

Visual Modeler

Best Practices Guide

Release 9.1



Copyright

This edition applies to the 9.1 Version of Visual Modeler and to all subsequent releases and modifications until otherwise indicated in new editions.

Before using this information and the product it supports, read the information in *Notices* on page 37.

Licensed Materials - Property of IBM

Visual Modeler

© Copyright IBM Corp. 1999, 2011. All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP

Schedule Contract with IBM Corp.

Contents

Introduction	5
Backup and Recovery Best Practices	6
Deployment Architecture Overview	6
Infrastructure	7
Backup Strategies	8
Database Recovery	8
Application Server and Web Server Recovery	8
Item Modeling	9
Absolute and Relative Paths	9
Planning Considerations	9
Models	10
Make Models as Small and Simple as Possible	10
Use popup-qty Controls for Entering Quantity	12
Properties	14
Use Meaningful Property Names	14
Do Not Use the Same Property to Mean Two Different Things	15
Define Properties at the Appropriate Level in the Model Hierarchy	16
Using Multiple Properties with the Same Value	16
Use Worksheets to Simplify Property Assignment	17
Avoid Chaining Property Formulas	18
Rules	18
Rule Firing Conditions	19
Order Rule Fragments So That Rules Fire Only When Necessary	19
Create General-Purpose Rules	19
Use Formulas Where Appropriate	19
Avoid Specifying Paths to Instances of Items or Properties	20
Constraint Tables vs. Rules	20
Modular Development	21
Tools	21
Using the Trace Log	21
Using the Model Reporting Tool	23
Using Load Testing Tools	23
Using the Model Reporting Tool	25
Using Load Testing Tools	25
Cache Status	26
Performance	26
Rules	26
Properties	26
Archiving Data	27

Updating Statistics	28
Updating Statistics For an Oracle Database	28
Updating Statistics For a SQL Server Database	28
JVM Memory and Tuning Guidelines	29
Adjusting JVM Memory Settings	29
Additional Performance Tuning	30
Tracing Garbage Collection Activities	30
Log Analyzer Tool	32
Setting Up Log Analyzer Daily Reports	33
Daily Reports Workflow	33
Setting Up the Daily Reports	34
Recommended directory layout	34
Configuration	35
Notices	37
<hr/>	
Trademarks	39
Index	41
<hr/>	

Introduction

This guide provides a description of the best practices that have been identified for implementing the Visual Modeler.

Backup and Recovery Best Practices

The best recovery plans focus on prevention. By setting up a robust environment, putting redundant systems in place, establishing regular backup and restore policies, and conducting regular tests that you can recover from backup, you can limit the effects of disasters by providing safeguards for each layer of your deployment infrastructure.

Some decisions about backup and restore policies must be made based upon business criteria. What is the value of the site being available per day for your business? How much data can you afford to lose? How long can your e-commerce Web site be unavailable to customers? What are the time and cost trade-offs for various backup and restore solutions? Answers to these questions will help to determine your backup and recovery requirements.

The simplest backup system might be saving data and a copy of your application to tape or other remote devices and storing that data at a remote data center. A better strategy is to put redundant systems in place for each part of your deployment so that if one system fails, another is already available. The most robust solution is to maintain a mirror image of your site at a remote location and synchronize the image with the live data regularly. The latter solution is the most costly, but is immediately recoverable. The former solutions are less expensive, but require more time and effort to perform recovery.

This topic covers:

- [Deployment Architecture Overview](#)
- [Infrastructure](#)
- [Backup Strategies](#)

Deployment Architecture Overview

Setting up a rich development environment not only allows for easier site updates and maintenance, but also provides a quick path to rebuild a complete application as a recovery step, if needed. The deployment architecture includes the following elements:

- Build environment: a predictable, known build environment containing all the elements needed to build the deployment, including:
 - ⋮ JDK's
 - ⋮ SDK's
 - ⋮ Code repository (such as CVS)
 - ⋮ Libraries

The steps required to go from code to production vary and may be iterative, but the build environment should contain everything required to rebuild your deployment in a predictable way if necessary.

- QA area: a separate environment for performing quality assurance tasks. QA is the first environment in which work from (possibly) several engineers is integrated to run as a unit.

