetCenter Server for over a week, it is no longer possible to connect to the database.

Split expense lines in cost centers module (CostCenter)

AssetCenter Server handles split operations for expense lines.

General overview

AssetCenter Server searches the expense lines to be split: These are the expense lines whose **Split operation status** field (SQL name: seSplitStatus) is set to **Not split**.

By default, all expense lines are to be split, regardless of their status (**Status** field (SQL name: seStatus) of an expense line).

AssetCenter Server splits the designated expense lines. When an expense line is split:

- A debit expense line, equivalent to the split expense line is created in the parent cost center.
- Expense lines are created in the target cost centers, according to the split percentage values. By default, these are **Not split**.

Specific example: Managing the removal of a cost center

When you decide to delete a cost center, and the cost center contains expense lines, AssetCenter will not allow you to perform the operation unless the **Authorize extended deletions** option in the **Edit** category of the **Edit/ Options** menu is checked.

In this case, AssetCenter gives you three possibilities:

- Delete all the linked records.
- Detach the linked records.
- Attach the linked records to another record.

What happens next depends on the option you choose:

Delete all linked records

When a cost center is deleted, AssetCenter deletes:

- The expense lines of the deleted cost center.
- The expense lines resulting from split operations on the deleted cost center.

An AssetCenter agent modifies the **Split operation status** field (seSplitStatus) so it displays "Not split" at the level of the expense lines highest up in the split operation. When these high-level expense lines were split, they generated the expense lines belonging to the deleted cost center (after any intermediate split operations).

When AssetCenter Server finds these expense lines, which are not split but have generated split expense lines, it deletes all the expense lines resulting from their split operations. In doing this, AssetCenter Server deletes the expense lines that, when split, generated the expense lines belonging to the deleted cost center.

Then AssetCenter Server performs the split operations on those expense lines, which have not yet been split. It thus recalculates, using new parameters, all the expense lines that, when split, generated the expense lines of the deleted cost center.

Detach all linked records

In this case:

- The expense lines of the deleted cost center are no longer associated with a cost center.
- The expense lines, which when split generated the expense lines for the deleted cost center, are split again.
- The expense lines, resulting from split operations on the deleted cost center, are not modified.

Attach linked records to another record

In this case, you select another cost center X, which takes the place of the deleted cost center:

- The expense lines of the deleted cost center are attached to cost center X.
- The expense lines, which when split generated the expense lines for the deleted cost center, are split again; cost center X is considered as the new target cost center.
- The expense lines resulting from split operations on the deleted cost center are deleted and the expense lines of cost center X are split.

Verify database server time zone module (TimeZone)

This module verifies the delay between the local time of the server and the client machines. This is useful if you specified a time zone for a client machine (menu **Administration/ Time zones**).

Verify alarms module (Alarms)

List of alarms monitored

At the asset level

Several key dates are monitored:

- The end-of-reservation date of an asset: This is shown in the **Reserv. end date** field (SQL name: dtEnd) in the **Portfolio/Reservations** tab of the asset detail.
- The warranty expiration date of an asset: Asset detail, **Maint.** tab, **Expiration** field (SQL name: dWarrEnd).
- End-of-term date for lease, rental, loan of an asset: This alarm can only be defined if the acquisition method of the asset (Asset detail, **Acquis.** tab, **Acq. method** field (SQL name: seAcquMethod)) is set to **Lease**, **Rental** or **Loan**. In this case the **Price and conditions** sub-tab of the **Acquis.** tab shows an **End date** field (SQL name: dEndAcqu).
- End-of-rent dates of an asset: Alarms can be attached to end of validity dates (**Acquis.** tab, rent descriptions sub-tabs, **Schedule** frame).

At the consumable level

AssetCenter Server monitors the end-of-reservation date for consumables: This is shown in the **Reserv. end date** field (SQL name: dReservEnd) in the reservation detail of a consumable. To access the reservation detail of a consumable:

1. Launch AssetCenter.
2. Select **Procurement/ Purchase requests**.
3. Select the purchase request reserving the consumable.
4. Display the composition of this purchase request.
5. Display the request line corresponding to the consumable.
6. Display the **Reservations** tab of the request line. This tab shows the list of reservations for consumables.
7. Display the detail of the reservation.

   The monitored field is **Date fin** (SQL name: dtEnd).

At the project level

AssetCenter Server monitors the end dates of project: Project detail, **General** tab, **End** field (SQL name: dEnd).

At the contract level

Several key dates are monitored:

- The end-of-term date: Contract detail, **General** tab, **End** field (SQL name: dEnd).
- If the contract **Type** (SQL name: seType) is **Lease schedule** or **Master lease**: Alarms can be attached to the notification dates for possible end of lease. These dates are shown to the right of the notification field in the sub-tabs describing the possible end of term options: **Renewal**, **Purchase**, **Return**.
- If the contract **Type** (SQL name: seType) is **Lease schedule**: Alarms can be attached to the end dates of validity for rent items as shown in the individual rent-description sub-tabs of the **Rent** tabs.

At the purchase request level

If the acquisition method of the purchase request (Purchase request detail, **Financing** tab, **Acq. method** field (SQL name: seAcquMethod)) is set to **Lease**, **Rental** or **Loan**, it is possible to define an alarm associated with the rental, lease or loan end dates (**Acq. method** field in **Financing** tab of purchase request detail).

The same is true for estimates and orders.

What happens in two-level alarms when the first level action has been triggered?

In the case of alarms with 2 levels, the triggering of the second level alarm depends on the action carried out at the first level.

- If the first-level alarm triggers an action other than the sending of a message via AssetCenter's internal messaging system (such as sending a message via a third-party messaging system), the second-level alarm will always be triggered at the defined moment.
- If the first level-alarm sends a message to a group of AssetCenter users via the internal messaging system, the action defined at the second level will not be triggered if one or more of the recipients has read the message.

Verify null-identifier records module (History)

This module verifies integrity of the records whose primary keys are null.

These records are automatically created in all the tables when the database is created.

They are used by AssetCenter to perform certain administrative tasks (which is transparent to you).

This module verifies that these records still exists, and will recreate them if necessary.

We recommend that you execute this module at least once every day to maintain the integrity of the database.

Verify history lines module (History)

Sometimes when a record is destroyed in the database, the corresponding history lines are not destroyed. AssetCenter Server verifies if there are any such history lines; if it finds any it destroys them.

Verify stocks module (Stock)

AssetCenter Server monitors stock reorder levels.

For each stock, AssetCenter Server refers to the stock rules defined in the **Manage** tab of the stock detail.

For each stock rule concerning a model:

- AssetCenter Server calculates the quantity of items actually available from the **Assignments** field in the detail of a portfolio item.
- When the quantity falls below the value specified in the **Reorder level** (SQL name: lReordLevel) field of the stock rule detail, AssetCenter Server automatically creates a purchase request.
  - The parameters of the purchase request can be found in the **Auto-request** and **Management** tabs of the detail of the stock.
  - The purchase request specifies the quantity to be reordered (**To order** field (SQL name: lQtyToOrder) in the detail of the stock rule).
- For as long as the request is not fully received, AssetCenter Server does not verify the stock rule that it has generated. Therefore, no new request is sent.
- As soon as delivery of the request is taken in full, AssetCenter Server:
  - Readjusts the stock levels.
  - Erases the contents of the **Request line** field (SQL name: ReqLine) in the stock rule detail.
  - Reactivates the stock rule.

CHAPTER 4 Assets table (amAsset)

This chapter provides an exhaustive list of all the mechanisms dealing with the Portfolio Items table. Each section deals with a different type of automatic mechanism.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 4.1. Validity scripts on the table

Script	Description

`If Not IsEmpty([dEndAcqu]) and Not IsEmpty([dStartAcqu]) and [dStartAcqu] > [dEndAcqu] Then`
`Err.Raise(-2009, "The end date (dEnd) must be greater than or equal to the start date (dStart).")`
`RetVal = FALSE`
`Else`
`RetVal = TRUE`
`End If`
	If the acquisition start date of the asset comes after the acquisition end date, the record is rejected.

Table 4.2. Default value scripts

Object concerned	Script	Description
AcctCode	`RetVal = [Model.AcctCode]`	By default, the accounting code of the asset is that of the model.
AssetTag	`RetVal = [Model.Prefix] + AmCounter("amAsset_AssetTag", 6)`	By default, the asset tag of the asset is the concatenation of the prefix of the model and the value of the **amAsset_AssetTag** counter on 6 figures.
dAcquisition	`RetVal = [dStartAcqu]`	By default,, the purchase date is set to the start of lease, loan or rental date.
dDeprRecalc	`RetVal = AmDate()`	By default, this field is set to the current system date.
DeprBasisCur	`RetVal = [PriceCur]`	By default, the currency in which the depreciation basis of an asset is expressed is identical to the one used to express its purchase value.
DeprValCur	`RetVal = AmDefaultCurrency()`	By default, this field is set to the value of the default currency.
dInstall	`RetVal = AmDate()`	By default, this field is set to the current system date.
dStartAcqu	`RetVal = AmDate()`	By default, this field is set to the current system date.

Object concerned	Script	Description
dtDeprBasisCv	`RetVal = AmDate()`	By default, this field is set to the current system date.
dtDeprValCv	`RetVal = AmDate()`	By default, this field is set to the current system date.
dtIntPayCv	`RetVal = AmDate()`	By default, this field is set to the current system date.
dtIntPayTaxCv	`RetVal = AmDate()`	By default, this field is set to the current system date.
dtListPriceCv	`RetVal = AmDate()`	By default, this field is set to the current system date.
dtMarketValCv	`RetVal = AmDate()`	By default, this field is set to the current system date.
dtNetValueCv	`RetVal = AmDate()`	By default, this field is set to the current system date.
dtPaymentsCv	`RetVal = AmDate()`	By default, this field is set to the current system date.
dtPriceCv	`RetVal = AmDate()`	By default, this field is set to the current system date.
dtPurchOptValCv	`RetVal = AmDate()`	By default, this field is set to the current system date.
dtResalePriceCv	`RetVal = AmDate()`	By default, this field is set to the current system date.
dtTaxCv	`RetVal = AmDate()`	By default, this field is set to the current system date.
fTotalQty	`RetVal = 1`	By default, the total quantity of the batch is 1.
IntPayCur	`RetVal = AmDefaultCurrency ()`	By default, this field is set to the value of the default currency.
IntPayTaxCur	`RetVal = AmDefaultCurrency ()`	By default, this field is set to the value of the default currency.
Label	`RetVal = [AssetTag]`	By default, the label of an asset is set to the asset tag. This is only relevant when the asset is a cable device.
lDeprSchId	`RetVal = [Model.lDeprSchId ]`	By default, the depreciation type of an asset is that of its model.

Object concerned	Script	Description
lIconId	`RetVal = [Model.lIconId]`	By default, this field, which contains the identifier of the icon used to represent the asset, inherits the same value as that of the model from which it is derived.
ListPriceCur	`RetVal = AmDefaultCurrency ()`	By default, this field is set to the value of the default currency.
lLabelRuleId	`RetVal = [Model.lLabelRule Id]`	By default, the label rule of an asset is set to the model. This is only relevant when the asset is a cable device.
lLessorId	`RetVal = [POrdLine.POrder. lSuppId]`	By default, the lessor of an asset is the supplier of the purchase order line at the origin of the creation of the asset.
lModelId	`RetVal=[PortfolioItem.lMod elId]`	By default, the model associated with the asset is that of the associated portfolio item.
lPhotoId	`RetVal = [Model.lPhotoId]`	By default, the photo of the asset is that of the model.
lSoftLicUseRights	`RetVal = [Model.lSoftLicUs eRights]`	By default, the installation and utilization rights are that of the model.
lSuppId	`RetVal = [POrdLine.POrder. lSuppId]`	By default, the supplier of an asset is the supplier of the purchase order line at the origin of the creation of the asset.
MarketValCur	`RetVal = [PriceCur]`	By default, the initial value of the asset is its purchase price.
mDeprBasis	`RetVal = [mPrice]`	By default, the depreciation basis of the asset is set to its purchase value.
mMarketVal	`RetVal = [mPrice]`	By default, the initial value of the asset is its purchase price.
NetValueCur	`RetVal = AmDefaultCurrency ()`	By default, this field is set to the value of the default currency.
PaymentsCur	`RetVal = AmDefaultCurrency ()`	By default, this field is set to the value of the default currency.

Object concerned	Script	Description
PriceCur	`RetVal = AmDefaultCurrency ()`	By default, this field is set to the value of the default currency.
PurchOptValCur	`RetVal = AmDefaultCurrency ()`	By default, this field is set to the value of the default currency.
ResalePriceCur	`RetVal = AmDefaultCurrency ()`	By default, this field is set to the value of the default currency.
sePeriodicity	`RetVal = 30`	By default, the frequency of associated rent payments is monthly (30 days).
seSoftLicMulti	`RetVal = [Model.seSoftLicMulti]`	By default, the software license type is that of the associated model.
seSoftLicType	`RetVal = [Model.seSoftLicType]`	By default, the software utilization license type is that of the associated model.
SoftMedia	`RetVal = [Model.SoftMedia]`	By default, the installation media is that of the associated model.
SoftOS	`RetVal = [Model.SoftOS]`	By default, the operating system is that of the associated model.
TaxCur	`RetVal = AmDefaultCurrency ()`	By default, this field is set to the value of the default currency.

Table 4.3. Read-Only scripts

Object concerned	Script	Description
fTotalQty	`RetVal = (2=[Model.Nature.seMgtConstraint] OR [lModellId]=0 OR [lAstId]<>0)`	

Table 4.4. Irrelevance scripts

Object concerned	Script	Description
dAcquisition	`RetVal = (0<>[seAcquMethod])`	This field, containing the acquisition date of the asset, is only relevant if the acquisition method of the asset is **Purchase**.
dDeprRecalc	`RetVal = (0<>[seAcquMethod])`	This field, containing the estimation date of depreciations and the residual value of the asset, is only relevant if the acquisition method of the asset is **Purchase**.
dEndAcqu	`RetVal = (1<>[seAcquMethod] AND 2<>[seAcquMethod] AND 3<>[seAcquMethod])`	This field, containing the end of acquisition date of the asset, is only relevant if the acquisition method of the asset is **Rental**, **Lease**, or **Loan**.
dIntPay	`RetVal = (1<>[seAcquMethod] AND 2<>[seAcquMethod])`	This field, containing the initial payment date of the asset, is only relevant if the acquisition method of the asset is **Rental**, **Lease**.
FixedAstNo	`RetVal = (0<>[seAcquMethod])`	This field, containing the fixed asset number of the asset, is only relevant if the acquisition method of the asset is **Purchase**.

Label	`RetVal = 1`
	`' Must be an asset and a device`
	`if [Model.Nature.seBasis] = 1 and [Model.Nature.bDevice] > 0 then`
	`RetVal = 0`
	`end if`
		This field, which contains the label of the asset, is only relevant if the asset is a cable device.

lAcquCntrId	`RetVal = (1<>[seAcquMethod] AND 2<>[seAcquMethod] AND 3<>[seAcquMethod])`	The link to a rental or leasing contract is only relevant if the acquisition method is **Rental**, **Lease** or **Loan**.
Language	`RetVal = (0=[Model.Nature.bSoftLicense])`	This field, containing the language version of the software, is only relevant if the asset is a software item.

Object concerned	Script	Description
lDeprSchId	`RetVal = (0<>[seAcquMethod])`	The link to a depreciation type is only relevant if the acquisition method of the asset is **Purchase**.
LessorCode	`RetVal = (1<>[seAcquMethod] AND 2<>[seAcquMethod])`	This field, containing the lessor code, is only relevant if the acquisition method of the asset is **Rental** or **Lease**.

lLabelRuleId	`RetVal = 1`
	`' Must be an asset and a device`
	`if [Model.Nature.seBasis] = 1 and [Model.Nature.bDevice] > 0 then`
	`RetVal = 0`
	`end if`
		This link to the label rule of the asset is only relevant if the asset is a cable device.

lLessorId	`RetVal = (1<>[seAcquMethod] AND 2<>[seAcquMethod] AND 3<>[seAcquMethod])`	The link to a lessor is only relevant if the acquisition method is **Rental**, **Lease** or **Loan**.
lLicCntrId	`RetVal = (0=[Model.Nature.bSoftLicense])`	The link to a license contract is only relevant if the asset is a software item.
lSoftLicUseRights	`RetVal = (0=[Model.Nature.bSoftLicense])`	This field, containing the number of utilization or installation rights, is only relevant if the asset is a software item.
mDeprBasis	`RetVal = (0<>[seAcquMethod])`	This field, containing the depreciation basis of the asset, is only relevant if the acquisition method of the asset is **Purchase**.
mDeprVal	`RetVal = (0<>[seAcquMethod])`	This field, containing the depreciation value of the asset, is only relevant if the acquisition method of the asset is **Purchase**.
mIntPay	`RetVal = (1<>[seAcquMethod] AND 2<>[seAcquMethod])`	This field, containing the initial payment for the asset, is only relevant if the acquisition method of the asset is **Rental**, **Lease**.

Object concerned	Script	Description
mIntPayTax	`RetVal = (1<>[seAcquMethod] AND 2<>[seAcquMethod])`	This field, containing the total amount of taxes on the initial payment made when acquiring the asset, is only relevant if the acquisition method of the asset is **Rental**, **Lease**.
mListPrice	`RetVal = (0<>[seAcquMethod] AND 1<>[seAcquMethod] AND 2<>[seAcquMethod])`	This field, containing list price of the asset, is only relevant if the acquisition method of the asset is **Purchase**, **Rental** or **Lease**.
mNetValue	`RetVal = (0<>[seAcquMethod])`	This field, containing the residual value of the asset, is only relevant if the acquisition method of the asset is **Purchase**.
mPrice	`RetVal = (0<>[seAcquMethod])`	This field, containing the purchase price of the asset, is only relevant if the acquisition method of the asset is **Purchase**.
mPurchOptVal	`RetVal = (2<>[seAcquMethod])`	This field, containing purchase option value of the asset, is only relevant if the acquisition method of the asset is **Lease**.
mTax	`RetVal = (0<>[seAcquMethod])`	This field, containing the purchase price of the asset, is only relevant if the acquisition method of the asset is **Purchase**.
pDiscount	`RetVal = (0<>[seAcquMethod] AND 1<>[seAcquMethod] AND 2<>[seAcquMethod])`	This field, containing the standard discount price of the asset, is only relevant if the acquisition method of the asset is **Purchase**, **Rental** or **Lease**.

sCnxCount	`RetVal = 1`
	`' Must be connectable`
	`if [Model.Nature.seBasis] = 1 and [Model.Nature.bIsCnxClient] > 0 then`
	`RetVal = 0`
	`end if`
		This field is only relevant for cable devices.

Object concerned	Script	Description

seCnxStatus	`RetVal = 1`
	`' Must be connectable`
	`if [Model.Nature.seBasis] = 1 and [Model.Nature.bIsCnxClient] > 0 then`
	`RetVal = 0`
	`end if`
		This field is only relevant for cable devices.

seSoftLicType	`RetVal = (0=[Model.Nature.bSoftLicense])`	This link, containing the utilization license type, is only relevant if the asset is a software item.