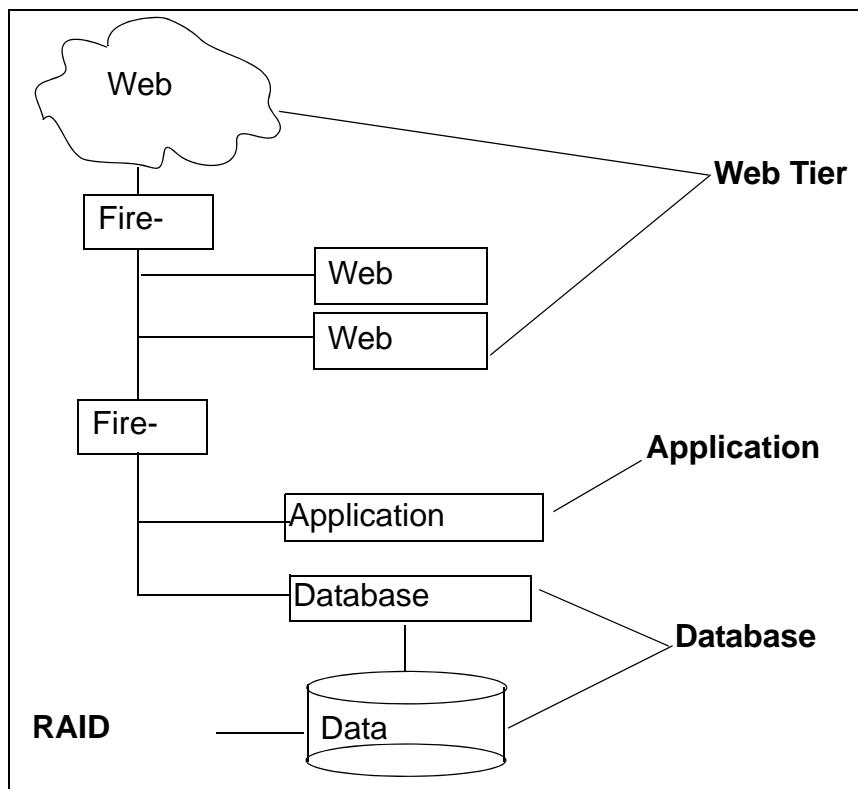
- Staging area: a separate environment in which the work integrated in QA now runs in a specific context that emulates production.

Infrastructure

A common strategy for establishing a robust infrastructure is “double everything”: put redundant systems in place so that when a primary system fails, a secondary system comes online as quickly as possible. The following figure shows a typical infrastructure consisting of three tiers:

- Web tier: all components handling requests from and serving content to Web browsers.
- Application tier: all components processing requests from and serving dynamic content to the Web tier, usually with data from the database tier.
- Database tier: all components serving data to the application tier.

The following diagram illustrates a typical deployment infrastructure diagram:



Two Web servers ensure that if one fails, the other can continue functioning. Having a second firewall in place to provide additional protection to data also satisfies certain regulatory requirements for securing data.

One strategy for providing physical protection for your data is to store production data in a RAID device. If a single drive fails, then there is no loss of data. There is a mechanical threshold with such a strategy: depending upon your configuration, if more than the threshold’s number of drives fails, you will lose data. That is something to consider as you determine your requirements.

Backup Strategies

There are different backup strategies for protecting data and for recovering your application servers and Web servers. Most backup strategies come down to saving copies of what is already running on the application and Web tiers. The following sections describe backup strategies for databases, application servers, and Web servers.

Database Recovery

Your backup strategy determines how quickly you can recover data after a disaster. Determine your acceptable timeline for getting your database back up and running, including time needed to rebuild the OS and reload the database if necessary, and plan your backup policy accordingly.

The following are the types of database backups you should perform:

- Checkpoint backups: database servers log activity on the transaction level as transactions are committed. Checkpoint backups write a log of transactions since the last time a checkpoint backup was done. Write the checkpoint log to a separate physical device. This creates a snapshot of the database activity on a transaction level. If the database fails, there's a chain of records that enable reconstructing the activity.
- The interval at which your deployment performs checkpoint backups is a business decision. If your site does millions of dollars of business each hour, doing checkpoint backups several times each hour would be advisable. If your site has only a few transactions each hour, you can do checkpoint backups less frequently. Determine an interval that allows data recovery at your level of business activity.
- Daily incremental backups: incremental backups save only those files that change each day. Performing daily incremental backups to a different facility, rather than using physical media, is a good strategy. The backup is effectively a disk-to-disk copy.
- Weekly full backups: full backups save the entire database, not just the files that have changed since the last backup. Performing full backups to a different facility is a good strategy.

A typical recovery scenario might be:

1. Perform the initial restore from the last full backup.
2. Next, perform restores in date order from the daily incremental backups.
3. Finally, reconstruct the last few hours' activity using the checkpoint backups.

Application Server and Web Server Recovery

When planning recovery policies for your application and Web servers, plan to be able to build exact replacements if either application or Web servers fail: this is the "build another one and go" principle.

Ensure that you have copies of everything that makes your application and Web server deployments unique: configuration files, properties files, the JVM, original source code from your CVS repository, and so on. Back up all static data kept on the Web tier, such as any custom JSP pages. Make sure that your backup process includes coverage for Web server and container configuration files or other files needed to operate your site, but which may not be considered as source code and are therefore not kept in a source code repository.

Use your QA and staging environments as starting points for rebuilding your servers and getting back to an operational state.

Item Modeling

This topic covers some of the best practices that you may want to keep in mind when modeling an item.

Absolute and Relative Paths

This topic refers to the use of absolute or relative paths to specify entities such as properties and rules. Paths have the following form:

<model group root node>.<path to the option item that has the property or rule>.<property name or rule name>

For example, consider the following absolute path to the property memoryProvided:

MXDS-7500.memory.sim256.memoryProvided

The model group's root node is MXDS-7500; the path to the option item that has the property memoryProvided is memory.sim256; memoryProvided is the property name.

If you plan to use a property or rule in more than one model, you can use special symbols to specify relative paths. For example, the "*" in the following path indicates that the path begins at the root of the model group hierarchy:

*.memory.sim256.memoryProvided

Beginning the path with a period (.) indicates "from the attachment point of the rule". For example, the "." in the following path indicates that "an option item called sim256 in an option class called memory in the current model".

.memory.sim256

Planning Considerations

A model represents a configurable item. When you sit down to plan a IBM® Sterling Web implementation, you start by considering how to design a model of the item. There is no one "right" way to model a particular item. On the other hand, there are an endless number of ways to create models that, while technically correct, are inefficient and hard to maintain.

The following are among the trade-offs to keep in mind.

- Cost factors: creation, maintenance, and performance.

You must balance the cost of creating the model, compared with the cost of maintaining it, compared with its expected performance. The cost of creating the model represents the one-time effort expended to develop it; the cost of maintaining the model represents the effort expended over time to maintain and enhance it, while the performance represents the execution speed of the model on a particular hardware platform. You can optimize for any one of these factors, but be mindful that trying to optimize for more than one of them means, in most cases, that you have competing goals. For example, a complex model might run fast, but would be hard to maintain. For an overview of models and best practice design principles, refer to the ["Models"](#) topic.

- Implementers' roles: model builder only, model builder and maintainer, model maintainer only.
The implementer's role influences their bias. For example, a consultant given a month to implement a model which will then be handed off to another group for maintenance may focus on designing the model quickly, rather than designing a model that will be easy to maintain. If the implementer will also maintain the model, the model may take longer to design and perhaps not run as fast, but will be easy to maintain long-term. Whatever your role, your goal should be to set up a model that will avoid problems down the road.

Models

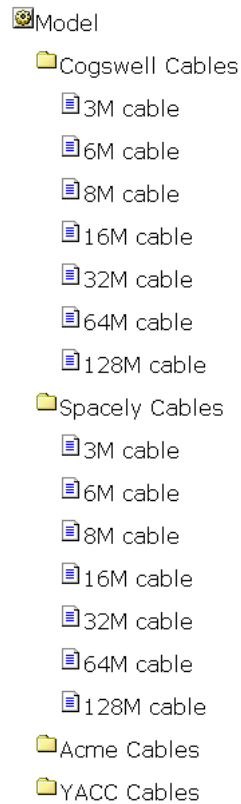
This section covers model design considerations and provides examples of designs that work, compared with designs that work best.