SharingName	`RetVal = 1`
	`' Must be connectable`
	`if [Model.Nature.seBasis] = 1 and [Model.Nature.bIsCnxClient] > 0 then`
	`RetVal = 0`
	`end if`
		This field is only relevant for cable devices.

sMaxCnxCount	`RetVal = 1`
	`' Must be connectable`
	`if [Model.Nature.seBasis] = 1 and [Model.Nature.bIsCnxClient] > 0 then`
	`RetVal = 0`
	`end if`
		This field is only relevant for cable devices.

SoftMedia	`RetVal = (0=[Model.Nature.bSoftLicense])`	This field, containing the installation media, is only relevant if the asset is a software item.
SoftOS	`RetVal = (0=[Model.Nature.bSoftLicense])`	This link, containing the operating system, is only relevant if the asset is a software item.
TerminOpt	`RetVal = (1<>[seAcquMethod] AND 2<>[seAcquMethod])`	This field, containing the termination option of the rental or leasing contract, is only relevant if the acquisition method is **Rental** or **Lease**.
VersionLevel	`RetVal = (0=[Model.Nature.bSoftLicense])`	This link is only relevant if the asset is a software item.
AcquContract	`RetVal = (1<>[seAcquMethod] AND 2<>[seAcquMethod] AND 3<>[seAcquMethod])`	The link to a rental contract is only relevant if the acquisition method is **Rental**, **Lease** or **Loan**.

Object concerned	Script	Description

AssetSlots	`RetVal = 1`
	`' Must be an asset and a device`
	`if [Model.Nature.seBasis] = 1 and [Model.Nature.bDevice] > 0 then`
	`RetVal = 0`
	`end if`
		This field is only relevant for cable devices.

DeprScheme	`RetVal = (0<>[seAcquMethod])`	The link to a depreciation type is only relevant if the acquisition method of the asset is **Purchase**.
FixedAssets	`RetVal = ((0<>[seAcquMethod]) And (3<>[seAcquMethod]))`	The link to the associated fixed assets is only relevant if the acquisition method of the asset is **Purchase** or **Loan**.

LabelRule	`RetVal = 1`
	`' Must be an asset and a device`
	`if [Model.Nature.seBasis] = 1 and [Model.Nature.bDevice] > 0 then`
	`RetVal = 0`
	`end if`
		This field is only relevant for cable devices.

Lessor	`RetVal = (1<>[seAcquMethod] AND 2<>[seAcquMethod] AND 3<>[seAcquMethod])`	The link to a lessor is only relevant if the acquisition method is **Rental**, **Lease** or **Loan**.
LicenseContract	`RetVal = (0=[Model.Nature.bSoftLicense])`	The link to a license contract is only relevant if the asset is a software item.

Link	`RetVal = 1`
	`' Must be an asset and a device`
	`if [Model.Nature.seBasis] = 1 and [Model.Nature.bDevice] > 0 then`
	`RetVal = 0`
	`end if`
		This field is only relevant for cable devices.

Pins	`RetVal = 1`
	`' Must be an asset and a device`
	`if [Model.Nature.seBasis] = 1 and [Model.Nature.bDevice] > 0 then`
	`RetVal = 0`
	`end if`
		This field is only relevant for cable devices.

Object concerned	Script	Description

Ports	`RetVal = 1`
	`' Must be connectable`
	`if [Model.Nature.seBasis] = 1 and [Model.Nature.bIsCnxClient] > 0 then`
	`RetVal = 0`
	`end if`
		This field is only relevant for cable devices.

Rents	`RetVal = (1<>[seAcquMethod] AND 2<>[seAcquMethod] AND 3<>[seAcquMethod])`	The link to the rent payments is only relevant if the acquisition method is **Rental**, **Lease** or **Loan**.

Integrity rules

Name of the rule	List of monitored objects	Rule(s) verified	List of any modified objects
CAssetDeprInteg	■ SyncRead on object: mDeprBasis ■ SyncRead on object: mDeprVal ■ SyncRead on object: mNetValue ■ SyncRead on object: DeprBasisCur ■ SyncRead on object: DeprValCur ■ SyncRead on object: NetValueCur	The rule forces the following relationship: `mNetValue = DeprBasis - mDeprVal` The residual value of an asset is always equal to the depreciation basis minus all amortizations.	■ DeprValCur ■ mDeprVal ■ mNetValue ■ NetValueCur

Name of the rule	List of monitored objects	Rule(s) verified	List of any modified objects
CBiSoftInteg	■ ASyncRead on object: Model.Nature.bSoftLicense ■ SyncRead on object: lSoftLicUseRights ■ SyncRead on object: seSoftLicType ■ SyncRead on object: seSoftLicMulti	The following rules are enforced for an asset for which the nature of the model is a software license: ■ If the license is not Multiple-user (**seSoftLicMulti**, the number of users for the license (**lSoftLicUserRights**) is forced to 1. The license type (**seSoftLicType**) is forced to **Per named workstation**. ■ If the number of users of the license is greater than 1, the license becomes Multiple-user.	■ lSoftLicUseRights ■ seSoftLicMulti ■ seSoftLicType

Name of the rule	List of monitored objects	Rule(s) verified	List of any modified objects
CDeprScriptInteg	■ SyncRead on object: mDeprBasis ■ SyncRead on object: dDeprRecalc ■ SyncRead on object: dStartAcqu ■ SyncRead on object: lDeprSchId ■ SyncRead on object: DeprBasisCur	If one of the monitored objects is updated, the rule runs the depreciation calculation script.	

Agents

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CAssetPinAgent	❖ Insert on object: amAsset	This agent creates the pins/terminals for the asset depending on the specified number in the model.	
CAssetPortAgent	■ Insert on object: amAsset ■ PreUpdate on object: lModelId	This agent maintains the integrity of any connections between an asset and another asset.	
CAssetSlotAgent	❖ Insert on object: amAsset	This agent takes the list of slots defined in the model and creates the corresponding slots for the asset.	

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CBatchQtyAgent	■ Insert on object: amPortfolio ■ PostUpdate on object: fTotalQty ■ PostUpdate on object: fQty ■ PostUpdate on object: lAstId ■ PreDelete on object: amPortfolio ■ PreDelete on object: amAsset	This agent maintains the consistency between the total quantity of a batch (**fTotalQty**) and the sum of the quantities of the batch items (**fQty**).	
CComputeNextRntStepAgent	❖ PostUpdate on object: dAccept	This agent adjusts the rent recalculation date according to the acceptance date of the asset.	**dRecalcul** in the **amAssetRent** table.
CDateAlarmAgent	❖ PostUpdate on object: dEndAcqu	This agent recalculates if necessary the alarms associated with the end of acquisition date of an asset.	None in the **amAsset** table.
CDateAlarmAgent	❖ PostUpdate on object: dWarrEnd	This agent recalculates if necessary the alarms associated with the end of warranty date of an asset.	None in the **amAsset** table.
CDateAlarmAgent	❖ PostUpdate on object: dEndCnx	This agent recalculates if necessary the alarms associated with the end of connection date of an asset.	None in the **amAsset** table.

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CGbAcquiDepAgent	■ Insert on object: amAsset ■ Insert on object: amPortfolio ■ PostUpdate on object: dAcquisition ■ PostUpdate on object: mPrice ■ PostUpdate on object: mTax ■ PostUpdate on object: dIntPay ■ PostUpdate on object: mIntPay ■ PostUpdate on object: mIntPayTax ■ PostUpdate on object: seAcquMethod	This agent updates the expense lines associated with the asset. It functions when an asset is created or the following data items are updated for an existing asset: ■ **seAcquMethod** ■ **dAcquisition** ■ **mPrice** ■ **mTax** ■ **mIntPay** ■ **mIntPayTax** ■ **dIntPay** Note: The agent takes into account the distribution (split-billing) of expenses to the cost centers and cost categories. It may therefore create multiple expense lines.	None in the **amAsset** table.

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CGbAssetAssignement	■ Insert on object: amCable ■ Insert on object: amContract ■ Insert on object: amComputer ■ Insert on object: amSoftInstall ■ Insert on object: amAsset ■ Insert on object: amPortfolio ■ Insert on object: amTraining ■ Insert on object: amWorkOrder ■ Insert on object: amPhone ■ PostDelete on object: amPortfolio ■ PreUpdate on object: lModelId ■ PreUpdate on object: lModelId ■ PreUpdate on object: lModelId ■ PreUpdate on object: lModelId ■ PreUpdate on object: lModelId	In case of creation of a portfolio item, if necessary this agent creates the corresponding record in the Assets table and the appropriate record in the overflow table matching the management constraint associated with the model of the portfolio item.	
CGbAssetPriceAgent	❖ PreUpdate on object: seAcquMethod	If the acquisition method of the asset is not **Purchase**, the agent empties the **Purchase date** and **Purchase price** fields of the asset.	■ **dAcquisition** ■ **mPrice**

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CGbBienContratAgent	■ Insert on object: amAsset ■ PostDelete on object: amAstCntrDesc ■ PostUpdate on object: lCntrId ■ PostUpdate on object: lAstId ■ PostUpdate on object: lMaintCntrId ■ PostUpdate on object: lAcquCntrId	This agent maintains the synchronization of the data between the **Contracts** tab of an asset and the **Schedule** and **Maint. contract** fields in the asset detail: ■ If one of these two fields is populated with a contract, then it is added to the list contracts in the **Contracts** tab. ■ If a rental or maintenance contract is removed from the list of contracts in the **Contracts** tab, the corresponding field is emptied.	■ **Schedule** ■ **Maint. contract**
CompteConnexions	■ Insert on object: amPort ■ PostDelete on object: amPort ■ PostUpdate on object: lPortId ■ PostUpdate on object: lAstId ■ PreUpdate on object: sCnxCount	This agent counts the number of ports linked to the asset (number of items listed in the **Ports** tab of the asset detail) and stores the information in the **sCnxCount** field.	◆ **sCnxCount**

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CRedundancyAgent	■ Insert on object: amAsset ■ Insert on object: amPortfolio ■ PostUpdate on object: AssetTag ■ PostUpdate on object: AssetTag ■ PostUpdate on object: lAstId	This agent makes sure that the **AssetTag** fields of an asset and its associated portfolio item are identical: ■ If the **AssetTag** field of a record in the **amAsset** table is modified, the agent propagates this change to the **AssetTag** field of the record in the corresponding **amPortfolio** table. ■ If the **AssetTag** field of a record in the **amPortfolio** table is modified, the agent propagates this change to the **AssetTag** field of the record in the corresponding **amAsset** table.	◆ **AssetTag**

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CRedundancyAgent	■ Insert on object: amAsset ■ Insert on object: amPortfolio ■ PostUpdate on object: lModelId ■ PostUpdate on object: lModelId ■ PostUpdate on object: lAstId	This agent makes sure that an asset and its associated portfolio item always point to the same model: ■ If the **lModelId** link of a record in the **amAsset** table is modified, the agent propagates this change to the **lModelId** field of the record in the corresponding **amPortfolio** table. ■ If the **lModelId** link of a record in the **amPortfolio** table is modified, the agent propagates this change to the **lModelId** link of the record in the corresponding **amAsset** table.	❖ **lModelId**

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
FinContAst	■ PreUpdate on object: lAcquCntrId ■ PreUpdate on object: lReturnEnvId ■ PreUpdate on object: dAccept		■ **dEndAcqu** ■ **dStartAcqu** ■ **lLessorId** ■ **seAcquMethod** ■ **seAcquStatus**

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
		The agent performs the following operations: ■ In case of modification of an asset's acquisition contract, it propagates the following information from the new contract to the asset: **dEndAcqu**, **dStartAcqu**, **lLessorId**, **seAcquMethod**. ■ If the acquisition status of the asset is **Not defined**, **On order** or **Received** and an acceptance date is set, then the acquisition status is automatically set to **Received**. ■ If the asset is assigned to a return envelope, the acquisition status of the asset is set to **To be returned**. ■ If the asset was **To be returned** and then removed from a return envelope, its acquisition status is set to **Accepted** if the acceptance date is populated. Otherwise, the acquisition status is set to	

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
		Not defined.	
LeaseSumAgent	■ Insert on object: amAstCntrDesc ■ Insert on object: amAsset ■ PostDelete on object: amAstCntrDesc ■ PostUpdate on object: lAstId ■ PostUpdate on object: lCntrId ■ PostUpdate on object: mMarketVal ■ PostUpdate on object: MarketValCur ■ PostUpdate on object: mIntPay ■ PostUpdate on object: IntPayCur	This agent: ■ Makes sure that a contract expressed in one currency cannot be linked to assets expressed in another. If this case arises, an error is returned. ■ Updates the following fields in the contract associated with the asset: **mMarketVal**, **mIntPay**, **mIntPayTax**, depending on the assets linked to the contract. ■ If a link between the asset and a contract is deleted, it recalculates the same information (**mMarketVal**, **mIntPay**, **mIntPayTax**) for the asset.	■ **mMarketVal** ■ **mIntPay** ■ **mIntPayTax**

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
RentAsset	■ Insert on object: amAssetRent ■ Insert on object: amAsset ■ PostUpdate on object: sePeriodicity ■ PostUpdate on object: bMainRent ■ PostUpdate on object: mPayments ■ PostUpdate on object: mPayments ■ PostUpdate on object: sePeriodicity	This agent updates the **Periodicity** and the asset rents associated with the asset using the same information stored at the asset level and vice versa. In addition, it creates an asset rent if this is not already the case.	None in the **amAsset** table.

CHAPTER 5 Assets Included in Projects table (amAstProjDesc)

This chapter provides an exhaustive list of all the mechanisms dealing with the Assets Included in Projects table. Each section deals with a different type of automatic mechanism.

Note:

There are no automatic mechanisms other than the default script values on this table.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 5.1. Default value scripts

Object concerned	Script	Description
dIncluded	`RetVal = AmDate()`	By default, the inclusion date of the asset in the project is the current date.
dRemoved	`RetVal = [Project.dEnd]`	By default, the removal date of the asset from the project is the project end date.

sSequenceNumber	`If [lAstProjDescId] = 0 Then`
	`RetVal = 1`
	`Else`
	`RetVal = AmDbGetLong("SELECT ISNULL(MAX(sSequenceNumber),0)+1 FROM amAstProjDesc where lAstProjDescId =" &[lAstProjDescId])`
	`End If`
		By default, the sequence number of the first asset added tp the project is set to 1. Otherwise, the sequence number of the asset is the last sequence number incremented by 1.

CHAPTER 6 Brands table (amBrand)

This chapter provides an exhaustive list of all the mechanisms dealing with the Brands table. Each section deals with a different type of automatic mechanism.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 6.1. Default value scripts

Object concerned	Script	Description
BarCode	`RetVal = "B" + AmCounter(" amBrand_BarCode", 6)`	By default, the barcode associated with a brand is the concatenation of the string "B" and the value of the **amBrand_BarCode** counter on 6 figures.

Table 6.2. Mandatory scripts

Object concerned	Script	Description
BarCode	`RetVal = (0<>[bInvent])`	If the brand is defined as to be inventoried at barcode inventories (**bInvent** field set to 1), the **Barcode** field becomes mandatory.

Integrity rules

There are no integrity rules on the Brands table (**amBrand**).

Agents

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
FullName agent	■ Insert in the **amBrand** table ■ Post-Update on the **Name** field ■ Post-Update on the **lParentId** link ■ Pre-Update on **Name** field ■ Pre-Update on **lParentId** link	This agent manages tree structures in hierarchic tables. In the Brands table, it maintains hierarchical integrity in the case of sub-brands. The full name of the brand and its hierarchical level are recalculated if: ■ A brand is created ■ The name of the brand is modified ■ The parent brand is modified	■ **FullName** ■ **sLvl**

CHAPTER 7 Catalogs table (amCatalog)

This chapter provides an exhaustive list of all the mechanisms dealing with the Catalogs table. Each section deals with a different type of automatic mechanism.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 7.1. Default value scripts

Object concerned	Script	Description
bExternal	`RetVal = 0`	By default, the catalog is not accessible externally.
Code	`RetVal = AmCounter("amCatalog_Code",6)`	By default, the internal catalog code takes the value of the **amCatalog_Code** counter on 6 figures
Description	`RetVal = [Name]`	By default, the description inherits the name of the catalog.
lDefCatSupplId	`RetVal = [Contract.lCpyId]`	The default supplier of a catalog is the company with which the associated contract was signed.
lLocalId	`RetVal = [DefSuppCat.lMainSite]`	The default location of the catalog is the main location of the default supplier.

Table 7.2. Irrelevance scripts

Object concerned	Script	Description

lDefCatSupplId	`RetVal = 1`
	`if amDbGetLong("SELECT COUNT(Distributors.lCpyId) FROM amCatalog where lCatalogId = " & [lCatalogId]) > 0 then`
	`Retval = 0`
	`end if`
		This link is only relevant if there are distributor companies for the catalog.

DefSuppCat	`RetVal = 1`
	`if amDbGetLong("SELECT COUNT(Distributors.lCpyId) FROM amCatalog where lCatalogId = " & [lCatalogId]) > 0 then`
	`Retval = 0`
	`end if`
		This field is only relevant in the case of a department or if the user has administration rights.

Integrity rules

There are no integrity rules on the Catalogs table (**amCatalog**).

Agents

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CCatalogDefSupplier	■ Post-Update on link **lDefCatSupplId** ■ Pre-Delete on table **amRelCatalogSuppliers**	Makes sure the default catalog supplier is in the list of catalog suppliers. If this is not the case, the supplier is added on the fly to this list.	◆ **amRelCatalogSuppliers**

CHAPTER 8 Products table (amCatProduct)

This chapter provides an exhaustive list of all the mechanisms dealing with the Products table. Each section deals with a different type of automatic mechanism.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 8.1. Validity scripts on the table

Script	Description

' Check that we have 1 default option per group
RetVal = TRUE
if [lProdOptId] <> 0 and [bDefaultOption] <> 0 Then
if amDbGetLong("SELECT COUNT(1) FROM amCatProduct WHERE lCatProductId <> " & [lCatProductId] & " AND lParentId = " & [lParentId] & " AND OptionGroup.lProdOptId = " & [lProdOptId] & " AND bDefaultOption <> 0 ") <> 0 then
Err.Raise(-2009, "There must be one and only one default option per option group for a product.")
RetVal = FALSE
end if
end if
	If there is more than one default option per option group for the product, the record is rejected.

Table 8.2. Default value scripts

Object concerned	Script	Description
bIsPackaged	RetVal = 0	By default, the product is not packaged. Orders for such products are not expressed in packaged units.

bPreinstalled	RetVal = 0
	if [lParentId]<>0 then
	RetVal = 1
	end if
		If the product has a parent then, by default, it is pre-installed on the product it is a component of.

Certification	RetVal = [Model.Certification]	By default, the certification associated with the product is inherited from the model of the product.
dCertification	RetVal = [Model.dCertification]	By default, the certification date associated with the product is inherited from the model of the product.

Object concerned	Script	Description
Description	`RetVal = [Model.Name]`	By default, the description of the takes the value of the name of the associated model.
dtPriceCv	`RetVal = AmDate()`	By default, the conversion date of the average price for the product corresponds to the date of creation of the record.
fPkgQty	`RetVal = 1`	By default, the quantity per item (expressed in the purchase unit) is set to 1.

fUnitConv	`If [lModelId] <> 0 And [PurchUnit.Dimension] <> [Model.UseUnit.Dimension] Then`
	
	`RetVal = 0`
	`ElseIf [lPurchUnitId] = 0 Then`
	`RetVal = 1`
	`Else`
	`RetVal = [PurchUnit.fConv]`
	
	`End If`
		◆ If the product has an associated model and the dimension (mass, temperature, etc.) in which its unit is expressed is different from that expressed at the model level, then the conversion coefficient for the purchase unit to the unit in which the model is used is zero.
		■ If no unit of measurement or packaging is defined for the product, then the coefficient is set to 1.
		■ In the other cases, this coefficient is identical to the conversion coefficient defined for the unit of measurement or packaging.