Make Models as Small and Simple as Possible

Model size matters. Large models take longer to render in the browser, are harder to maintain, and during configuration the Configurator "walks" the model structure a number of times to get prices, fire rules, and so on. Keeping the model size as small as possible through the use of subassemblies and other techniques described in this topic improves performance and decreases maintenance costs.

For example, suppose that you have a number of cable suppliers, each of which supplies a number of different lengths of cables. You want to allow the user to select the quantity, length and cable supplier.

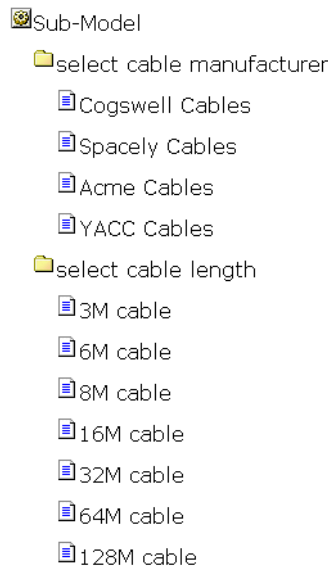
One way to do this is to create option items in your model to represent every single one of the available options, as shown in the following figure:



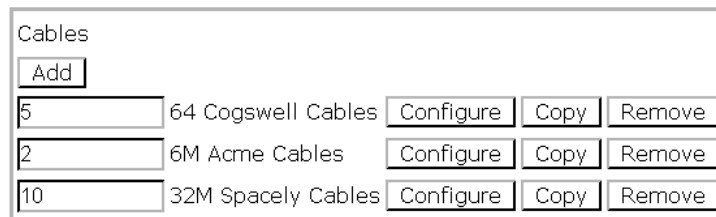
This approach will work, although it may be a bit tedious to implement and maintain and creates a model with many options that will never be of interest to the end user.

An alternative approach might be to implement the different cable length option items as an option item group, then include the cable length option item group under each of the manufacturers. This would ease maintenance: the modeler would have to look in only one place to update the cable option item information. However, this approach presents the end user with a huge list of cables to choose from and does not improve performance since there's still a large model to walk.

A better alternative is to create a submodel that allows the user to select a cable manufacturer and length, then use dynamic instantiation to let the user add as many different cable types and lengths as necessary, as shown in the following figure:



The following figure shows a sample cable selection UI that uses dynamic instantiation to allow end users to configure their cable selections.



As you can see, this approach keeps the model small. Maintaining this model will be easier since the modeler no longer has to deal with a huge number of duplicate option items, performance is better since the model is smaller, and configuration is easier for the end user since there isn't a long list of cable types and manufacturers to look through in order to find the right one.

Use *popup-qty* Controls for Entering Quantity


Sometimes the modeler wants to allow users to select an item, then enter the number of items they want. The best way to do this is to set the Option Class Display to *popup-qty*. When the end user selects an item, a quantity box will display, allowing the end user to enter the number of items they want.

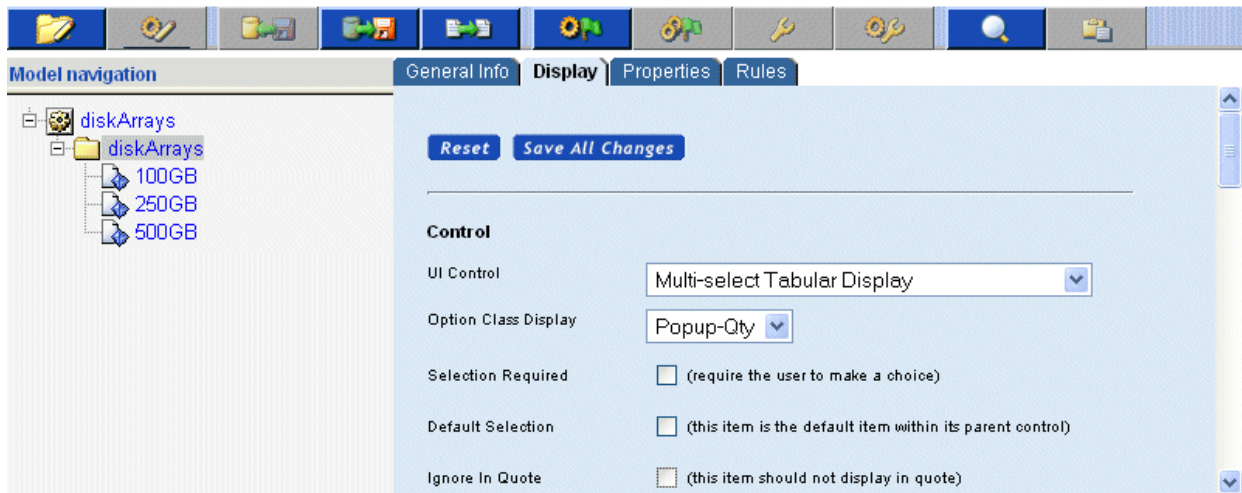
Some modelers do not like the placement of the quantity box, so instead use the User Entered Values (UEV) control to display an edit field next to the item in which users can enter a quantity. The problem is that the behavior of the *popup-qty* control differs significantly from the behavior of UEV controls: the *popup-qty* control has quantity processing built in, while UEV controls require additional work.