InternalRef	`RetVal = AmCounter("InternalRef", 6)`	By default, the internal reference of the product takes the value of the **InternalRef** counter, truncated to 6 figures.
lBrandId	`RetVal = [Model.lBrandId]`	By default, the brand of the product is inherited from the associated model.
lIconId	`RetVal = [Model.lIconId]`	By defaut, the icon associated with the product is that of its associated model.

Object concerned	Script	Description
lPurchUnitId	`RetVal = [Model.lUseUnitId]`	By default, the unit of measurement or packaging of the product is inherited from the associated model.
lSetQty	`RetVal = 1`	By default, the number of items in the product packaging is 1.
PriceCur	`RetVal = AmDefaultCurrency()`	By default, the currency in which the average price of the product is expressed, is the default currency.

Table 8.3. Irrelevance scripts

Object concerned	Script	Description
bDefaultOption	`RetVal = (0=[lParentId] or 0=[bOption])`	This field, which designates the default option, is only relevant if the product has a parent and the product is an option of its parent product.
bOption	`RetVal = (0=[lParentId])`	This field is irrelevant if the product does not have a parent.
bPreinstalled	`RetVal = (0=[lParentId])`	This field is irrelevant if the product does not have a parent.
fPkgQty	`'Package Qty is not relevant if lSetQty is irrelevant or fUnitConv is irrelevant` `RetVal = (0=[lPurchUnitId]) OR (0=[bIsPackaged]) OR (0=[lSetQty])`	This field is only relevant in the following cases: ■ A unit of measurement or packaging is defined for the product ■ The product is packaged ■ The number of items in the packaged product is not zero
fUnitConv	`RetVal = (0=[lPurchUnitId] OR amEvalScript("Irrelevant", "PurchUnit", "")=TRUE)`	This field is only relevant if a unit of measurement or packaging is defined for the product

Object concerned	Script	Description
lProdOptId	`RetVal = (0=[lParentId] or 0=[bOption])`	This field, which designates the default option, is only relevant if the product has a parent and the product is an option of its parent product.
lSetQty	`RetVal = (0=[bIsPackaged])`	This field is only relevant if the product is packaged.
OptionGroup	`RetVal = (0=[lParentId] or 0=[bOption])`	This field, which designates the default option, is only relevant if the product has a parent and the product is an option of its parent product.

Agents

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
FullName agent	■ Insert on object: amCatProduct ■ PostUpdate on object: InternalRef ■ PostUpdate on object: lParentId ■ PreUpdate on object: InternalRef ■ PreUpdate on object: lParentId	This agent manages tree structures in hierarchic tables. In the Products table, it maintains the integrity of the hierarchical structure. The full name of the product and its hierarchical level are recalculated if: ■ the internal reference of the product is modified ■ its parent is modified	■ FullName ■ sLvl

CHAPTER 9 Catalog References table (amCatRef)

This chapter provides an exhaustive list of all the mechanisms dealing with the Catalog References table. Each section deals with a different type of automatic mechanism.

Note:

There are no automatic mechanisms other than the default script values on this table.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 9.1. Default value scripts

Object concerned	Script	Description
Certification	`RetVal = [CatProduct.Certification]`	By default, the certification of a catalog reference is inherited from the product.
Description	`RetVal = [CatProduct.Description]`	By default, the description of a catalog reference is inherited from the product.
dPriceUpdate	`RetVal = AmDate()`	By default, the price update date of the catalog reference is the date of creation of the catalog reference.
dtEndValidity	`RetVal = [Catalog.dtEndValidity]`	By default, the end date of validity of the reference is that of the catalog containing the reference.
dtStartValidity	`RetVal = [Catalog.dtStartValidity]`	By default, the validity start date of the reference is that of the catalog containing the reference.
fMinQty	`RetVal = 1`	By default, the minimum orderable quantity for the reference is set to 1.

fPrice	`If [CatProduct.PriceCur] = [Catalog.Currency.Name] Then`
	`RetVal = [CatProduct.mPrice]`
	`Else`
	`RetVal = 0`
	`End If`
		■ If the currency used for the product is identical to that used for the catalog then the purchase price in the catalog reference is inherited from the product.
		■ Otherwise, the purchase price is set to 0.

Object concerned	Script	Description
Ref	`RetVal = [CatProduct.Description] + " (" + [Catalog.Name] + ")"`	By default, the catalog reference number corresponds to the product description.

Table 9.2. Mandatory scripts

Object concerned	Script	Description
IClassCodeId	`RetVal = (""<>[Catalog.ProdClass])`	This field, which designates the classification codes, is mandatory if the product has a classification used as a reference.

Integrity rules

There are no integrity rules on the Catalog References table (**amCatRef**).

Agents

The following table lists the active agents on the Catalog References table (**amCatRef**).

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CDateAlarmAgent	◆ Post-Update on the **dtEndValidity** field	This agent recalculates alarms associated with the End of Validity date of the catalog reference.	None in the **amCatRef** table.

10 CHAPTER Companies table (amCompany)

This chapter provides an exhaustive list of all the mechanisms dealing with the Companies table. Each section deals with a different type of automatic mechanism.

Note:

There are no automatic mechnisms other than the scripts on this table.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 10.1. Default value scripts

Object concerned	Script	Description
Code	`RetVal = "S" + AmCounter("amCompany_Code", 6)`	By default, the unique code associated with the company is the concatentation of the letter **C** and the value of the **amCompany_Code** counter on 6 figures.

Table 10.2. Irrelevance scripts

Object concerned	Script	Description
CardTypesAccepted	`RetVal = (1=[sePayment])`	This link, which points to the card types accepted by the company, is irrelevant if the company does not accept payment cards.
Contacts	`RetVal = (0=[lCpyId])`	This link, which points to the contracts defined for the company, is irrelevant if the identifier of the company is zero.

11 CHAPTER Computers table (amComputer)

This chapter provides an exhaustive list of all the mechanisms dealing with the Computers table. Each section deals with a different type of automatic mechanism.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 11.1. Default value scripts

SQL name of the object concerned	Script	Description
CPUType	`RetVal = [Portfolio.Model.CPUType]`	By default, this field, which contains the processor type of the computer, inherits the same value as that of the portfolio item model from which it is derived.
lCPUSpeedMHz	`RetVal = [Portfolio.Model.lCPUSpeedMHz]`	By default, this field, which contains the processor speed of the computer, inherits the same value as that of the portfolio item model from which it is derived.
lDiskSizeMb	`RetVal = [Portfolio.Model.lDiskSizeMb]`	By default, this field, which contains the hard disk size of the computer, inherits the same value as that of the portfolio item model from which it is derived.
lIconId	`RetVal = [Portfolio.lIconId]`	By default, this field, which contains the identifier of the icon used to represent the computer, inherits the same value as that of the model from which it is derived.
lMemorySizeMb	`RetVal = [Portfolio.Model.lMemorySizeMb]`	By default, this field, which contains the RAM size of the computer, inherits the same value as that of the portfolio item model from which it is derived.

SQL name of the object concerned	Script	Description

Name	`if [bGroup]=0 then`
	`RetVal = "CPU" + AmCounter ("amComputer_Name", 6)`
	`else`
	`RetVal = "GRP" + AmCounter ("amComputer_Group", 6)`
	`end if`
		This script enables you to automatically name a computer or computer group:
		■ If it is not a computer group (`[bGroup] = 0`), the name assigned by default is the result of concatenating the "CPU" string and the value of the **amComputer_Name** counter on 6 figures.
		■ If it is a computer group, the name assigned by default is the result of concatenating the string "GRP" and the value of the **amComputer_Group** counter on 6 figures.
		Note:
		For further information on the **AmCounter()**, refer to the AssetCenter Programmer's Reference.
		For further information on counter, refer to the **Administration** guide, chapter **Standard database description files**, section **Customizing the database/ Counters in field default values**.

TcpIpAddress	`if [bGroup] <> 0 then`
	`RetVal = ""`
	`else`
	`RetVal = [Name]`
	`end if`
		This script is useful for computer groups. In this case, it populates the field that usually contains the IP address of the computer with the name of the computer group. If it is not a group (`[bGroup] <> 0`), the field is left empty.

SQL name of the object concerned	Script	Description

TcpIpHostName	`if [bGroup] <> 0 then`
	`RetVal = ""`
	`else`
	`RetVal = [Name]`
	`end if`
		This script is useful for computer groups. In this case, it populates the field that usually contains the IP name of the computer with the name of the computer group. If it is not a group (`[bGroup] <> 0`), the field is left empty.

Table 11.2. Irrelevance scripts

Object concerned	Script
BIOSAssetTag	`RetVal = (0<>[bGroup])`
BIOSSource	`RetVal = (0<>[bGroup])`
bTcpIpRouting	`RetVal = (0<>[bGroup])`
ComputerDesc	`RetVal = (0<>[bGroup])`
ComputerType	`RetVal = (0<>[bGroup])`
CPUInternal	`RetVal = (0<>[bGroup])`
CPUType	`RetVal = (0<>[bGroup])`
dtBIOS	`RetVal = (0<>[bGroup])`
dtHardScan	`RetVal = (0<>[bGroup])`
dtLastScan	`RetVal = (0<>[bGroup])`
dtNetworkScan	`RetVal = (0<>[bGroup])`
dtNextScan	`RetVal = (0<>[bGroup])`
dtSoftScan	`RetVal = (0<>[bGroup])`
IpxSpxAddress	`RetVal = (0<>[bGroup])`
IpxSpxServer	`RetVal = (0<>[bGroup])`
lColorDepth	`RetVal = (0<>[bGroup])`
lCPUNumber	`RetVal = (0<>[bGroup])`
lCPUSpeedMHz	`RetVal = (0<>[bGroup])`
lDiskSizeMb	`RetVal = (0<>[bGroup])`
lHorizontalRes	`RetVal = (0<>[bGroup])`
lItemId	`RetVal = (0<>[bGroup])`
lMemorySizeMb	`RetVal = (0<>[bGroup])`
lScanHistId	`RetVal = (0<>[bGroup])`
lSwapSizeMb	`RetVal = (0<>[bGroup])`
lVerticalRes	`RetVal = (0<>[bGroup])`
OperatingSystem	`RetVal = (0<>[bGroup])`
OSBuildNumber	`RetVal = (0<>[bGroup])`
OSDirectory	`RetVal = (0<>[bGroup])`

Object concerned	Script
OSLocale	`RetVal = (0<>[bGroup])`
OSServiceLevel	`RetVal = (0<>[bGroup])`
PhysicalAddress	`RetVal = (0<>[bGroup])`
ScannerDesc	`RetVal = (0<>[bGroup])`
ScannerVersion	`RetVal = (0<>[bGroup])`
SoundCard	`RetVal = (0<>[bGroup])`
TcpIpAddress	`RetVal = (0<>[bGroup])`
TcpIpDomain	`RetVal = (0<>[bGroup])`
TcpIpHostName	`RetVal = (0<>[bGroup])`
VideoCard	`RetVal = (0<>[bGroup])`
Workgroup	`RetVal = (0<>[bGroup])`
Agents	`RetVal = (0<>[bGroup])`
DaTracking	`RetVal=(0<>[bGroup])`
ExtensionCards	`RetVal = (0<>[bGroup])`
LogicalDrives	`RetVal = (0<>[bGroup])`
NetworkCards	`RetVal = (0<>[bGroup])`
PhysicalDrives	`RetVal = (0<>[bGroup])`
Portfolio	`RetVal = (0<>[bGroup])`
ScanHistory	`RetVal = (0<>[bGroup])`
SubGroups	`RetVal = (0=[bGroup])`

The following objects share the same irrelevance script:

```
RetVal = (0=[bGroup])
```

In the case of a computer group, they are not relevant. There are therefore not displayed.

Integrity rules

There are no integrity rules on the Computers table (**amComputer**).

Agents

The following table lists the agents working on the Computers table (**amComputer**).

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CGbAssetAssignement	■ Insert in **amComputer** table. ■ Pre-Commit of a transaction when it impacts the **amComputer** table.	The Computers table is an overflow table of the Portfolio Items table. When a record is created in the **amComputer** table, a record is created in the reference table - in this case the Portfolio Items table (**amPortfolio**) - except if the overflow link is irrelevant, which is the case for computer groups. A record is also created in the Assets table (**amAsset**). Note: For further information on overflow tables, refer to the **Portfolio** guide, chapter **Overview (Portfolio)**, section **Overflow tables**.	None in the **amComputer** table.

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CRedundancyAgent	■ Insert in the **amComputer** table ■ Insert in the **amPortfolio** table ■ Post-Update on the **lItemId** link in the **amComputer** table. ■ Post-Update on the **AssetTag** field in the **amComputer** table. ■ Post-Update on the **AssetTag** field in the **amPortfolio** table.	This agent makes sure that the **AssetTag** fields of a computer and its associated portfolio item are identical: ■ If the **AssetTag** field of a record in the **amComputer** table is modified, the agent propagates this change to the **AssetTag** field of the record in the corresponding **amPortfolio** table. ■ If the **AssetTag** field of a record in the **amPortfolio** table is modified, the agent propagates this change to the **AssetTag** field of the record in the corresponding **amComputer** table.	■ **AssetTag** field in the **amComputer** table. ■ **AssetTag** field in the **amPortfolio** table.

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
FullName	■ Insert in the **amComputer** table ■ Post-Update on the **Name** field ■ Post-Update on the **bGroup** field ■ Post-Update on the **lParentId** link ■ Pre-Update on the **Name** field ■ Pre-Update on the **bGroup** field ■ Pre-Update on the **lParentId** link	This agent manages tree structures in hierarchic tables. In the Computers table, it maintains hierarchical integrity in the case of computer groups. The full name of the computer and its hierarchical level are recalculated if: ■ the name of the computer or its parent group is modified ■ the group of the computer (of its parent) is modified ■ the computer is changed to a computer group or vice-versa	■ **FullName** field ■ **sLvl** field

Workflows

The following tables summarize the workflows dealing with the Computers table (**amComputer**).

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Determining the workflows used for a table [page 183] at the end of this document.

Workflow reference	Workflow type	Description
BST_SAM20	Synchronous	This workflow updates the installed software not detected at the last scan by setting their **Assignment (seAssignment)** field to **Missing**. It is automatically triggered when the **Last software inventory (dtSoftScan)** field is updated.
STD_PROCUR_POPULATED	Synchronous	This workflow updates the information on a computer. It is triggered automatically when a record is created in the **amComputer** table.
DA_SCAN	Asynchronous	This workflow, triggered at regular intervals, starts the computer scan according to the options specified in the database.

12 CHAPTER Contacts table (amContact)

This chapter provides an exhaustive list of all the mechanisms dealing with the Contracts table. Each section deals with a different type of automatic mechanism.

Note:

There are no automatic mechanisms other than the default script values on this table.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 12.1. Default value scripts

Object concerned	Script	Description
Fax	`RetVal = [Company.Fax]`	By default, the fax number of a contact is that of their company.
Phone	`RetVal = [Company.Phone]`	By default, the telephone number of a contact is that of their company.

13 CHAPTER Contracts table (amContract)

This chapter provides an exhaustive list of all the mechanisms dealing with the Contracts table. Each section deals with a different type of automatic mechanism.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 13.1. Validity scripts on the table

Script	Description
`If Not IsEmpty([dEnd]) and Not IsEmpty([dStart]) and [dStart] > [dEnd] Then Err.Raise(-2009, "The end date (dEnd) must be greate r than or equal to the sta rt date (dStart).") RetVal = FALSE Else RetVal = TRUE End If`	If the start and end dates of the contract are not empty and the end date comes before the start date, the record is rejected.

Table 13.2. Default value scripts

Object concerned	Script	Description
AmountCur	`RetVal = AmDefaultCurrency ()`	By default, the amount of the contract is expressed in the default currency.
AssignCond	`RetVal = [Parent.AssignCon d]`	By default, the assignment conditions of a contract are inherited from its parent contract.
AstIntPayTaxCur	`RetVal = AmDefaultCurrency ()`	By default, the tax on the interim rents of the assets on the contract are expressed in the default currency.
bAssignable	`RetVal = [Parent.bAssignab le]`	By default, the possibility of assigning a contract is inherited from its parent.
bPurchOpt	`RetVal = [Parent.bPurchOpt ]`	By default, the possibility of purchasing the assets on a contract is inherited from its parent.
bRenOpt	`RetVal = [Parent.bRenOpt]`	By default, the possibility of renewing the assets on a contract is inherited from its parent.
bRetOpt	`RetVal = [Parent.bRetOpt]`	By default, the possibility of returning the assets on a contract is inherited from its parent.

Object concerned	Script	Description
bUpgOpt	`RetVal = [Parent.bUpgOpt]`	By default, the possibility of upgrading the contract is inherited from its parent.

dEnd	`If [lParentId]<>0 OR [Model.tsCntrDuration]=0 Then`
	`RetVal = [Parent.dEnd]`
	`Else`
	`RetVal = AmDateAddLogical( [dStart], [Model.tsCntrDuration])`
	`End If`
		If the contract has a parent contract, or the planned duration of the contracts at the model level is zero, then the default end date of the contract is that of the parent contract. Otherwise, the contract end date is calculated by adding the length specifiied at the model level to the contract start date.

dPurchNotice	`RetVal = [Parent.dPurchNotice]`	By default, the purchase notice date for the contract is inherited from its parent.
dRenNotice	`RetVal = [Parent.dRenNotice]`	By default, the renewal notice date for the contract is inherited from its parent.
dRetNotice	`RetVal = [Parent.dRetNotice]`	By default, the return notice date for the contract is inherited from its parent.

dStart	`If [lParentId]<>0 Then`
	`RetVal = [Parent.dStart]`
	`Else`
	`RetVal = Date()`
	`End If`
		If the contract has a parent contract, the contract start date is inherited from the parent. Otherwise, it is contract creation date.

dtAmountCv	`RetVal = AmDate()`	By default, the conversion date for the amount of the contract is the creation date of the record.
dtAstIntPayTaxCv	`RetVal = AmDate()`	By default, the conversion date for the tax on the interim rent for the assets on the contract is the record creation date.
dtIntPayAstCv	`RetVal = AmDate()`	By default, the conversion date for the interim rent for the assets on the contract is the record creation date.

Object concerned	Script	Description
dtIntPayCv	`RetVal = AmDate()`	By default, the conversion date for the interim rent of the contract is the record creation date.
dtIntPayTaxCv	`RetVal = AmDate()`	By default, the conversion date for the tax on the interim rent of the contract is the record creation date.
dtMarketValCv	`RetVal = AmDate()`	By default, the conversion date for the total value of the assets on the contract is the record creation date.
dtPOCommitmentCv	`RetVal = AmDate()`	By default, the conversion date for the contract commitment is the creation date of the record.
IntPayAstCur	`RetVal = AmDefaultCurrency()`	By default, the interim rents of the assets on the contract are expressed in the default currency.
IntPayCur	`RetVal = AmDefaultCurrency()`	By default, the interim rent for the contract is expressed in the default currency.
IntPayTaxCur	`RetVal = AmDefaultCurrency()`	By default, the tax on the interim rent for the contract is expressed in the default currency.
lAssigneeId	`RetVal = [Parent.lAssigneeId]`	By default, the contact assignee is inherited from its parent.
lBillAddrId	`RetVal = [Parent.lBillAddrId]`	By default, the billing address of a contract is inherited from its parent.
lBillCnctId	`RetVal = [Parent.lBillCnctId]`	By default, the billing contact of a contract is inherited from its parent.
lCntrCnctId	`RetVal = [Parent.lCntrCnctId]`	By default, the contact of a contract is inherited from its parent.

lCostCatId	`If [lParentId]<>0 Then`
	`RetVal = [Parent.lCostCatId]`
	`Else`
	`RetVal = [Model.lCostCatId]`
	`End If`
		If the contract has a parent contract, the cost category of the contract is inherited from its parent. Otherwise, it is inherited from its model.

Object concerned	Script	Description
lCostId	`RetVal = [Parent.lCostId]`	By default, the cost center of a contract is inherited from its parent.

lCpyId	`RetVal = [POrdLine.POrder.lSuppId]`
	`If RetVal=0 Then`
	`RetVal = [Parent.lCpyId]`
	`End If`
		If the contract has a parent contract, the company associated with the contract is inherited from the parent. Otherwise, the company is taken from the supplier specified in the order line giving rise to the contract.

lIconId	`RetVal = [Model.lIconId]`	By default, the icon used to represent the contract is identical to that used for the model.
lInsurCnctId	`RetVal = [Parent.lInsurCnctId]`	By default, the insurance contract of a contract is inherited from its parent.
lLessorId	`RetVal = [Parent.lLessorId]`	By default, the lessor associated with a contract is inherited from its parent.
lLossValRuleId	`RetVal = [Parent.lLossValRuleId]`	By default, the loss value rule associated with a contract is inherited from its parent.
lNotifAddrId	`RetVal = [Parent.lNotifAddrId]`	By default, the notification address of a contract is inherited from its parent.
LossCond	`RetVal = [Parent.LossCond]`	By default, the lessor indemnification conditions in case of loss or destruction of assets are inherited from the parent.
lSupervId	`RetVal = [Parent.lSupervId]`	By default, the contact supervisor is inherited from its parent.
lTechCnctId	`RetVal = [Parent.lTechCnctId]`	By default, the technical contact of a contract is inherited from its parent.
MarketValCur	`RetVal = AmDefaultCurrency()`	By default, the total value of the assets on the contract is expressed in the default currency.

Object concerned	Script	Description
Nature	`If [Parent]<>0 Then RetVal = [Parent.Nature] End If`	By default, the nature of a contract is inherited from its parent.
pDefLRF	`RetVal = [Parent.pDefLRF]`	By default, the lease rate factor for a contract is inherited from its parent.
pDefRenPercent	`RetVal = [Parent.pDefRenPercent]`	By default, the percentage to apply to the previous rent to determine the renewed rent payments, is inherited from the parent contract.
pIntRentPercent	`RetVal = [Parent.pIntRentPercent]`	By default, the percentage associated with a contract is inherited from its parent.
POCommitmentCur	`RetVal = AmDefaultCurrency()`	By default, the commitment amount associated with the contract is expressed in the default currency.
PurchOptType	`RetVal = [Parent.PurchOptType]`	By default, the purchase option type of a contract is inherited from its parent.
Purpose	`RetVal = [Model.Name]`	By default, the contract purpose takes the value of the name of the model associated with the contract.
Ref	`RetVal = "C" + AmCounter("amContract_Ref", 6)`	By default, the contract reference is the concatentation of the letter **C** and the value of the **amContract_Ref** counter on 6 figures.
RenOptType	`RetVal = [Parent.RenOptType]`	By default, the renewal option type of a contract is inherited from its parent.
RetOptType	`RetVal = [Parent.RetOptType]`	By default, the return option type of a contract is inherited from its parent.
seAcquMethod	`RetVal = 2`	By default, the acquisition method for the assets on the contract is **Lease**.
seFreightOutPayer	`RetVal = [Parent.seFreightOutPayer]`	By default, whether freight out costs are payable by the lessor or the lessee is inherited from the parent contract.

Object concerned	Script	Description
seInstallCountType	`if [lModelId] > 0 Then RetVal=[Model.seSoftLicType]`	By default, software installations count type is inherited from the license type defined at the model level.
seInsurPayer	`RetVal = [Parent.seInsurPayer]`	By default, whether the insurance costs are payable by the lessor or by the lessee is inherited from the parent contract.
seIntRentType	`RetVal = [Parent.seIntRentType]`	By default, the calculation method for interim rent is inherited from the parent contract.
seLossValCalcMode	`RetVal = [Parent.seLossValCalcMode]`	By default, the calculation method for loss values is inherited from the parent contract.
sePayType	`RetVal = [Parent.sePayType]`	By default, the nature of payments of a contract is inherited from its parent.
sePeriodicity	`RetVal = 360`	By default, the frequency of payment for contract rents is annual.
sePlannedOpt	`RetVal = [Parent.sePlannedOpt]`	By default, the planned end-of-contract option is inherited from the parent contract.
seShipCostPayer	`RetVal = [Parent.seShipCostPayer]`	By default, whether the shipping costs are payable by the lessor or by the lessee is inherited from the parent contract.
seStatus	`RetVal = 0`	By default, the contract is **In preparation**.

seType	`If [lModelId] <> 0 Then`
	`RetVal = [Model.seContractType]`
	`ElseIf [Parent.seType] = 1 Then`
	`RetVal = 2`
	`Else`
	`RetVal = 0`
	`End If`
		■ If there is a model associated with the contract, then the contract type is derived from that specified at the model level.
		■ If no model is associated with the contract and the parent contract type is **Master lease**, then, by default, the contract is **Lease schedule**.
		■ In the other cases, the contract type is **Other**.

Object concerned	Script	Description
Status	`RetVal = [Parent.Status]`	By default, the status of a contract is inherited from its parent.
tsDefRenDur	`RetVal = [Parent.tsDefRenDur]`	By default, the renewal period of a contract is inherited from its parent.
tsLessorNotice	`RetVal = [Parent.tsLessorNotice]`	By default, the notification period for any modifications to the contract is inherited from the parent contract.
tsNotice	`RetVal = [Parent.tsNotice]`	By default, the notice period of a contract is inherited from its parent.
tsPurchNotice	`RetVal = [Parent.tsPurchNotice]`	By default, the minimum purchase notice period for assets before the end of a contract is inherited from the parent contract.
tsRenNotice	`RetVal = [Parent.tsRenNotice]`	By default, the minimum renewal notice period for assets before the end of a contract is inherited from the parent contract.
tsRetNotice	`RetVal = [Parent.tsRetNotice]`	By default, the minimum return notice period for assets before the end of a contract is inherited from the parent contract.
UpgOptType	`RetVal = [Parent.UpgOptType]`	By default, the upgrade option type is inherited from the parent contract.

Table 13.3. Read-Only scripts

Object concerned	Script	Description

seType	`If [Parent.seType] = 1 Then`
	`RetVal = 1`
	`Else`
	`RetVal = 0`
	`End If`
		If the parent contract is a **Master lease**, then the field containing the contract **Type** is read only.

Table 13.4. Irrelevance scripts

Object concerned	Script	Description

AssignCond	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

bAssignable	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

bPurchOpt	`RetVal = ([seType]<>1 and [seType]<>2)`	This field is only relevant if the contract is a **Master lease** or **Lease schedule**.
bRenOpt	`RetVal = ([seType]<>1 and [seType]<>2)`	This field is only relevant if the contract is a **Master lease** or **Lease schedule**.
bRetOpt	`RetVal = ([seType]<>1 and [seType]<>2)`	This field is only relevant if the contract is a **Master lease** or **Lease schedule**.
bUpgOpt	`RetVal = ([seType]<>1 and [seType]<>2)`	This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

dPurchNotice	`if amEvalScript("Irrelevant", "PurchOptType", "")<>0 OR [seType]<>2 then`
	`RetVal = 1`
	`else`
	`RetVal =0`
	`end if`
		This field is irrelevant if the **Purchase option type** is irrelevant or if the contract is not a **Lease schedule**.

dRenNotice	`if amEvalScript("Irrelevant", "RenOptType", "")<>0 OR [seType]<>2 then`
	`RetVal = 1`
	`else`
	`RetVal =0`
	`end if`
		This field is irrelevant if the **Renewal option type** is irrelevant or if the contract is not a **Lease schedule**.

dRetNotice	`if amEvalScript("Irrelevant", "RetOptType", "")<>0 OR [seType]<>2 then`
	`RetVal = 1`
	`else`
	`RetVal =0`
	`end if`
		This field is irrelevant if the **Return option type** is irrelevant or if the contract is not a **Lease schedule**.

lAssigneeId	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

Object concerned	Script	Description
lCpyId	`RetVal = (amEvalScript("Irrelevant", "Lessor", "")=FALSE)`	This field is only relevant if the **Lessor** link is irrelevant.
lDefPOrdId	`RetVal = [seType]<>6`	This field is only relevant if the contract **Type** is **Blanket PO**.

lLessorId	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

lLossValRuleId	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

lOptCmtId	`RetVal = ([seType]<>1 and [seType]<>2)`	This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

LossCond	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

lPurchOptCmtId	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bPurchOpt", "")<>0 or [bPurchOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Purchase** field is irrelevant or the assets on the contract cannot be purchased (**bPurchOpt**=0).

lRenOptCmtId	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bRenOpt", "")<>0 or [bRenOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Renewal** field is irrelevant or the assets on the contract cannot be renewed (**bRenOpt**=0).

lRetOptCmtId	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bRetOpt", "")<>0 or [bRetOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Return** field is irrelevant or the assets on the contract cannot be returned (**bRetOpt**=0).

lUpgOptCmtId	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bUpgOpt", "")<>0 or [bUpgOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Upgrade** field is irrelevant or the assets on the contract cannot be upgraded (**bUpgOpt**=0).

mMarketVal	`RetVal = [seType]<>2`	This field is only relevant if the contract **Type** is **Lease schedule**.

Object concerned	Script	Description
mPOCommitment	`RetVal = [seType]<>6`	This field is only relevant if the contract **Type** is **Blanket PO**.

pDefLRF	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

pDefRenPercent	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bRenOpt", "")<>0 or [bRenOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Renewal** field is irrelevant or the assets on the contract cannot be renewed (**bRenOpt**=0).

pIntRentPercent	`RetVal = [seType]<>1`	This field is only relevant if the contract **Type** is **Master lease**.

PurchOptType	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bPurchOpt", "")<>0 or [bPurchOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Purchase** field is irrelevant or the assets on the contract cannot be purchased (**bPurchOpt**=0).

RenOptType	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bRenOpt", "")<>0 or [bRenOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Renewal** field is irrelevant or the assets on the contract cannot be renewed (**bRenOpt**=0).

RetOptType	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bRetOpt", "")<>0 or [bRetOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Return** field is irrelevant or the assets on the contract cannot be returned (**bRetOpt**=0).

seAcquMethod	`RetVal = ([seType]<>1 and [seType]<>2)`	This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

seFreightOutPayer	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

seInsurPayer	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

seIntRentType	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

Object concerned	Script	Description

seLossValCalcMode	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

sePlannedOpt	`RetVal = ([seType]<>1 and [seType]<>2)`	This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

seShipCostPayer	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

tsDefRenDur	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bRenOpt", "")<>0 or [bRenOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Renewal** field is irrelevant or the assets on the contract cannot be renewed (**bRenOpt**=0).

tsLessorNotice	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

tsPurchNotice	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bPurchOpt", "")<>0 or [bPurchOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Purchase** field is irrelevant or the assets on the contract cannot be purchased (**bPurchOpt**=0).

tsRenNotice	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bRenOpt", "")<>0 or [bRenOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Renewal** field is irrelevant or the assets on the contract cannot be renewed (**bRenOpt**=0).

tsRetNotice	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bRetOpt", "")<>0 or [bRetOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Return** field is irrelevant or the assets on the contract cannot be returned (**bRetOpt**=0).

UpgOptType	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bUpgOpt", "")<>0 or [bUpgOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Upgrade** field is irrelevant or the assets on the contract cannot be upgraded (**bUpgOpt**=0).

Object concerned	Script	Description

Assignee	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

AstCntrDescs	`RetVal = ([seType]=1 or [seType]=2 or [seType]=6)`	This field is irrelevant if the contract **Type** is **Master lease**, **Lease schedule** or **Blanket PO**.
Company	`RetVal = (amEvalScript("Irrelevant", "Lessor", "")=FALSE)`	This field is only relevant if the **Lessor** link is irrelevant.
DefPOrder	`RetVal = [seType]<>6`	This field is only relevant if the contract **Type** is **Blanket PO**.
ExpenseLines	`RetVal = ([seType]=1)`	This field is irrelevant if the contract **Type** is **Master lease**.
LeasedAssets	`RetVal = [seType]<>2`	This field is only relevant if the contract **Type** is **Lease schedule**.

Lessor	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

Licenses	`RetVal = [seType]<>5`	This field is only relevant if the contract **Type** is **License**.
Loans	`RetVal = ([seType]=1 or [sePayType]=-1 or [sePayType]=0)`	This field is irrelevant if the contract **Type** is **Master lease**, or if the **Nature of payments** is **None** or **Rents**.

LossValRule	`RetVal = 0`
	`if [seType]<>1 and [seType]<>2 then`
	`RetVal = 1`
	`end if`
		This field is only relevant if the contract is a **Master lease** or **Lease schedule**.

OptCmt	`RetVal = ([seType]<>1 and [seType]<>2)`	This field is only relevant if the contract is a **Master lease** or **Lease schedule**.
POrdersBlanketPO	`RetVal = [seType]<>6`	This field is only relevant if the contract **Type** is **Blanket PO**.

PurchOptCmt	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bPurchOpt", "")<>0 or [bPurchOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Purchase** field is irrelevant or the assets on the contract cannot be purchased (**bPurchOpt**=0).

Object concerned	Script	Description

RenOptCmt	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bRenOpt", "")<>0 or [bRenOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Renewal** field is irrelevant or the assets on the contract cannot be renewed (**bRenOpt**=0).

Rents	`RetVal = ([seType]=1 or [sePayType]=-1 or [sePayType]=1)`	This field is irrelevant if the contract **Type** is **Master lease**, or if the **Nature of payments** is **None** or **Rents**.

RetOptCmt	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bRetOpt", "")<>0 or [bRetOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Return** field is irrelevant or the assets on the contract cannot be returned (**bRetOpt**=0).

Schedules	`RetVal = [seType]<>1`	This field is only relevant if the contract **Type** is **Master lease**.

UpgOptCmt	`RetVal = 0`
	`if amEvalScript("Irrelevant", "bUpgOpt", "")<>0 or [bUpgOpt]=0 then`
	`RetVal = 1`
	`end if`
		This field is irrelevant if the **Upgrade** field is irrelevant or the assets on the contract cannot be upgraded (**bUpgOpt**=0).

WorkOrders	`RetVal = [seType]<>4`	This field is only relevant if the contract **Type** is **Maintenance**.

Integrity rules

Name of the rule	List of monitored objects	Rule(s) verified	List of any modified objects
CNotifContDateInteg	■ SyncRead on object: dEnd ■ SyncRead on object: dPurchNotice ■ SyncRead on object: tsPurchNotice	Ensures the consistency between the **End date**, the **Purchase notif. date** and the **Purchase notice period** of the contract. The contract end date is always kept. The last value entered by the user is the kept to the detriment of the remaining value.	■ dPurchNotice ■ tsPurchNotice

Name of the rule	List of monitored objects	Rule(s) verified	List of any modified objects
CNotifContDateInteg	■ SyncRead on object: dEnd ■ SyncRead on object: dRenNotice ■ SyncRead on object: tsRenNotice	Ensures the consistency between the **End date**, the **Renewal notice. date** and the **Renewal notice period** of the contract. The contract end date is always kept. The last value entered by the user is the kept to the detriment of the remaining value.	■ dRenNotice ■ tsRenNotice
CNotifContDateInteg	■ SyncRead on object: dEnd ■ SyncRead on object: dRetNotice ■ SyncRead on object: tsRetNotice	Ensures the consistency between the **End date**, the **Return notice. date** and the **Return notice period** of the contract. The contract end date is always kept. The last value entered by the user is the kept to the detriment of the remaining value.	■ dRetNotice ■ tsRetNotice

Agents

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CContDateInteg	■ SyncRead on object: dStart ■ SyncRead on object: dEnd ■ SyncRead on object: tsDuration	Ensures the consistency between the start and end dates of the contract and its length. The last value entered by the user is kept to the detriment of the others.	■ dEnd ■ dStart ■ tsDuration

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CContractInherit3rdPartyAgent	◆ Insert on object: amContract	On creating a contract, the information on third-party companies is copied over from the parent, if there is one.	
CContractLink3rdPartyAgent	■ PostUpdate on object: lLessorId ■ PostUpdate on object: lAssigneeId	Makes sure that these two links belong to the list of third-party companies of the contract.	
CDateAlarmAgent	◆ PostUpdate on object: dRenNotice	This agent, if necessary, recalculates the alarms associated with the contract renewal notification date.	
CDateAlarmAgent	◆ PostUpdate on object: dRetNotice	This agent, if necessary, recalculates the alarms associated with the contract return notification date.	
CDateAlarmAgent	◆ PostUpdate on object: dEnd	This agent, if necessary, recalculates the alarms associated with the contract end date.	
CDateAlarmAgent	◆ PostUpdate on object: dPurchNotice	This agent, if necessary, recalculates the alarms associated with the contract purchase (buyout) notification date.	

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CGbAssetAssignement	■ Insert on object: amCable ■ Insert on object: amContract ■ Insert on object: amComputer ■ Insert on object: amSoftInstall ■ Insert on object: amAsset ■ Insert on object: amPortfolio ■ Insert on object: amTraining ■ Insert on object: amWorkOrder ■ Insert on object: amPhone ■ PostDelete on object: amPortfolio ■ PreUpdate on object: lModelId ■ PreUpdate on object: lModelId ■ PreUpdate on object: lModelId ■ PreUpdate on object: lModelId ■ PreUpdate on object: lModelId	In case of creation of a portfolio item, if necessary this agent creates the corresponding record in the Assets table and the appropriate record in the overflow table matching the management constraint associated with the model of the portfolio item.	

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CLSLossLineAgent	■ Insert on object: amContract ■ PostDelete on object: amPortfolio ■ PostUpdate on object: dStart ■ PostUpdate on object: dEnd ■ PostUpdate on object: mMarketVal ■ PostUpdate on object: lLossValRuleId	Applies the loss-value rule according to the monitored fields. The contract loss-value records are created.	
ContCompany	■ PreUpdate on object: seType ■ PreUpdate on object: lCpyId ■ PreUpdate on object: lLessorId	If the contract **Type** is neither **Master lease**, nor **Lease schedule**, the link to the lessor is deleted. For other contract types, any change to the lessor is carried over to the **Company** with which the contract is signed.	■ lCpyId ■ lLessorId
ContVersInit	❖ PreUpdate on object: mIntPayAst	Propagates modifications to the total of initial payments for assets financed by the contract to the initial payment of the contract.	■ IntPayCur ■ mIntPay
COverflow-ChangeAgent	❖ PreUpdate on object: lModelId	This agent stops a contract model from being changed if this means changing the associated overflow table.	