When an end user enters a quantity in a popup-qty box, the application automatically selects the quantity of the selected item. Any properties attached to the item are included in the configurator state (property pool), and the values of any numeric properties are multiplied by the quantity entered.

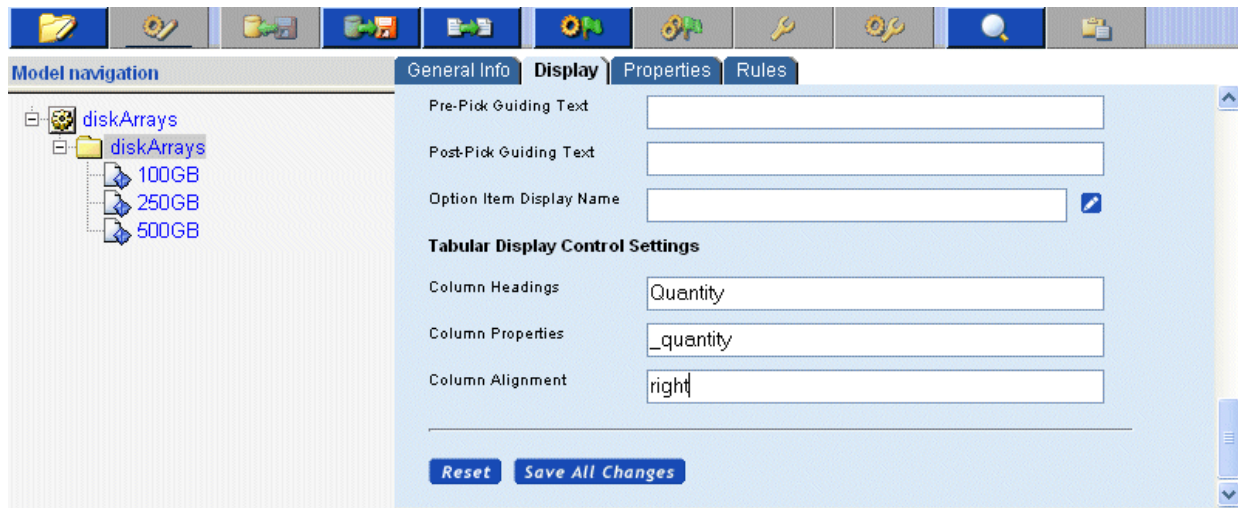
When a value is entered in a UEV control, nothing else happens. UEV controls were designed simply to capture some additional information from the user. To get the UEV to behave as a quantity, the modeler must write an expansion rule that takes the value entered in the UEV control and picks that many of the selected item. Using the value entered in the UEV control to set the `_quantity` property using an assignment rule will not work as expected, since this does not automatically create instances of the item's properties in the property pool.

To display a popup-qty box beside the item selected, use the popup-qty Option Class Display style and one of the tabular displays with quantity controls. This will ensure the correct number of items are selected and the correct properties are copied to the property pool.

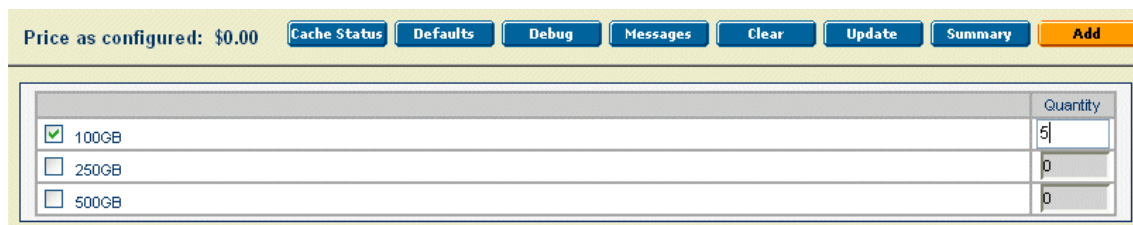
For example, the following figure shows how to set up a popup-qty control using the Visual Modeler. From the **Models and Groups** panel, select the model you wish to modify, then click the  icon. The Model navigation page is displayed. Click the option group you wish to modify, click the **Display** tab, then select Multi-select Tabular Display from the **UI Control** drop-down list, as shown in the following figure:



Scroll to the bottom of the page and enter the Column Headings, Column Properties, and Column Alignment settings. The following figure shows sample settings:



Finally, compile and test the model. You should see a popup-qty control placed as you specified on the Product Configurator page.










Properties

Properties are ubiquitous in the IBM® Sterling Configurator. Modelers attach properties to models, option classes and option items and then write rules that work on these properties in order to display messages, show, hide, or select items, and even set the values of other properties. Considering the important part that properties play in modeling a configurable item, some care should be given to how properties are defined and used. This section outlines some useful tips and procedures to follow when defining and attaching properties.








Use Meaningful Property Names

Sometimes when developing a model, especially when under severe time constraints, the modeler is tempted to take shortcuts in order to speed the development process. One of the most common shortcuts is to create properties with short, and often vague or cryptic, names. This may speed the development of the model in the short term, but dramatically increases the amount of effort required to maintain the model. The modeler should always design their models so that it is immediately obvious what a given property represents. The more meaningful the name you give to a property, the easier it will be to debug and maintain the model now and in the future.

Consider the following example:

model node	properties
 Model	tmr=sum(mr) tmp=sum(mp) amr=value(tmr)-value(tmp)
 OC1	 oi1 mr=1  oi2 mr=2
 OC2	 oi3 mp=2  oi4 mp=1

At first glance it may not be obvious what the properties assigned to this model are trying to accomplish. With a little more time spent creating meaningful names it becomes much easier to grasp the essence of all the properties and how they relate to one another.

model node	properties
 Model	total_mem_required=sum(mem_required) total_mem_provided=sum(mem_provided) additional_mem_required=value(total_mem_required) - value(total_mem_provided)
 OC1	 oi1 mem_required=1  oi2 mem_required=2
 OC2	 oi3 mem_provided=2  oi4 mem_provided=1








Do Not Use the Same Property to Mean Two Different Things

Often, in their haste to implement a particular feature, a modeler will reuse an existing property instead of creating a new property designed specifically for the problem at hand. This has two possible implications:

- The model may be harder to understand if the existing property name bears no relation to the problem at hand.
- Re-use of the property name may actually cause errors in the model if the re-use conflicts with the property's original use.

Let us revisit our example from the previous section. Suppose that our modeler created a property to store memory required and called it "memory" (see previous section for why that was a bad choice to begin with).

Now when he determines that he needs a property to store memory provided, he notices that he has a property called memory and decides to use it instead of creating a new property.

model node	properties
 Model	total_memory_required=sum(memory)
 OC1	
 oi1	memory=1
 oi2	memory=2
 OC2	
 oi3	memory=2
 oi4	memory=1

Now, at first glance, it looks like all option items require some amount of memory, instead of two items requiring memory and two providing it. Not only that, but our total_memory_required property will no longer have the correct value, since it now performs a sum of both memory required and memory provided. If modeled in this fashion, then the modeler will have to do extra work to separate out the specific instances of the properties he needs: such as using full or relative paths to the items containing the appropriate property instances. (See "Rules" on why using paths to specific instances of properties can be a bad idea.)

Define Properties at the Appropriate Level in the Model Hierarchy

Properties may be defined at any level in the model group hierarchy, from the root model group level to the individual model level. Where a property is defined determines which models can see and make use of the property. A little thought during the design of your models will speed model development and help prevent property clutter. Use the following guidelines to determine where a property should be defined:

If a property will only be used in a particular model, then define the property at the model level.