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
FullName agent	■ Insert on object: amContract ■ PostUpdate on object: Ref ■ PostUpdate on object: lParentId ■ PreUpdate on object: Ref ■ PreUpdate on object: lParentId	This agent manages tree structures in hierarchic tables. In the Contracts table, it maintains the integrity of the hierarchical structure. The full name of the contract and its hierarchical level are recalculated if: ■ the contract reference code is modified ■ its parent is modified	■ FullName ■ sLvl
RentContract	■ Insert on object: amContract ■ Insert on object: amCntrRent ■ PostUpdate on object: sePeriodicity ■ PostUpdate on object: mAmount ■ PostUpdate on object: mPayments ■ PostUpdate on object: sePeriodicity ■ PostUpdate on object: bMainRent	Ensures the consistency of the frequency of payment and the full amount between the contract and the main rent.	

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
VersInitExpLine	■ Insert on object: amContract ■ PostUpdate on object: mIntPay ■ PostUpdate on object: mIntPayAst ■ PostUpdate on object: dStart	If the initial payment of the contract, the total of initial payments for the assets on the contract or the contract start date is modified, this agent updates or creates the expense line corresponding to the initial payment of the contract. In particular, if there is a difference between the initial payment of the contract and the sum of initial payments of the assets on the contract a compensating expense line is created or updated.	

14 CHAPTER Cost Centers table (amCostCenter)

This chapter provides an exhaustive list of all the mechanisms dealing with the Cost Centers table. Each section deals with a different type of automatic mechanism.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 14.1. Validity scripts on the table

Script	Description
`If Not IsEmpty([dEnd]) and Not IsEmpty([dStart]) and [dStart] > [dEnd] Then Err.Raise(-2009, "The end date (dEnd) must be greater than or equal to the start date (dStart).") RetVal = FALSE Else RetVal = TRUE End If`	If the creation and end dates of the cost center are not empty and the end date comes before the creation date, the record is rejected.

Table 14.2. Default value scripts

Object concerned	Script	Description
Code	`RetVal = "C" + AmCounter("amCostCenter_Code", 6)`	By default, the unique code of a cost center is the concatenation of the letter **C** and the value of the **amCostCenter_Code** counter on 6 figures.
dRecalcFrom	`RetVal = AmDate()`	By default, the date from which the expense lines of the cost center are to be split is the record creation date.
dRecalcTo	`RetVal = AmDate()`	By default, the date up until which the expense lines of the cost center are to be split is the record creation date.
dStart	`RetVal = AmDate()`	By default, the cost center creation date is the record creation date.

Integrity rules

There are no integrity rules on the Cost Centers table (**amCostCenter**).

Agents

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
FullName agent	■ Insert in the **amCostCenter** table ■ Post-Update on the **Code** field ■ Post-Update on the **lParentId** link ■ Pre-Update on the **Code** field ■ Post-Update on the **lParentId** link	This agent manages tree structures in hierarchic tables. In the Brands table, it maintains hierarchical integrity in the case of sub-cost centers. The full name of the brand and its hierarchical level are recalculated if: ■ A cost center is created ■ The code of the cost center is modified ■ The parent cost center is modified	■ **FullName** ■ **sLvl**

CHAPTER 15 Departments and Employees table (amEmplDept)

This chapter provides an exhaustive list of all the mechanisms dealing with the Departments and Employees table. Each section deals with a different type of automatic mechanism.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 15.1. Default value scripts

Object concerned	Script	Description
BarCode	`RetVal = "U" + AmCounter(" amEmplDept_BarCode", 6)`	By default, the barcode associated with a the employee or department is the concatenation of the letter **U** and the value of the **amEmplDept_BarCode** counter on 6 figures.
dHire	`RetVal = AmDate()`	By default, the hire date is the record creation date.
EMail	`RetVal = [Parent.EMail]`	By default, this field, which contains the e-mail of the employee or department, takes the same value as the e-mail of its parent.
Fax	`RetVal = [Parent.Fax]`	By default, this field, which contains the fax number of the employee or department, takes the same value as the fax number of its parent.
IDNo	`if [bDepartment]=0 Then RetVal = "U" + AmCounter(" amEmplDept_BarCode", 6) End If`	In the case of employees only, by default, the employee ID is the concatenation of the letter **U** and the value of the **amEmplDept_BarCode** counter on 6 figures.
lCostId	`RetVal = [Parent.lCostId]`	By default, this field, which contains the identifier of the cost center of the employee or department, takes the same value as the identifier of its parent.
lIconId	`RetVal = [Parent.lIconId]`	By default, this field, which contains the identifier of the icon used to represent the department or employee, takes the same value as the identifier of its parent.

Object concerned	Script	Description
lLocaId	`RetVal = [Parent.lLocaId]`	By default, this field, which contains the identifier of the location of the employee or department, takes the same value as the identifier of its parent.
lSupervId	`RetVal = [Parent.lSupervId]`	By default, this field, which contains the identifier of the supervisor of the employee or department, takes the same value as the identifier of its parent.
Phone	`RetVal = [Parent.Phone]`	By default, this field, which contains the telephone number of the employee or department, takes the same value as the telephone number of its parent.

Table 15.2. Irrelevance scripts

Object concerned	Script	Description
bAdminRight	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
bCanReadArchive	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
bHDAdmin	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
bIsRCHotliner	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
bIsRCManager	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
dHire	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
dLeave	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
FirstName	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
FirstName2	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
HomePhone	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.

Object concerned	Script	Description
Identifier	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
lDefCurId	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
lLoginActId	`RetVal = (0<>[bDepartment]) OR (""=[UserLogin])`	This field is only relevant in the case of a department or if the user login is empty.
LoginPassword	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
lPhotoId	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
lProfileId	`RetVal = (0<>[bDepartment] OR 0<>[bAdminRight])`	This field is only relevant in the case of a department or if the user has administration rights.
lSupervId	`RetVal = (0=[bDepartment])`	This field is only relevant in the case of a department.
MailLogin	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
MailPassword	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
MobilePhone	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
MrMrs	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
seLoginClass	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
Title	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
UserDesc	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
UserDomain	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
UserLogin	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
UserName	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
Absences	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
DefCurrency	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
EmplGroups	`RetVal = (0<>[bDepartment])`	This field is irrelevant in the case of a department.

Object concerned	Script	Description
Entitlement	`RetVal = (0<>[bDepartment])`	This field is irrelevant in the case of a department.
LoginAction	`RetVal = (0<>[bDepartment]) OR (""=[UserLogin])`	This field is only relevant in the case of a department or if the user login is empty.
Photo	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.
Profile	`RetVal = (0<>[bDepartment] OR 0<>[bAdminRight])`	This field is only relevant in the case of a department or if the user has administration rights.
Supervisor	`RetVal = (0=[bDepartment])`	This field is only relevant in the case of a department.
Trainings	`RetVal=(0<>[bDepartment])`	This field is irrelevant in the case of a department.

Integrity rules

Name of the rule	List of monitored objects	Rule(s) verified	List of any modified objects
CPasswordInteg	■ SyncRead on object: UserLogin ■ SyncRead on object: LoginPassword	When an employee's login is updated, this integrity rule empties the password.	◆ LoginPassword

Name of the rule	List of monitored objects	Rule(s) verified	List of any modified objects
CPasswordInteg	■ SyncRead on object: MailLogin ■ SyncRead on object: MailPassword	When an employee's mail is updated, this integrity rule empties the password.	◆ MailPassword

Agents

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CAdminLoginAgent	■ PreDelete on object: amEmplDept ■ PreUpdate on object: UserLogin ■ PreUpdate on object: seLoginClass ■ PreUpdate on object: bAdminRight	Forbids the Admin user from doing the following operations: ■ Archival ■ Deletion ■ Modifying login ■ Modifying login type ■ Modifying administration rights	
CGBLoginNumberCheck	■ PreUpdate on object: UserLogin ■ PreUpdate on object: seLoginClass	Makes sure the number of named users is not exceeded. If this the case, the login being edited is turned into a concurrent user.	◆ seLoginClass
CGbPerson2Service	■ Insert on object: amEmplDept ■ PreUpdate on object: lParentId	Changes an employee, created on the fly or imported, into a department if another employee or department is attached to them.	

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
FullName agent	■ Insert on object: amEmplDept ■ PostUpdate on object: Name ■ PostUpdate on object: FirstName ■ PostUpdate on object: IDNo ■ PostUpdate on object: bDepartment ■ PostUpdate on object: lParentId ■ PreUpdate on object: Name ■ PreUpdate on object: FirstName ■ PreUpdate on object: IDNo ■ PreUpdate on object: bDepartment ■ PreUpdate on object: lParentId	This agent manages tree structures in hierarchic tables. In the Departments and Employees table, it ensures the consistency of the hierarchy. The full name of the employee or the service and their hierarchical level are recalculated if: ■ the name of the employee or department is modified ■ the first name of the employee is modified ■ the Employee ID of the employee is modified ■ the employee record is converted to a department or vice-versa ■ its parent is modified	■ FullName ■ sLvl

16 CHAPTER Locations table (amLocation)

This chapter provides an exhaustive list of all the mechanisms dealing with the Locations table. Each section deals with a different type of automatic mechanism.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 16.1. Default value scripts

Object concerned	Script	Description
Address1	`RetVal = [Parent.Address1]`	By default, the address of a location is identical to that of its parent location.
Address2	`RetVal = [Parent.Address2]`	By default, the address of a location is identical to that of its parent location.
BarCode	`RetVal = "L" + AmCounter("amLocation_BarCode", 6)`	By default, the unique code of the location is the concatenation of the letter **L** and the value of the **amLocation_BarCode** counter on 6 figures.
City	`RetVal = [Parent.City]`	By default, the city of a location is identical to that of its parent location.
lCostId	`RetVal = [Parent.lCostId]`	By default, the cost center of a location is identical to that of its parent location.
lCountryId	`RetVal = [Parent.lCountryId]`	By default, the country of a location is identical to that of its parent location.
lIconId	`RetVal = [Parent.lIconId]`	By default, this field, which contains the identifier of the icon used to represent the location, inherits the same value from its parent location.
lStockUsedId	`RetVal = [Parent.lStockUsedId]`	By default, the stock serving a location is identical to that of its parent location.
lTaxJurisId	`RetVal = [Parent.lTaxJurisId]`	By default, the jurisdiction of a location is identical to that of its parent location.

Name	`RetVal = ""`
	`if 0<>[lSocId] then`
	`RetVal = [Company.Name]`
	`end if`
		By default, if the location is a company site, it inherits its name from the company. Otherwise, the name is left empty.

State	`RetVal = [Parent.State]`	By default, the state of a location is identical to that of its parent location.

Object concerned	Script	Description
ZIP	`RetVal = [Parent.ZIP]`	By default, the postal code of a location is identical to that of its parent location.

Integrity rules

There are no integrity rules on the Locations table (**amLocation**).

Agents

The following table lists the agents working on the Locations table (**amLocation**).

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
FullName agent	■ Insert in the **amLocation** table ■ Post-Update on the **Name** field ■ Post-Update on the **lParentId** link ■ Pre-Update on the **Name** field ■ Pre-Update on **lParentId** link	This agent manages tree structures in hierarchic tables. In the Locations table, it maintains hierarchical integrity in the case of sub-locations. The full name of the location and its hierarchical level are recalculated if: ■ A location is created ■ The name of the location is modified ■ The parent location is modified	■ **FullName** ■ **sLvl**

Workflows

The following table summarizes the workflows operating on the Locations table (**amLocation**).

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Determining the workflows used for a table [page 183] at the end of this document.

Workflow reference	Workflow type	Description
PROP_ADDR	Synchronous	This workflow is triggered if the address of a location is modified (**Address1**, **Address2**, **City**, **Country**, **State**, **ZIP** fields). It propagates the modifications to the sub-locations.

17 CHAPTER Models table (amModel)

This chapter provides an exhaustive list of all the mechanisms dealing with the Models table. Each section deals with a different type of automatic mechanism.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 17.1. Default value scripts

Object concerned	Script	Description
AcctCode	`RetVal = [Parent.AcctCode]`	By default, the accounting code of the model is that of the model.
BarCode	`RetVal = "M" + AmCounter("amModel_BarCode", 6)`	By default, the barcode of the model is the concatentation of the letter **M** and the value of the **amModel_BarCode** counter on 6 figures.
bInvent	`RetVal = 1`	By default, the model is inventories during barcode inventories.
Certification	`RetVal = [Parent.Certification]`	By default, the certification is inherited from the parent model.
fCountFactor	`RetVal = 1`	By default, the number of points to be counted by installation or utilization of the model is set to 1.
fRoundingQty	`RetVal = 0`	By default, no roundings are tolerated for the quantities linked to the model.
fUseQty	`RetVal = 1`	By default, the indivisible quantity of the model is set to 1. This quantity enables you to specify the fraction used to divide batches created from the model.
lBrandId	`RetVal = [Parent.lBrandId]`	By default, this field, which contains the identifier of the brand of the model, takes the same value as the identifier of the brand of the parent model.
lIconId	`RetVal = [Parent.lIconId]`	By default, this field, which contains the identifier of the icon used to represent the model, takes the same value as the identifier of the icon of the parent model.

Object concerned	Script	Description
lNatureId	`RetVal = [Parent.lNatureId ]`	By default, this field, which contains the identifier of the nature of the model, takes the same value as the identifier of the nature of the parent model.
lUseUnitId	`RetVal = [Parent.lUseUnitI d]`	By default, this field, which contains the identifier of the unit of the model, takes the same value as the identifier of the unit of the parent model.
Prefix	`RetVal = [Parent.Prefix]`	By default, the prefix of the model is that of the model.

pTaxRate	`RetVal = 19.6/100`
	`if [lParentId] <> 0 then`
	`RetVal = [Parent.pTaxRate]`
	
	`end if`
		By default, the applicable tax rate for the model is 7.75%. If the model has a parent model, it inherits its tax rate.

seContractType	`RetVal = [Nature.seCntrTyp e]`	By default, the contract type associated with the model is inherited from the nature of the model.
seDevSdType	`RetVal = 0`	Used for Cable only. In this case, by default, the model represents a single-sided device.
seDevType	`RetVal = 0`	Used for Cable only. In this case, by default, the model represents an active device.
seSoftLicMulti	`RetVal = 0`	By default, software based on this model can be installed on one single computer.
seSoftLicType	`RetVal=3`	By default, the license type associated with the model is **Not defined**.

Table 17.2. Mandatory scripts

Object concerned	Script	Description
BarCode	`RetVal = (0<>[bInvent])`	This field must be populated if the model is to be inventoried in barcode inventories.

Table 17.3. Irrelevance scripts

Object concerned	Script	Description
bSpeaker	`RetVal = ("amPhone"<>[Nature.OverflowTbl])`	This field is only relevant if the nature of the model creates a telephone.
bVoiceMail	`RetVal = ("amPhone"<>[Nature.OverflowTbl])`	This field is only relevant if the nature of the model creates a telephone.
CableType	`RetVal = (8<>[Nature.seBasis])`	This field is only relevant if the model is intended to create a cable.
Certification	`RetVal = (0=[bRequestable])`	This field is only relevant if the model can be included in a purchase request.
ContractNature	`RetVal = (4<>[Nature.seBasis])`	This field is only relevant if the model is intended to create a contract.
CPUType	`RetVal = (1<>[Nature.seBasis]) OR ("amComputer"<>[Nature.OverflowTbl])`	This field is only relevant if the model is intended to create a portfolio item or if the nature of the model creates a computer.
dCertifEnd	`RetVal = (0=[bRequestable])`	This field is only relevant if the model can be included in a purchase request.
dCertification	`RetVal = (0=[bRequestable])`	This field is only relevant if the model can be included in a purchase request.

DeviceType	`RetVal = 1`
	`' Must be an Asset and a Device`
	`if [Nature.seBasis] = 1 and [Nature.bDevice] = 1 then`
	`RetVal = 0`
	`end if`
		This field is only relevant if the model is intended to create a porfolio item that is a cable device.

fCountFactor	`RetVal = ("amSoftInstall"<>[Nature.OverflowTbl])`	This field is only relevant if the nature of the model creates a software installation.

Object concerned	Script	Description

fRoundingQty	`RetVal = 1`
	`' Must be a bulk asset or a Cable (length)`
	`if ([Nature.seBasis] = 1 and [Nature.seMgtConstraint]<>2) or ([Nature.seBasis] = 8) then`
	`RetVal = 0`
	`end if`
		This field is only relevant if the model is intended to create a portfolio item whose management constraint is **Free** or **Unique asset tag**, or is a cable.

fUseQty	`RetVal = 1`
	`' Must be a bulk asset or a Cable (length)`
	`if ([Nature.seBasis] = 1 and [Nature.seMgtConstraint]<>2) or ([Nature.seBasis] = 8) then`
	`RetVal = 0`
	`end if`
		This field is only relevant if the model is intended to create a cable or a porfolio item that is a cable device.

InstLanguage	`RetVal=([Nature.seOverflowTbl]<>3)`	This field is only relevant if the nature of the model creates a software installation.
lColorCodeId	`RetVal = (8<>[Nature.seBasis])`	This field is only relevant if the model is intended to create a cable.
lCPUSpeedMHz	`RetVal = (1<>[Nature.seBasis]) OR ("amComputer"<>[Nature.OverflowTbl])`	This field is only relevant if the model is intended to create a portfolio item or if the nature of the model creates a computer.
lDiskSizeMb	`RetVal = (1<>[Nature.seBasis]) OR ("amComputer"<>[Nature.OverflowTbl])`	This field is only relevant if the model is intended to create a portfolio item or if the nature of the model creates a computer.
LicLanguage	`RetVal=([Nature.bSoftLicense]=0)`	This field is only relevant if the nature of the model creates a software license.

lLabelRuleId	`RetVal = 1`
	`' Must be a cable or (an asset and a device)`
	`if ([Nature.seBasis] = 8) or ([Nature.seBasis] = 1 and [Nature.bDevice] = 1) then`
	`RetVal = 0`
	`end if`
		This field is only relevant if the model is intended to create a cable or a porfolio item that is a cable device.

Object concerned	Script	Description
lMemorySizeMb	`RetVal = (1<>[Nature.seBasis]) OR ("amComputer"<>[Nature.OverflowTbl])`	This field is only relevant if the model is intended to create a portfolio item or if the nature of the model creates a computer.

lPins	`RetVal = 1`
	`' Must be an Asset and a Device`
	`if [Nature.seBasis] = 1 and [Nature.bDevice] = 1 then`
	`RetVal = 0`
	`end if`
		This field is only relevant if the model is intended to create a porfolio item that is a cable device.

lSoftLicUseRights	`RetVal = (0=[Nature.bSoftLicense])`	This field is only relevant if the nature of the model creates a software license.

lUseUnitId	`RetVal = 1`
	`' Must be a bulk asset or a Cable (length)`
	`if ([Nature.seBasis] = 1 and [Nature.seMgtConstraint]<>2) or ([Nature.seBasis] = 8) then`
	`RetVal = 0`
	`end if`
		This field is only relevant if the model is intended to create a portfolio item whose management constraint is **Free** or **Unique asset tag**, or is a cable.

lWOCalendarId	`RetVal = (3<>[Nature.seBasis])`	This field is only relevant if the model is intended to create a work order.
seAuthorization	`RetVal = ("amSoftInstall"<>[Nature.OverflowTbl])`	This field is only relevant if the nature of the model creates a software installation.
seContractType	`RetVal = (4<>[Nature.seBasis])`	This field is only relevant if the model is intended to create a contract.

seDevSdType	`RetVal = 1`
	`' Must be an Asset and a Device`
	`if [Nature.seBasis] = 1 and [Nature.bDevice] = 1 then`
	`RetVal = 0`
	`end if`
		This field is only relevant if the model is intended to create a porfolio item that is a cable device.

seDevType	`RetVal = 1`
	`' Must be an Asset and a Device`
	`if [Nature.seBasis] = 1 and [Nature.bDevice] = 1 then`
	`RetVal = 0`
	`end if`
		This field is only relevant if the model is intended to create a porfolio item that is a cable device.

Object concerned	Script	Description
seSoftLicType	`RetVal = (0=[Nature.bSoftLicense])`	This field is only relevant if the nature of the model creates a software license.
seWOType	`RetVal = (3<>[Nature.seBasis])`	This field is only relevant if the model is intended to create a work order.
SoftMedia	`RetVal = (0=[Nature.bSoftLicense])`	This field is only relevant if the nature of the model creates a software license.
SoftOS	`RetVal = (0=[Nature.bSoftLicense])`	This field is only relevant if the nature of the model creates a software license.
tsCntrDuration	`RetVal = (4<>[Nature.seBasis])`	This field is only relevant if the model is intended to create a contract.
tsTrngDuration	`RetVal = (6<>[Nature.seBasis])`	This field is only relevant if the model is intended to create a training.
tsWOSchedFixDelay	`RetVal = (3<>[Nature.seBasis])`	This field is only relevant if the model is intended to create a work order.
tsWOSchedFixDur	`RetVal = (3<>[Nature.seBasis])`	This field is only relevant if the model is intended to create a work order.
VersionLevel	`RetVal = ("amSoftInstall"<>[Nature.OverflowTbl])`	This field is only relevant if the nature of the model creates a software installation.
WOPriority	`RetVal = (3<>[Nature.seBasis])`	This field is only relevant if the model is intended to create a work order.
ColorCode	`RetVal = (8<>[Nature.seBasis])`	This field is only relevant if the model is intended to create a cable.
FieldAdjustTempls	`RetVal = (99=[Nature.seBasis])`	This field is only relevant if the model creates nothing.

LabelRule	`RetVal = 1`
	`' Must be a cable or (an asset and a device)`
	`if ([Nature.seBasis] = 8) or ([Nature.seBasis] = 1 and [Nature.bDevice] = 1) then`
	`RetVal = 0`
	`end if`
		This field is only relevant if the model is intended to create a cable or a porfolio item that is a cable device.

Object concerned	Script	Description
LicenseSoftInfos	`RetVal = (0=[Nature.bSoftLicense])`	This field is only relevant if the nature of the model creates a software license.

ModelSlots	`RetVal = 1`
	`' Must be an Asset and a Device`
	`if [Nature.seBasis] = 1 and [Nature.bDevice] = 1 then`
	`RetVal = 0`
	`end if`
		This field is only relevant if the model is intended to create a porfolio item that is a cable device.

Pairs	`RetVal = (8<>[Nature.seBasis])`	This field is only relevant if the model is intended to create a cable.

Ports	`RetVal = 1`
	`' Must be connectable`
	`if [Nature.seBasis] = 1 and [Nature.bIsCnxClient] > 0 then`
	`RetVal = 0`
	`end if`
		This field is only relevant if the model is intended to create a porfolio item that can be connected

SoftwareSoftInfos	`RetVal = ("amSoftInstall"<>[Nature.OverflowTbl])`	This field is only relevant if the nature of the model creates a software installation.

UseUnit	`RetVal = 1`
	`' Must be a bulk asset or a Cable (length)`
	`if ([Nature.seBasis] = 1 and [Nature.seMgtConstraint]<>2) or ([Nature.seBasis] = 8) then`
	`RetVal = 0`
	`end if`
		This field is only relevant if the model is intended to create a portfolio item whose management constraint is **Free** or **Unique asset tag**, or is a cable.

WOCalendar	`RetVal = (3<>[Nature.seBasis])`	This field is only relevant if the model is intended to create a work order.

Agents

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CBiSoftInteg	■ ASyncRead on object: Nature.bSoftLicense ■ SyncRead on object: lSoftLicUseRights ■ SyncRead on object: seSoftLicType ■ SyncRead on object: seSoftLicMulti	The following rules are enforced for a model for which the nature is a software license: ■ If the license is not Multiple-user (**seSoftLicMulti**, the number of users for the license (**lSoftLicUserRights**) is forced to 1. The license type (**seSoftLicType**) is forced to **Per named workstation**. ■ If the number of users of the license is greater than 1, the license becomes Multiple-user.	■ lSoftLicUseRights ■ seSoftLicMulti ■ seSoftLicType
COverflowChangeAgent	❖ PreUpdate on object: lNatureId	This agent stops the nature of a model from being changed if doing so implies the associated overflow table being changed also.	

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
FullName agent	■ Insert on object: amModel ■ PostUpdate on object: Name ■ PostUpdate on object: lParentId ■ PreUpdate on object: Name ■ PreUpdate on object: lParentId	This agent manages tree structures in hierarchic tables. In the Models table, it maintains the integrity of the hierarchical structure. The full name of the product and its hierarchical level are recalculated if: ■ the name of the model is modified ■ its parent is modified	■ FullName ■ sLvl

18 CHAPTER Portfolio Items table (amPortfolio)

This chapter provides and exhaustive list of all the mechanisms dealing with the Portfolio Items table. Each section deals with a different type of automatic mechanism.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 18.1. Validity scripts on the table

Script	Description
`If IsEmpty([dAssignment]) and [seAssignment]=0 Then Err.Raise(-2009, "Since it is no longer in stock, you must specify an assignment (in-service date) for this asset.") RetVal = FALSE Else RetVal = TRUE End If`	If the item is **In use** (`[seAssignment]=0`), you must specify an in-service date. Otherwise the record is rejected.

Table 18.2. Default value scripts

Object concerned	Script	Description
AssetTag	`RetVal = [Asset.AssetTag]`	By default, the asset tag of a portfolio item is that of the associated asset.
AvgPriceCur	`RetVal = AmDefaultCurrency ()`	By default, this field is set to the value of the default currency.
bUseQty	`if [Model.Nature.seMgtConstraint]=2 OR [lModelId]=0 Then RetVal = 0 else RetVal = 1 End if`	If the management constraint of the nature of the model associated with the portfolio item is set to Unique asset tag or the identifier of the associated model is null, then the portfolio item does not have an associated quantity.
Code	`RetVal = AmCounter("amAssignment_Code", 6)`	By default, this field is set to the value of the **amAssignment_Code** counter on 6 figures.
dAssignment	`' Do we assign it now ? if 1<>[seAssignment] then RetVal = AmDate() end if`	If the portfolio item is not set to **In stock**, it is in service and the in-service date is populated with the current date.
dtAvgPriceCv	`RetVal = AmDate()`	By default, the conversion date of the unit value is the current date.

Object concerned	Script	Description

fQty	`RetVal = 1`
	`if 0<>[lAstId] then`
	`RetVal = (amDbGetDouble("S elect SUM(fTotalQty) FROM amAsset WHERE lAstId=" & [ lAstId])) - (amDbGetDouble ("Select SUM(fQty) FROM am Portfolio WHERE lAstId=" & [lAstId]))`
	`end if`
		■ When no asset is associated with the portfolio item, the number of units in the batch is set to 1.
		■ When an asset is associated with the portfolio item, this field is set to the difference between the total quantity of units in the batch and the number of units in the batch.

lCostCatId	`RetVal = [Model.lCostCatId ]`	By default, the cost category associated with the portfolio item is that of the model.

lCostId	`if [lParentId]=0 Then`
	`RetVal = [User.lCostId]`
	`else`
	`RetVal = [Parent.lCostId]`
	`End if`
		■ If the portfolio item has a parent item, its cost center is that of the parent.
		■ Otherwise, the cost center is that of the user of the portfolio item.

lIconId	`RetVal = [Model.lIconId]`	By default, this field, which contains the identifier of the icon used to represent the portfolio item, inherits the same value as that of the model from which it is derived.

Object concerned	Script	Description
lLocaId	`if (0<>[lParentId]) AND (0 <>[Parent.lLocaId]) then RetVal = [Parent.lLocaId] elseif (0<>[lUserId]) AND (0<>[User.lLocaId]) then RetVal = [User.lLocaId] elseif (0<>[lStockId]) AND (0<>[Stock.lStockId]) then RetVal = [Stock.lLocaId] end if`	■ If the portfolio item has a parent item and a location is defined for the parent, then the location of the portfolio item is that if its parent. ■ If this is not the case and the portfolio item has a user with a defined location, then the location of item is set to that of its user. ■ Otherwise, if the item has a stock that is associated with a location, then the location of the item is that of the stock.
lModelId	`RetVal = [Asset.lModelId]`	By default, the model is that of the asset associated with the portfolio item.
lStockId	`RetVal = [Location.lStockUsedId]`	By default, the stock is that of the location of the portfolio item.
lSupervId	`RetVal = [Parent.lSupervId]`	By default, the supervisor is that of the parent portfolio item.
lUserId	`RetVal = [Parent.lUserId]`	By default, the user is that of the parent portfolio item.

Table 18.3. Mandatory scripts

Object concerned	Script	Description
lStockId	`RetVal = (1 = [seAssignment])`	If the **Assignment** of the portfolio item is **In stock** then is mandatory to specify a stock for the portfolio item.

Table 18.4. Read-Only scripts

Object concerned	Script	Description
fQty	`RetVal = (2=[Model.Nature.seMgtConstraint] OR [lModelId]=0 OR [lAstId]<>0)`	If the management constraint of the nature of the model associated with the portfolio item is set to **Unique asset tag**, then the number of units in the batch cannot be modified.

Table 18.5. Irrelevance scripts

Object concerned	Script	Description
AssetTag	`RetVal = (2<>[Model.Nature.seMgtConstraint] OR [lModelId]=0)`	This field is irrelevant if the item is not managed with a **Unique asset tag**. It is not displayed in this case.
bUseQty	`RetVal = (2=[Model.Nature.seMgtConstraint] OR [lModelId]=0)`	This field is irrelevant if the item is not managed with a **Unique asset tag**. It is not displayed in this case.

Folder	`if [Model.Nature.OverflowTbl] = "amSoftInstall" then`
	
	`RetVal = 0`
	`else`
	`RetVal = 1`
	`end if`
		This field, which stores the name of the installation folder of the software, is irrelevant if the corresponding item is not a software installation.

lAstId	`RetVal = (0=[lAstId] OR [fQty] <> [Asset.fTotalQty])`	If there is not asset associated with the portfolio item or the number of units in the batch is different from the total number of units in a batch then the link to an asset is irrelevant.

lLocaId	`'Relevant when in stock or assigned`
	`RetVal = 0`
	`if [seAssignment]<>0 and [seAssignment]<>1 then`
	`RetVal = 1`
	`end if`
		The link to a reservation is only relevant if the portfolio item is **In stock** or **In use**.

Object concerned	Script	Description

lStockId	`'Relevant when in stock or waiting to enter stock`
	`RetVal = 0`
	`if [seAssignment]<>1 and [seAssignment]<>3 then`
	`RetVal = 1`
	`end if`
		The link to a stock is only relevant if the portfolio item is **In stock** or **Awaiting receipt**.

lUserId	`RetVal = (amEvalScript("Irrelevant", "Stock", "")=FALSE OR [seAssignment]=2)`	The link to a user is only relevant if the portfolio item is not **In stock** or **Retired**.
lWorkOrderId	`RetVal = (1<>[Model.Nature.bConsumable])`	The link to a work order is only relevant if the portfolio item is a consumable.
RMANumber	`RetVal = [seAssignment]<>4`	This field, which contains the RMA number, is only relevant if the **Assignment** field of the portfolio item is **Return for maintenance**.
AddOn	`RetVal = (2<>[Model.Nature.seMgtConstraint] OR [lModelId]=0)`	This field is irrelevant if the item is not managed with a **Unique asset tag**. It is not displayed in this case.
Asset	`RetVal = (0=[lAstId] OR [fQty] <> [Asset.fTotalQty])`	If there is not asset associated with the portfolio item or the number of units in the batch is different from the total number of units in a batch then this link is irrelevant.
Batch	`RetVal = (0=[lAstId] OR [fQty] <> [Asset.fTotalQty])`	If there is not asset associated with the portfolio item or the number of units in the batch is different from the total number of units in a batch then the link is irrelevant.
Computer	`RetVal = ("amComputer"<>[Model.Nature.OverflowTbl])`	This link is only relevant if the portfolio item is a computer.

Location	`'Relevant when in stock or assigned`
	`RetVal = 0`
	`if [seAssignment]<>0 and [seAssignment]<>1 then`
	`RetVal = 1`
	`end if`
		The link to a reservation is only relevant if the portfolio item is **In stock** or **In use**.

Phone	`RetVal = ("amPhone"<>[Model.Nature.OverflowTbl])`	This link is only relevant if the portfolio item is a telephone.

Object concerned	Script	Description

Reservation	`'Relevant when in stock or waiting to enter stock`
	`RetVal = 0`
	`if [seAssignment]<>1 and [seAssignment]<>3 then`
	`RetVal = 1`
	`end if`
		The link to a reservation is only relevant if the portfolio item is **In stock** or **Awaiting receipt**.

Slot	`RetVal = 1`
	`' Must be an asset and a device`
	`if [Model.Nature.seBasis] = 1 and [Model.Nature.bDevice] > 0 then`
	`RetVal = 0`
	`end if`
		This link to the available slots of a portfolio item is only relevant if the portfolio item is a cable device.

SoftInstall	`RetVal = ("amSoftInstall"<>[Model.Nature.OverflowTbl])`	This link is only relevant if the portfolio item is a software installation.

Stock	`'Relevant when in stock or waiting to enter stock`
	`RetVal = 0`
	`if [seAssignment]<>1 and [seAssignment]<>3 then`
	`RetVal = 1`
	`end if`
		The link to a stock is only relevant if the portfolio item is **In stock** or **Awaiting receipt**.

User	`RetVal = (amEvalScript("Irrelevant", "Stock", "")=FALSE OR [seAssignment]=2)`	The link to a user is only relevant if the portfolio item is not **In stock** or **Retired**.
WorkOrder	`RetVal = (1<>[Model.Nature.bConsumable])`	The link to a work order is only relevant if the portfolio item is a consumable.

Integrity rules

There are no integrity rules on the Portfolio Items table (**amPortfolio**).

Agents

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CAssignmentMergeAgent			None in records in the **amPortfolio** table. Depending on the operations performed by the agent, record may however be created or deleted.

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
		This agent makes sure there are no two identical portfolio items in the database. The comparison is performed on all fields except the following: ■ **fQty** ■ **lPortfolioItemId** ■ **Code** ■ **FullName** ■ **dtLastModif** ■ **mAvgPrice** ■ **bCreatedOnTheFly** On the basis of this comparison, if two identical portfolio items are found, the agents merges them into one single portfolio item and updates the quantity (**fQty**) and unit price (**mAvgPrice**). Note: The comparison also takes into account the features linked to the portfolio items. Two portfolio items that only differ in terms of a feature value are not considered to be the same.	

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
	■ Insert on object: amPortfolio ■ PostUpdate on object: AssetTag ■ PostUpdate on object: bUseQty ■ PostUpdate on object: dAssignment ■ PostUpdate on object: dtInvent ■ PostUpdate on object: Folder ■ PostUpdate on object: AvgPriceCur ■ PostUpdate on object: RMANumber ■ PostUpdate on object: seAssignment ■ PostUpdate on object: sLvl ■ PostUpdate on object: lAstId ■ PostUpdate on object: lParentId ■ PostUpdate on object: lCommentId ■ PostUpdate on object: lCostCatId ■ PostUpdate on object: lCostId ■ PostUpdate on object: lIconId ■ PostUpdate on object: lLocaId ■ PostUpdate on object: lModelId ■ PostUpdate on object: lStockId ■ PostUpdate on ob-		

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
	ject: lSupervId ■ PostUpdate on object: lUserId ■ PostUpdate on object: lWorkOrderId		
CAssignmentParentAgent	■ Insert on object: amPortfolio ■ PostUpdate on object: lParentId	This agent performs the following operations: ■ It makes sure that the parent of a portfolio item is always an asset. An error is returned if this is not the case. ■ If the portfolio item has software installations as child records, it makes sure that the nature of the model of the portfolio item is flagged **Has software installed**. An error is returned if this is not the case. ■ If a portfolio item is a consumable, it makes sure that if it is linked to an asset, the assignment is set to **Retired (or consumed)**. The assignment date is set to the current date. A consumption line is created and linked to both the portfolio item and its corresponding asset.	■ **seAssignment** ■ **dAssignment**

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CBatchQtyAgent	■ Insert on object: amPortfolio ■ PostUpdate on object: fTotalQty ■ PostUpdate on object: fQty ■ PostUpdate on object: lAstId ■ PreDelete on object: amPortfolio ■ PreDelete on object: amAsset	This agent maintains the consistency between the total quantity of a batch (**fTotalQty**) and the sum of the quantities of the batch items (**fQty**).	

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CGbAcquiDepAgent	■ Insert on object: amAsset ■ Insert on object: amPortfolio ■ PostUpdate on object: dAcquisition ■ PostUpdate on object: mPrice ■ PostUpdate on object: mTax ■ PostUpdate on object: dIntPay ■ PostUpdate on object: mIntPay ■ PostUpdate on object: mIntPayTax ■ PostUpdate on object: seAcquMethod	This agent updates the expense lines associated with the portfolio item. It functions when a portfolio item is created or the following data items are updated for an existing portfolio item: ■ **seAcquMethod** ■ **dAcquisition** ■ **mPrice** ■ **mTax** ■ **mIntPay** ■ **mIntPayTax** ■ **dIntPay** Note: The agent takes into account the distribution (split-billing) of expenses to the cost centers and cost categories. It may therefore create multiple expense lines.	None in the **amPortfolio** table.
CGbAssetAssignement	■ Insert on object: amPortfolio ■ PostDelete on object: amPortfolio ■ PreUpdate on object: lModelId	In case of creation of a portfolio item, if necessary this agent creates the corresponding record in the Assets table and the appropriate record in the overflow table matching the management constraint associated with the model of the portfolio item.	None in the **amPortfolio** table.