If a property will be used in more than one model within a particular model group, then define the property at that model group level.

If a property will be used in models that span model groups, then define the property in the first model group that contains all of the model groups whose models will use the property.

As a last resort, define the property at the root model group node.

Using Multiple Properties with the Same Value

Multiple properties with the same value can sometimes make a model easier to build and maintain. This concept may be confusing at first and is best demonstrated by an example. Suppose that you are building a model that allows the user to choose from a selection of disks arrays. Each type of disk array has some

number of disks associated with it. The user can choose multiple disk arrays of any type. One of the pieces of information that you need to calculate is the total number of disks that the user has selected (see below).

model node	properties
DiskArray SubModel	total_disks=sum(disks)
100GB disk arrays	
blue	disks=4
mauve	disks=6
250GB disk arrays	
blue	disks=8
mauve	disks=12

Now let us assume that you realize that you also need to know the number of 100GB disk arrays and the number of 250GB disk arrays. Instead of calculating these values by specifying item paths to the properties that we want, or writing rules that have to be attached at a particular point in the model, or re-working all the disk and total_disk properties, we can simply define a couple of new properties that have the same values as our old disk property (see below).

model node	properties
DiskArray SubModel	total_disks=sum(disks) total_100GBdisks = sum(100GB_disks) total_250GBdisks = sum(250GB_disks)
100GB disk arrays	
blue	disks=4 100GB_disks=4
mauve	disks=6 100GB_disks=6
250GB disk arrays	
blue	disks=8 250GB_disks=8
mauve	disks=12 250GB_disks=12

Now, if we want the total disks, we can still get sum(disks). If we want the individual values, then we can get those as well: and all without specifying paths to individual properties or modifying the work that we had already done.

Use Worksheets to Simplify Property Assignment

When developing a model, it is often necessary to assign the same set of properties to multiple option classes or option items. Worksheets are very useful in this case, since they allow you to rapidly set the values for a particular property on any number of option classes or items. This is especially true when using a formula to set the value of a property in multiple places. The modeler can simply copy and paste the formula onto all the items he wishes. An example of this is shown below. Here we have some display properties that are set

for each item within a tabular display. We use a worksheet to allow us to easily cut and paste the formulas for col1 and col2 to each item in the option class.

modelDisplay		
Item	col1	col2
U100	$\frac{\text{expand("min_array_disk")}}{\text{expand("max_array_disk")}}$	$\frac{\text{expand("min_cache_memory")}}{\text{expand("max_cache_memory")}}$ GB
U600	$\frac{\text{expand("min_array_disk")}}{\text{expand("max_array_disk")}}$	$\frac{\text{expand("min_cache_memory")}}{\text{expand("max_cache_memory")}}$ GB
U1100	$\frac{\text{expand("min_array_disk")}}{\text{expand("max_array_disk")}}$	$\frac{\text{expand("min_cache_memory")}}{\text{expand("max_cache_memory")}}$ GB

An added benefit of using worksheets is that can provide a concise picture of a section of the model. With a little thought and planning, a worksheet can provide an overview of a particular section of the model or a complete representation of the solution to a particular problem. Below is a different view of the same option class. In this case, we are interested in seeing all the min and max properties that are set for each of the option items.

modelMinMax				
Item	min_array_disk	max_array_disk	min_cache_memory	max_cache_memory
U100	4	252	4	64
U600	64	508	6	$\text{=(value(exp_cache) == 0) ? 64 : 128}$
U1100	128	1148	6	128

Avoid Chaining Property Formulas

Properties attached to an item do not have any notion of sequence. By this we mean that, when using formulas to set property values, we cannot rely on any particular order of evaluation of the formulas. If property A contains a formula and property B contains a formula that relies on property A, then we have no guarantee that the rule created from formula B will fire after the rule created for formula A. In order to get around this issue, the modeler has two choices:

- Turn the first formula into a rule that fires before the second formula is evaluated. All rules generated from formulas have a priority of 50. By creating a rule for the first formula, and setting its priority to be less than 50, we ensure that the value of property A will be set before the value of property B is calculated.
- Turn on repeat rule firing. In this case the first phase of rule-firing will calculate the value for property A. The second pass of the rule-firing loop will calculate the value of property B based on the value of property A computed in the first pass. Note: massive amounts of chaining of formulas in this way may result in degradation of performance due to the number of passes through the rule-firing loop necessary to satisfy all the conditions. For this reason, we recommend the first alternative and advocate limiting formula chaining as much as possible.