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CGbSousBienIntegriteAgent	■ PostUpdate on object: lLocaId ■ PostUpdate on object: lUserId ■ PostUpdate on object: lSupervId ■ PostUpdate on object: lStockId ■ PostUpdate on object: seAssignment	If one of the objects monitored for a portfolio item is updated, the agent propagates the modifications to all the child records.	None in the **amPortfolio** table.
CGbSousBienIntegriteAgent2	❖ PreUpdate on object: lParentId	If the parent record of a portfolio item is modified, then the agent propagates the values of the following fields from the new parent record to the portfolio item: ■ **lLocaId** ■ **lStockId** ■ **lSupervId** ■ **lUserId** ■ **seAssignment**	■ **lLocaId** ■ **lStockId** ■ **lSupervId** ■ **lUserId** ■ **seAssignment**

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CGbStockInOutAgent	■ PreUpdate on object: seAssignment ■ PreUpdate on object: lStockId ■ PreUpdate on object: lUserId		■ **fQty** ■ **lLocaId** ■ **lStockId** ■ **lSupervId** ■ **lUserId** ■ **seAssignment**

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
		The agent performs the following operations: ■ If the assignment of the portfolio item changes from **In stock**, the link pointing to the stock is emptied. ■ If the item has a stock and its assignment is **In use**, the agent changes it to **In stock**. ■ If the item has a stock and its assignment is set to **Return for maintenance**, **Return to supplier** or **Missing**, the agent returns an error. ■ If the item is **In stock** or **Awaiting receipt** and has no assigned stock, the agent changes the assignment to **In use**. ■ If the item is **In stock** or **Awaiting receipt** and has no assigned location, the agent sets it to that of the stock. ■ If the portfolio item does not have a location or is not **In use** and is assigned to a user, the agent sets its assignment to **In**	

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
		use and gives it the location of the user. ■ If the assignment of the asset is not **In use**, the agent empties the **User** and **Supervisor** fields. ■ If the assignment of the asset changes from **In stock** to **In use**, the agent propagates the **User** from the reservation. ■ If the assignment of the portfolio item is set to **In stock**, the agent deletes any remaining reservations associated with the portfolio item.	
COverflow-ChangeAgent	❖ PreUpdate on object: lModelId	This agent stops the model of a portfolio item from being changed if doing so implies the associated overflow table being changed also.	

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CRedundancyAgent	■ Insert on object: amComputer ■ Insert on object: amPortfolio ■ PostUpdate on object: lItemId ■ PostUpdate on object: AssetTag ■ PostUpdate on object: AssetTag	This agent makes sure that the **AssetTag** fields of a computer and its associated portfolio item are identical: ■ If the **AssetTag** field of a record in the **amComputer** table is modified, the agent propagates this change to the **AssetTag** field of the record in the corresponding **amPortfolio** table. ■ If the **AssetTag** field of a record in the **amPortfolio** table is modified, the agent propagates this change to the **AssetTag** field of the record in the corresponding **amComputer** table.	◆ **AssetTag**

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CRedundancyAgent	■ Insert on object: amAsset ■ Insert on object: amPortfolio ■ PostUpdate on object: lModelId ■ PostUpdate on object: lModelId ■ PostUpdate on object: lAstId	This agent makes sure that an asset and its associated portfolio item always point to the same model: ■ If the **lModelId** link of a record in the **amAsset** table is modified, the agent propagates this change to the **lModelId** field of the record in the corresponding **amPortfolio** table. ■ If the **lModelId** link of a record in the **amPortfolio** table is modified, the agent propagates this change to the **lModelId** link of the record in the corresponding **amAsset** table.	◆ **lModelId**

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CRedundancyAgent	■ Insert on object: amAsset ■ Insert on object: amPortfolio ■ PostUpdate on object: AssetTag ■ PostUpdate on object: AssetTag ■ PostUpdate on object: lAstId	This agent makes sure that the **AssetTag** fields of an asset and its associated portfolio item are identical: ■ If the **AssetTag** field of a record in the **amAsset** table is modified, the agent propagates this change to the **AssetTag** field of the record in the corresponding **amPortfolio** table. ■ If the **AssetTag** field of a record in the **amPortfolio** table is modified, the agent propagates this change to the **AssetTag** field of the record in the corresponding **amAsset** table.	◆ **AssetTag**
CReturnAssignmentAgent	◆ PostUpdate on object: seAssignment	When the assignment of a portfolio item (which is not a consumable) is set to **Return to supplier** or **Retired (or consumed)**, this item is de-hierarchized.	

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
FullName agent	■ Insert on object: amPortfolio ■ PostUpdate on object: Code ■ PostUpdate on object: lParentId ■ PreUpdate on object: Code ■ PreUpdate on object: lParentId	This agent manages tree structures in hierarchic tables. In the Portfolio items table, it maintains hierarchical integrity. The full name of the portfolio item and its hierarchical level are recalculated if: ■ the portfolio item's code is changed ■ its parent is modified	■ FullName ■ sLvl

Workflows

The following tables summarize the workflows dealing with the Assets table (**amAsset**).

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Determining the workflows used for a table [page 183] at the end of this document.

Workflow reference	Workflow type	Description
BST_SAM04	Synchronous	When a portfolio item is retired (its assignment changes to **Retired**), this workflow asks the administrator if the licenses that were linked to this item can be freed up or not. A wizard is available to help in selecting the licenses to be freed.

19 CHAPTER Projects table (amProject)

This chapter provides an exhaustive list of all the mechanisms dealing with the Projects table. Each section deals with a different type of automatic mechanism.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 19.1. Validity scripts on the table

Script	Description
`If Not IsEmpty([dEnd]) and Not IsEmpty([dStart]) and [dStart] > [dEnd] Then Err.Raise(-2009, "The end date (dEnd) must be greate r than or equal to the sta rt date (dStart).") RetVal = FALSE Else RetVal = TRUE End If`	If the start and end dates of the project are not empty and the end date comes before the start date, the record is rejected.

Table 19.2. Default value scripts

Object concerned	Script	Description
Code	`RetVal = "C" + AmCounter(" amProject_Code", 6)`	By default, the unique code of a project is the concatentation of the letter **C** and the value of the **amProject_Code** counter on 6 figures.
dStart	`RetVal = AmDate()`	By default, the start date of the project is the date of creation of the record.

Agents

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
CDateAlarmAgent	◆ Post-Update on the **dEnd** object	This agent recalculates the alarms associated with the project end date.	None in the **amProject** table.

20 CHAPTER Stocks table (amStock)

This chapter provides an exhaustive list of all the mechanisms dealing with the Stocks table. Each section deals with a different type of automatic mechanism.

Note:

There are no automatic mechanisms other than the default script values on this table.

Scripts

The following tables summarize the objects to which the scripts are attached and describe the operations performed by the scripts.

Warning:

This section lists all the standard scripts touching upon the objects in the table concerned. This list cannot include any customizations and modifications specific to your own implementation of AssetCenter. To learn how to extract the scripts really used in your implementation concerning this table, refer to the appendix Extracting all the scripts from a database [page 173] at the end of this document.

Table 20.1. Default value scripts

Object concerned	Script	Description
Code	`RetVal = "C" + AmCounter("amStock_Code", 6)`	By default, the unique code of the stock is the concatentation of the letter **C** and the value of the **amStock_Code** counter on 6 figures.
DeliveryAddr	`RetVal = [Location.Address1] + " " + [Location.Address2] + " " + [Location.ZIP] + " " + [Location.City] + " " + [Location.State] + " " + [Location.Country.Name]`	By default, the delivery address for the stock is the concatentation of the address, postal code, city, state and country of the location associated with the stock.
dtValueCv	`RetVal = AmDate()`	By default, the conversion date for the stock value is the stock creation date.
ValueCur	`RetVal = AmDefaultCurrency()`	By default, the currency used to express the value of the stock is the default currency.

21 CHAPTER Third-Party Companies table (amThirdParty)

This chapter provides an exhaustive list of all the mechanisms dealing with the Third-Party Companies table. Each section deals with a different type of automatic mechanism.

Note:

There are no automatic mechnisms other than the agents on this table.

Integrity rules

There are no integrity rules on the Third-Party Companies table (**amThirdParty**).

Agents

SQL name of the agent	List of monitored objects	Operations performed	List of any modified objects
lContactId_lCpyId	■ SyncRead on object: lContactId ■ SyncRead on object: lCpyId		■ lContactId ■ lCpyId

22 CHAPTER Glossary

Database terms

Stored procedure

Stored procedures enable you to move the burden of certain processes to the database engine rather than issue SQL statements from the client application. In practice, a stored procedure is unit of processing that receives parameters, executes operations and returns a result. They are written in a procedural language that includes SQL and are saved at the database server level.

Transaction

A transaction may be defined as a series of operations that are performed in full, or not at all, but never in part. If one of the operations fails then all the operations are cancelled. For example, if you wish to transfer a person from the Departments and Employees table to the Contracts table, the person must be first inserted into the Contracts table then deleted from the Employees table. It cannot be allowed for the second operation to be neglected otherwise the database would become inconsistent. From a practical point of view, a transaction is initiated by

a SQL statement and any modifications made to the database or only visible inside the transaction. They only become effective once the transaction has been validated by SQL operation called a **Commit**. If an anomaly occurs, all modifications can be cancelled by finishing the transaction with a **Rollback** command.

A transaction has the four following properties, which are universally recognized the domain of database engines:

1. Atomicity: A transaction is a unit of processing that so called atomic. Either it is performed in full, or not at all.
2. Integrity: A transaction changes the database from one consistent state to another. During the time of the transaction, the database remains unchanged.
3. Isolation: Modifications made as part of a transaction are invisible (in particular, to other transactions) for so long as they are not committed.
4. Permanence: After the **Commit** is reached in a transaction, the modifications are permanent and cannot be cancelled.

Trigger

A trigger associates a process with a specific action on the database. When the action is performed and the data matches a certain condition, the process is executed automatically by the database server. A systematic process is generally linked with an integrity constraint.

A trigger is specific type of stored procedure.

Exclusive lock

An exclusive lock is held by a transaction in order to exclude any other manipulations of the object or data locked.

APPENDIX A

Extracting all the scripts from a database

This appendix aims to help you extract all the scripts included in your AssetCenter implementation.

AssetCenter Database Administrator, shipped with AssetCenter provides a template-based method to extract information (**`.tpl`** extension files).

Among the standard templates provided with AssetCenter, one of them, the **`dbdict.tpl`** file, enables you to export all the customization information from your database (including information on features, calculated featues, configuration scripts, etc.) to a standard text-formatted file. Used along with a source control tool, this description file can be very useful for keeping a trace of all customization modifications made to the database.

This appendix includes a simplified template that just extracts information related to the script. You can copy the contents to a file with the **`.tpl`** extension and execute it in AssetCenter Database Administrator.

Note:

For further information on templates, refer to the **Administration** guide, chapter **Standard database description files**.

Executing a template in AssetCenter Database Administrator

To execute a template in AssetCenter Database Administrator, use the following procedure:

1. Start AssetCenter Database Administrator if it is not already running and connect to your database,
2. Select **Action/ Templates/ Select folder** and select the folder containing the template or templates you wish to execute,
3. Select **Action/ Templates/ Refresh list**. The list of available templates is displayed in the second section of the **Action/ Templates** menu.
4. Execute the script of you choice by selecting **Action/ Templates**, and then the name of the script.

Examples of templates

The two following templates extract the information related to scripts. The first template saves the information in the form of an XML file (one XML file per table) using the DocBook format, the second in classic HTML format (one HTML file per table).

XML version

```
$ Desc: Scripts catalog XML - English - AssetCenter/InfraCenter
$ Type: XML
$ (c) Peregrine Systems, Inc. Documentation Team - 2004
$ Maintainer: Stéphane Bline
$ Warning: Do not modify this file directly. Send a formal change request
to me.
$OutputDir = $(Output.Path)
$MkDir($(OutputDir) + "tables")
$for Tables sort (SqlName ASC)
$SetOutput($(OutputDir) + "\tables\" + $(SqlName) + ".xml")
$TableSQLName=$(SqlName)
$ Output for the tables
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE sect1 PUBLIC "-//Norman Walsh//DTD DocBk XML V3.1.7//EN" "docboo
```

```
kx.dtd">
<sect1 lang="en" id="$(SqlName)"><title id="$(SqlName).Title">Scripts on t
able $(SqlName) ($(Label))</title>

$if ($(IsValidScript.CalcMode) = 2)
<sect2 id="SB-190919"><title id="SB-190920">Validity script on table $(Sql
Name)</title>
<programlisting id="SB-190921">$ReplaceChars($ReplaceChars($ScriptFormat($
(IsValidScript.Source),4),"&", "&"), "<", "<")</programlisting id="SB-1909
22">
</sect2>
$endif

<sect2 id="SB-190923"><title id="SB-190924">Scripts on fields</title>
$TableSQLName=$(SqlName)
$TableLabel=$(Label)
$for Fields sort (SqlName ASC)
$if  ($(ReadOnlyScript.CalcMode) = 2) or ($(HistoryScript.CalcMode) = 2) o
r
($(MandatoryScript.CalcMode) = 2) or ($(DefaultScript.Source)< id="SB-1909
25">"") or ($(RelevantScript.CalcMode)
=
2)
<sect3 lang="en" id="$(TableSQLName).$(SqlName)"><title id="$(TableSQLName
).$(SqlName).Title">Field $(SqlName) ($(Label))</title>
<informaltable id="SB-190926">
<tgroup cols="2" id="SB-190927">
<colspec colnum="1" colname="col1" colwidth="1*"/>
<colspec colnum="2" colname="col2" colwidth="1*"/>
<thead id="SB-190928">
<row id="SB-190929">
<entry colname="col1" align="center" id="SB-190930"><emphasis>Property</em
phasis></entry>
<entry colname="col2" align="center" id="SB-190931"><emphasis>Value</empha
sis></entry>
</row>
</thead>
<tbody id="SB-190932">
<row id="SB-190933">
<entry colname="col1" id="SB-190934"><emphasis>SQL name</emphasis></entry>
<entry colname="col2" align="center" id="SB-190935">$(SqlName)</entry>
</row>
<row id="SB-190936"><entry colname="col1" id="SB-190937"><emphasis>Name</e
mphasis></entry>
<entry colname="col2" align="center" id="SB-190938">$(Label)</entry>
</row>
$if ($(ReadOnlyScript.Source)< id="SB-190939">"")
<row id="SB-190940"><entry colname="col1" id="SB-190941"><emphasis>Read-on
ly script</emphasis></entry>
<entry colname="col2" align="left" id="SB-190942"><programlisting id="SB-1
90943">$ReplaceChars($ReplaceChars($(ReadOnlyScript.Source),"&", "&"), "<"
, "<")</programlisting id="SB-190944"></entry>
</row>
$endif
$if ($(HistoryScript.Source)< id="SB-190945">"")
<row id="SB-190946"><entry colname="col1" id="SB-190947"><emphasis>History
```

```
script</emphasis></entry>
<entry colname="col2" align="left" id="SB-190948"><programlisting id="SB-1
90949">$ReplaceChars($ReplaceChars($(HistoryScript.Source),"&", "&"), "<",
"<")</programlisting id="SB-190950"></entry>
</row>
$endif
$if ($(MandatoryScript.Source)< id="SB-190951">"")
<row id="SB-190952"><entry colname="col1" id="SB-190953"><emphasis>Mandato
ry script</emphasis></entry>
<entry colname="col2" align="left" id="SB-190954"><programlisting id="SB-1
90955">$ReplaceChars($ReplaceChars($(MandatoryScript.Source),"&", "&"), "<
", "<")</programlisting id="SB-190956"></entry>
</row>
$endif
$if ($(DefaultScript.Source)< id="SB-190957">"")
<row id="SB-190958"><entry colname="col1" id="SB-190959"><emphasis>Default
value script</emphasis></entry>
<entry colname="col2" align="left" id="SB-190960"><programlisting id="SB-1
90961">$ReplaceChars($ReplaceChars($(DefaultScript.Source),"&", "&"), "<",
"<")</programlisting id="SB-190962"></entry>
</row>
$endif
$if ($(RelevantScript.Source)< id="SB-190963">"")
<row id="SB-190964"><entry colname="col1" id="SB-190965"><emphasis>Relevan
ce script</emphasis></entry>
<entry colname="col2" align="left" id="SB-190966"><programlisting id="SB-1
90967">$ReplaceChars($ReplaceChars($(RelevantScript.Source),"&", "&"), "<"
, "<")</programlisting id="SB-190968"></entry>
</row>
$endif
</tbody>
</tgroup>
</informaltable>
</sect3>
$endif
$endfor
</sect2>

<sect2 id="SB-190969"><title id="SB-190970">Scripts on links</title>
$TableSQLName=$(SqlName)
$TableLabel=$(Label)
$for Links sort (SqlName ASC)
$if  ($(RelevantScript.CalcMode) = 2)
<sect3 lang="en" id="$(TableSQLName).$(SqlName)"><title id="$(TableSQLName
).$(SqlName).Title">Link $(SqlName) ($(Label))</title>
<informaltable id="SB-190971">
<tgroup cols="2" id="SB-190972">
<colspec colnum="1" colname="col1" colwidth="1*"/>
<colspec colnum="2" colname="col2" colwidth="1*"/>
<thead id="SB-190973">
<row id="SB-190974">
<entry colname="col1" align="center" id="SB-190975"><emphasis>Property</em
phasis></entry>
<entry colname="col2" align="center" id="SB-190976"><emphasis>Value</empha
sis></entry>
</row>
```

```
</thead>
<tbody id="SB-190977">
<row id="SB-190978">
<entry colname="col1" id="SB-190979"><emphasis>SQL name</emphasis></entry>
<entry colname="col2" align="center" id="SB-190980">$(SqlName)</entry>
</row>
<row id="SB-190981"><entry colname="col1" id="SB-190982"><emphasis>Name</e
mphasis></entry>
<entry colname="col2" align="center" id="SB-190983">$(Label)</entry>
</row>
$if ($(RelevantScript.Source)< id="SB-190984">"")
<row id="SB-190985"><entry colname="col1" id="SB-190986"><emphasis>Relevan
ce script</emphasis></entry>
<entry colname="col2" align="left" id="SB-190987"><programlisting id="SB-1
90988">$ReplaceChars($ReplaceChars($(RelevantScript.Source),"&", "&"), "<"
, "<")</programlisting id="SB-190989"></entry>
</row>
$endif
</tbody>
</tgroup>
</informaltable>
</sect3>
$endif
$endfor
</sect2>
</sect1>
$endfor

$script
'-----------------------------------------------------------------------
' Format a script to put it in a cfg
'-----------------------------------------------------------------------
Function ScriptFormat(strMemos as String, iSpace as Integer) as String
ScriptFormat = ReplaceChars(strMemos, Chr(10), Chr(10) + Space(iSpace))
End Function

'-----------------------------------------------------------------------
' Replaces a string with another one
'-----------------------------------------------------------------------
Function ReplaceChars(strMemos as String, strToRep as String, strReplaceme
nt as String) as String
Dim I as Integer
ReplaceChars = strMemos
I = InStr(0, ReplaceChars, strToRep)
While (I < id="SB-190990"> 0)
ReplaceChars = Left(ReplaceChars, I - 1) + strReplacement + Mid(ReplaceCha
rs, I +
Len(strToRep), Len(ReplaceChars))
I = InStr(I + Len(strToRep), ReplaceChars, strToRep)
Wend
End Function
$endscript
```

HTML version

```
$ Desc: Scripts catalog HTML - English - AssetCenter/InfraCenter
$ Type: HTML
$ (c) Peregrine Systems, Inc. Documentation Team - 2004
$ Maintainer: Stéphane Bline
$ Warning: Do not modify this file directly. Send a formal change request
to me.
$OutputDir = $(Output.Path)
$MkDir($(OutputDir) + "tables")
$for Tables sort (SqlName ASC)
$SetOutput($(OutputDir) + "\tables\" + $(SqlName) + ".htm")
$TableSQLName=$(SqlName)
$ Output for the tables
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head id="SB-190994">
<title id="SB-190995">Scripts on table $(SqlName) ($(Label))</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" id
="SB-190996">
</head>
<body id="SB-190997">

$if ($(IsValidScript.CalcMode) = 2)
<h1 style="FONT-WEIGHT: bold; FONT-SIZE: 18pt; COLOR: #000066; FONT-FAMILY
: Verdana" id="SB-190998">Validity script on table $(SqlName)</h1>
<p style="font-family : Courier New; text-align : left; border : thin groo
ve" id="SB-190999">$ReplaceChars($ReplaceChars($ScriptFormat($(IsValidScri
pt.Source),4),"&", "&"), "<", "<")</p id="SB-191000">
$endif

<h1 style="FONT-WEIGHT: bold; FONT-SIZE: 18pt; COLOR: #000066; FONT-FAMILY
: Verdana" id="SB-191001">Scripts on fields</h1>
$TableSQLName=$(SqlName)
$TableLabel=$(Label)
$for Fields sort (SqlName ASC)
$if  ($(ReadOnlyScript.CalcMode) = 2) or ($(HistoryScript.CalcMode) = 2) o
r ($(MandatoryScript.CalcMode) = 2) or ($(DefaultScript.Source)< id="SB-19
1002">"") or ($(RelevantScript.CalcMode) = 2)
<h2 style="FONT-WEIGHT: bold; FONT-SIZE: 10pt; COLOR: #000066; FONT-FAMILY
: Verdana; align: left" id="SB-191003">Field $(SqlName) ($(Label))</h2>
<table style="BORDER-RIGHT: #000066 1px solid; BORDER-TOP: #000066 1px sol
id; MARGIN-BOTTOM: 10px; BORDER-LEFT: #000066 1px solid; WIDTH: 400px; BOR
DER-BOTTOM: #000066 1px solid; table-width: 400px" id="SB-191004">
<tr style="PADDING-RIGHT: 2px; PADDING-LEFT: 2px; FONT-WEIGHT: bold; FONT-
SIZE: 8pt; PADDING-BOTTOM: 2px; COLOR: #ffffff; PADDING-TOP: 2px; FONT-FAM
ILY: Verdana, Helvetica, sans-serif; BACKGROUND-COLOR: #000066" id="SB-191
005">
<th id="SB-191006">Property</th>
<th id="SB-191007">Value</emphasis></th>
</tr>
<tr style="PADDING-RIGHT: 2px; PADDING-LEFT: 2px; FONT-SIZE: 8pt; PADDING-
BOTTOM: 2px; VERTICAL-ALIGN: top; COLOR: #000066; PADDING-TOP: 2px; FONT-F
AMILY: Verdana, Tahoma, Helvetica, sans-serif; BACKGROUND-COLOR: #ffffff"
```

```
id="SB-191008">
<td id="SB-191009">SQL name</td>
<td id="SB-191010">$(SqlName)</td>
</tr>
<tr style="PADDING-RIGHT: 2px; PADDING-LEFT: 2px; FONT-SIZE: 8pt; PADDING-
BOTTOM: 2px; VERTICAL-ALIGN: top; COLOR: #000066; PADDING-TOP: 2px; FONT-F
AMILY: Verdana, Tahoma, Helvetica, sans-serif; BACKGROUND-COLOR: #ffffff"
id="SB-191011">
<td id="SB-191012">Name</td>
<td id="SB-191013">$(Label)</td>
</tr>
$if ($(ReadOnlyScript.Source)< id="SB-191014">"")
<tr style="PADDING-RIGHT: 2px; PADDING-LEFT: 2px; FONT-SIZE: 8pt; PADDING-
BOTTOM: 2px; VERTICAL-ALIGN: top; COLOR: #000066; PADDING-TOP: 2px; FONT-F
AMILY: Verdana, Tahoma, Helvetica, sans-serif; BACKGROUND-COLOR: #ffffff"
id="SB-191015">
<td id="SB-191016">Read-only script</td>
<td id="SB-191017"><p style="font-family : Courier New; text-align : left;
border : thin groove" id="SB-191018">$ReplaceChars($ReplaceChars($(ReadOnl
yScript.Source),"&", "&"), "<", "<")</p id="SB-191019"></td>
</tr>
$endif
$if ($(HistoryScript.Source)< id="SB-191020">"")
<tr style="PADDING-RIGHT: 2px; PADDING-LEFT: 2px; FONT-SIZE: 8pt; PADDING-
BOTTOM: 2px; VERTICAL-ALIGN: top; COLOR: #000066; PADDING-TOP: 2px; FONT-F
AMILY: Verdana, Tahoma, Helvetica, sans-serif; BACKGROUND-COLOR: #ffffff"
id="SB-191021">
<td id="SB-191022">History script</td>
<td id="SB-191023"><p style="font-family : Courier New; text-align : left;
border : thin groove" id="SB-191024">$ReplaceChars($ReplaceChars($(History
Script.Source),"&", "&"), "<", "<")</p id="SB-191025"></td>
</tr>
$endif
$if ($(MandatoryScript.Source)< id="SB-191026">"")
<tr style="PADDING-RIGHT: 2px; PADDING-LEFT: 2px; FONT-SIZE: 8pt; PADDING-
BOTTOM: 2px; VERTICAL-ALIGN: top; COLOR: #000066; PADDING-TOP: 2px; FONT-F
AMILY: Verdana, Tahoma, Helvetica, sans-serif; BACKGROUND-COLOR: #ffffff"
id="SB-191027">
<td id="SB-191028">Mandatory script</td>
<td id="SB-191029"><p style="font-family : Courier New; text-align : left;
border : thin groove" id="SB-191030">$ReplaceChars($ReplaceChars($(Mandato
ryScript.Source),"&", "&"), "<", "<")</p id="SB-191031"></td>
</tr>
$endif
$if ($(DefaultScript.Source)< id="SB-191032">"")
<tr style="PADDING-RIGHT: 2px; PADDING-LEFT: 2px; FONT-SIZE: 8pt; PADDING-
BOTTOM: 2px; VERTICAL-ALIGN: top; COLOR: #000066; PADDING-TOP: 2px; FONT-F
AMILY: Verdana, Tahoma, Helvetica, sans-serif; BACKGROUND-COLOR: #ffffff"
id="SB-191033">
<td id="SB-191034">Default value script</td>
<td id="SB-191035"><p style="font-family : Courier New; text-align : left;
border : thin groove" id="SB-191036">$ReplaceChars($ReplaceChars($(Default
Script.Source),"&", "&"), "<", "<")</p id="SB-191037"></td>
</tr>
$endif
$if ($(RelevantScript.Source)< id="SB-191038">"")
```

```
<tr style="PADDING-RIGHT: 2px; PADDING-LEFT: 2px; FONT-SIZE: 8pt; PADDING-
BOTTOM: 2px; VERTICAL-ALIGN: top; COLOR: #000066; PADDING-TOP: 2px; FONT-F
AMILY: Verdana, Tahoma, Helvetica, sans-serif; BACKGROUND-COLOR: #ffffff"
id="SB-191039">
<td id="SB-191040">Relevance script</td>
<td id="SB-191041"><p style="font-family : Courier New; text-align : left;
border : thin groove" id="SB-191042">$ReplaceChars($ReplaceChars($(Relevan
tScript.Source),"&", "&"), "<", "<")</p id="SB-191043"></td>
</tr>
$endif
</table>
$endif
$endfor

<h1 style="FONT-WEIGHT: bold; FONT-SIZE: 18pt; COLOR: #000066; FONT-FAMILY
: Verdana" id="SB-191044">Scripts on links</h1>
$TableSQLName=$(SqlName)
$TableLabel=$(Label)
$for Links sort (SqlName ASC)
$if  ($(RelevantScript.CalcMode) = 2)
<h2 style="FONT-WEIGHT: bold; FONT-SIZE: 10pt; COLOR: #000066; FONT-FAMILY
: Verdana" id="SB-191045">Link $(SqlName) ($(Label))</h2>
<table style="BORDER-RIGHT: #000066 1px solid; BORDER-TOP: #000066 1px sol
id; MARGIN-BOTTOM: 10px; BORDER-LEFT: #000066 1px solid; WIDTH: 400px; BOR
DER-BOTTOM: #000066 1px solid; table-width: 400px" id="SB-191046">
<tr style="PADDING-RIGHT: 2px; PADDING-LEFT: 2px; FONT-WEIGHT: bold; FONT-
SIZE: 8pt; PADDING-BOTTOM: 2px; COLOR: #ffffff; PADDING-TOP: 2px; FONT-FAM
ILY: Verdana, Helvetica, sans-serif; BACKGROUND-COLOR: #000066" id="SB-191
047">
<th id="SB-191048">Property</th>
<th id="SB-191049">Value</th>
</tr>

<tr style="PADDING-RIGHT: 2px; PADDING-LEFT: 2px; FONT-SIZE: 8pt; PADDING-
BOTTOM: 2px; VERTICAL-ALIGN: top; COLOR: #000066; PADDING-TOP: 2px; FONT-F
AMILY: Verdana, Tahoma, Helvetica, sans-serif; BACKGROUND-COLOR: #ffffff"
id="SB-191050">
<td id="SB-191051">SQL name</td>
<td id="SB-191052">$(SqlName)</td>
</tr>
<tr style="PADDING-RIGHT: 2px; PADDING-LEFT: 2px; FONT-SIZE: 8pt; PADDING-
BOTTOM: 2px; VERTICAL-ALIGN: top; COLOR: #000066; PADDING-TOP: 2px; FONT-F
AMILY: Verdana, Tahoma, Helvetica, sans-serif; BACKGROUND-COLOR: #ffffff"
id="SB-191053">
<td id="SB-191054">Name</td>
<td id="SB-191055">$(Label)</td>
</tr>
$if ($(RelevantScript.Source)< id="SB-191056">"")
<tr style="PADDING-RIGHT: 2px; PADDING-LEFT: 2px; FONT-SIZE: 8pt; PADDING-
BOTTOM: 2px; VERTICAL-ALIGN: top; COLOR: #000066; PADDING-TOP: 2px; FONT-F
AMILY: Verdana, Tahoma, Helvetica, sans-serif; BACKGROUND-COLOR: #ffffff"
id="SB-191057">
<td id="SB-191058">Relevance script</td>
<td id="SB-191059"><p style="font-family : Courier New; text-align : left;
border : thin groove" id="SB-191060">$ReplaceChars($ReplaceChars($(Relevan
tScript.Source),"&", "&"), "<", "<")</p id="SB-191061"></td>
```

```
</tr>
$endif
</table>
$endif
$endfor
$endfor

$script
'-----------------------------------------------------------------
' Format a script to put it in a cfg
'-----------------------------------------------------------------
Function ScriptFormat(strMemos as String, iSpace as Integer) as String
ScriptFormat = ReplaceChars(strMemos, Chr(10), Chr(10) + Space(iSpace))
End Function

'-----------------------------------------------------------------
' Replaces a string with another one
'-----------------------------------------------------------------
Function ReplaceChars(strMemos as String, strToRep as String, strReplaceme
nt as String) as String
Dim I as Integer
ReplaceChars = strMemos
I = InStr(0, ReplaceChars, strToRep)
While (I < id="SB-191062"> 0)
ReplaceChars = Left(ReplaceChars, I - 1) + strReplacement + Mid(ReplaceCha
rs, I + Len(strToRep), Len(ReplaceChars))
I = InStr(I + Len(strToRep), ReplaceChars, strToRep)
Wend
End Function
$endscript
```

APPENDIX B Determining the workflows used for a table

This appendix aims to help you determine which workflows concern a given table in your AssetCenter implementation.

Workflows have a general context, also called the context of the start object. It is the table which is monitored for an event. The event can be a record inserted/deleted or a field updated, etc.

This context can change as the workflow progresses. Thus, each workflow activity can have its own context, different from the start context.

When searching for workflows operating on a given table, we can thus take the two following cases into account:

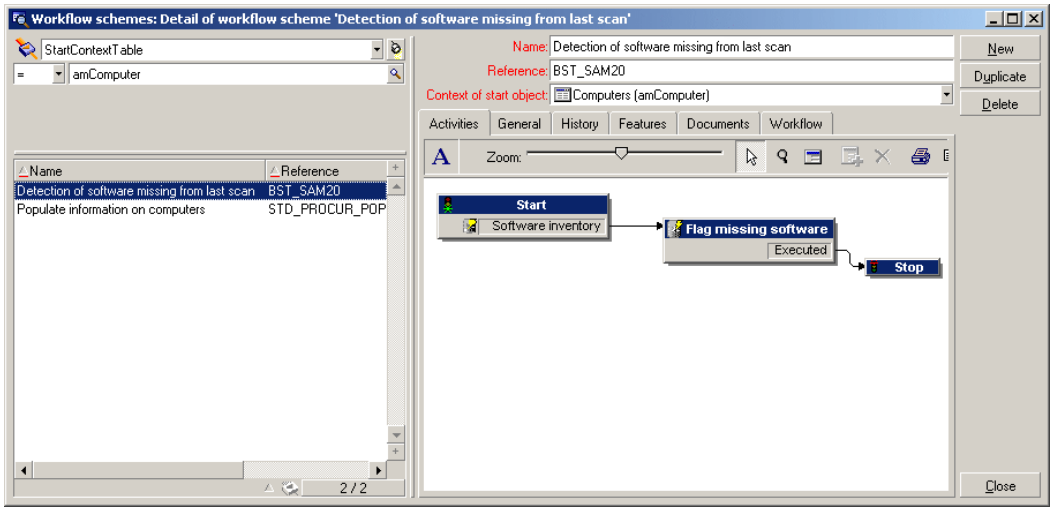
- The workflows whose start context is the table in question,
- The workflows with activities whose context is table in question.

In the following example, we are going list all workflows concerning the Computers table (**amComputer**).

First, look for the workflows whose start context is the Computers table. To do this:

1. Start AssetCenter if it not already running and then select **Tools/ Workflows/ Workflow schemes**.
2. Create a simple filter as shown below. Only those workflows whose start context is the **amComputer** table are displayed in the list. The list of workflows is as follows:
   - Detection of software missing from last scan

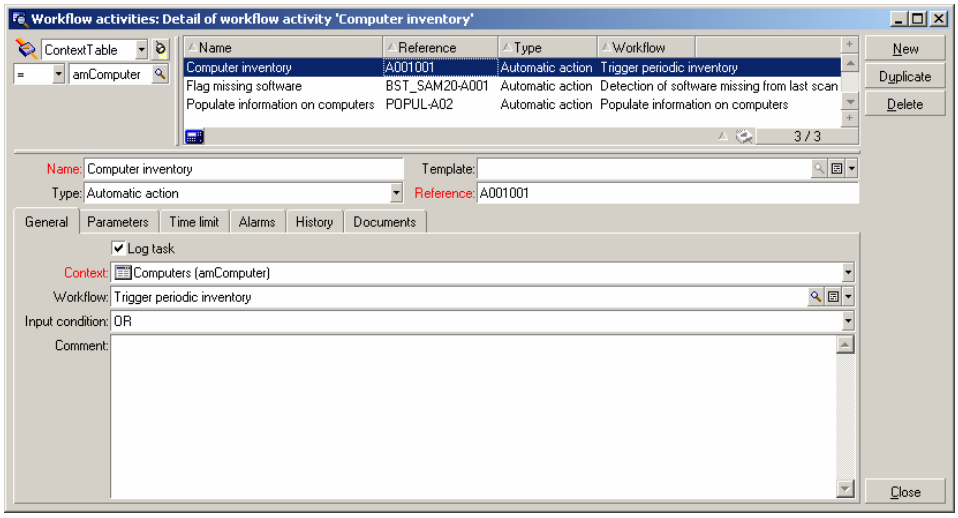
- Populate information on computers



Let's now look for workflows with one or more activities whose context is the Computers table. To do this:

1. Start AssetCenter if it not already running and then select **Administration/ List of screens**.
2. Select the **Workflow activities (sysamWfActivity)** screen from the list. AssetCenter displays the list of all the workflow activities.
3. Create a simple filter as shown below. Only those activities whose context is the **amComputer** table are displayed in the list with their associated workflow names. These are the following workflows:
   - Detection of software missing from last scan
   - Populate information on computers

- Trigger periodic inventory



Note:

The two workflows found earlier are of course included in this list since they have an activity (start activity) whose context matches our filter.

APPENDIX C

Extracting the list of fields and links of the screens

This appendix aims to help you extract the list of fields and links of a screen of a given table.

AssetCenter Database Administrator, shipped with AssetCenter provides a template-based method to extract information (**.tpl** extension files).

This appendix includes a simplified and commented template that just extracts the list of the objects of the screens defined for the tables. This resulting list uses the pipe character "|" as a separator; You can change this by modifying the template. You can copy the contents to a file with the **.tpl** extension and execute it in AssetCenter Database Administrator.

Note:

For further information on templates, refer to the **Administration** guide, chapter **Standard database description files**.

Executing a template in AssetCenter Database Administrator

To execute a template in AssetCenter Database Administrator, use the following procedure:

1. Start AssetCenter Database Administrator if it is not already running and connect to your database,
2. Select **Action/ Templates/ Select folder** and select the folder containing the template or templates you wish to execute,
3. Select **Action/ Templates/ Refresh list**. The list of available templates is displayed in the second section of the **Action/ Templates** menu.
4. Execute the script of you choice by selecting **Action/ Templates**, and then the name of the script.

Template example

```
$ Desc: Helper template (Tables - Screen - Fields)
$ Type: TXT
$ (c) Peregrine Systems, Inc. Documentation Team - 2004
$ Maintainer: Stéphane Bline
$ Warning: Do not modify this file directly. Send a formal change request
to me.

$ Specify the output folder for the list. A folder named fieldlist is crea
ted to store the result of the template execution
$OutputDir = $(Output.Path)
$MkDir($(OutputDir) + "fieldlist")
$ The output will be dumped to a text file name fields.txt
$SetOutput($(OutputDir) + "\fieldlist\fields.txt")
$ A first line containing the column titles is created
Table|Table Label|Field|Field Label|Screen|Screen Name|Tab|Tab Label
$ The template iterates on the screens defined within the database. For ea
ch one, the screen SQL name is retrieved
$for Screens sort (SqlName ASC)
$ScreensSQLName=$(SqlName)
$ The SQL Name and the label of the table attached to this screen is also
retrieved
$TableSQLName=$(Table.SqlName)
$TableLabel=$(Table.Label)
$ Now that the context is the screen, the script iterates on the tabs cont
ained in the screen and retrieves the tab SQL Name and label
```

```
$for Pages sort (SqlName ASC)
$PageSQLName=$(SqlName)
$PageLabel=$(Label)
$ If tab label is empty, then we are not inside a tab and the tab label an
d SQL names are not meaningful anymore
$if ($(PageLabel)="")
$PageLabel="N/A"
$PageSQLName="N/A"
$endif
$ Now that the context is the tab, the script iterates on the elements con
tained in this tab (fields, links, ...)
$ The script also retrieves the SQL Name and label of the object
$for Fields sort (SqlName ASC)
$FieldSQLName=$(SqlName)
$FieldLabel=$(Label)
$ For the sake of the example we are going to limit the output to a list o
f fields and links.
$ If the Islink or Isfield conditional block below is removed then ALL obj
ects will be retrieved (features, screen geometry, calculated fields,...)
$if $(IsLink) or $(IsField)
$ A line containing all the information is sent to the output file
$(TableSQLName)|$(TableLabel)|$(FieldSQLName)|$(FieldLabel)|$(ScreensSQLNa
me)|$(PageSQLName)|$(PageLabel)
$endif
$endfor
$endfor
$endfor

$script
```

PEREGRINE

Index


Peregrine
SYSTEMS