Rules

Rules affect the efficiency and ease of maintenance of your model. This section describes considerations to keep in mind while writing rules.

Rule Firing Conditions

Rule conditions are created by applying boolean operations to relational expressions. A relational expression is the comparison of one function/property pair with another function/property pair using relational operations such as less than, equal to, greater than, in, not in, and so on. The result is either true or false. Boolean operators like AND and OR wrap sets of these relational expressions. The relational expressions are called *fragments*, as they are fragments of a rule. The left-hand-side of the relational operator is often abbreviated LHS, while RHS stands for right-hand-side.

Order Rule Fragments So That Rules Fire Only When Necessary

The evaluation of rule fragments determines when a rule fires, so the order in which fragments appear in a rule is important. The more quickly the model can determine whether a rule is true or false, the more efficient the model can be. And of course, the more quickly the model determines that a rule should not be fired, the sooner the model can continue to other processing. Placing rule fragments in order, from most likely to prevent the rule from firing to least likely to prevent the rule from firing, can improve performance.

Always test your rules to ensure that they fire only when appropriate. Knowing under what circumstances a rule's results will or will not be used is also important. For example, an expansion rule that always fires but will not pick something in the expansions section if the quantity formula results in zero, or if there are not any matches for the formula in the > and <= fields in the expansions section, is very inefficient.

Create General-Purpose Rules

Whenever possible, write rules that are as general as possible. For example, the following rule can be attached to any item to which the `productType` and `handsetType` properties are attached:

```
If propval(productType) != value(selectProductType)
and propval(handsetType) != value(phonePreference)
    set _isVisible=0
```

This rule fires only for items where the `productType` property is attached AND does not match the selected product types AND if the selected phone preferences do not match the current item's preferences. A general rule such as this one can replace dozens of other specific rules such as the following specific ones:

```
If propval(productType) == literal("handset")
and propval(handsetType) != literal("camera")
and value(phonePreference) == literal("camera")
    set _isVisible=0
If propval(productType) == literal("handset")
and propval(handsetType) != literal("flip")
and value(phonePreference) == literal("flip")
    set _isVisible=0
```

...

Use Formulas Where Appropriate

In many circumstances, formulas can be used instead of rules. During modeling, formulas are maintained as attached properties that have as their value an expression that is evaluated at runtime. If any of the functions referenced in the expression cannot be evaluated, the formula acts like a rule that hasn't fired. If multipass rule firing is turned on, the formula will be reevaluated during each firing pass until rule firing ends or until the formula produces its result.

Use a formula rather than a rule when the only condition for requiring that you compute a result is that the function/properties used in the formula have values.

For example, suppose that you want to compute the turning radius for truck components such as axle and wheelbase to ensure that a user's choice of truck components makes sense. You might attach a formula to the relevant truck components to compute the turningRadius as follows:

```
turningRadius = value(axleTurnFactor) * value(wheelBaseTurnFactor) *  
sum(turningElements)
```

This formula will fire when each of the value(axleTurnFactor), value(wheelBaseTurnFactor), and sum(turningElements) expressions all produce numeric results.

The equivalent rule is as follows:

```
if (value(axleTurnFactor) >= 0 or value(axleTurnFactor) < 0)  
and (value(wheelBaseTurnFactor) >= 0 or value(wheelBaseTurnFactor) < 0)  
and (sum(turningElements) >= 0 or sum(turningElements) < 0)  
    turningRadius = value(axleTurnFactor) * value(wheelBaseTurnFactor) *  
sum(turningElements)
```

The condition portion of the rule is quite long and seems to always evaluate to true. However, functions can return NULL if a property that they reference does not exist, so this rule is really checking that the result is non-NULL by evaluating whether a returned value is >= 0 or < 0.

Avoid Specifying Paths to Instances of Items or Properties

The LHS and RHS of a rule fragment consist of a function and a property name. The property name can contain both relative and absolute path information. However, specifying a property's path information in a rule fragment can result in the rule becoming inoperable if the path information or option classes change.

For example, the following rule references wheelSize and wellSize using fully specified path information. If the modeler ever needs to rename either the wheels or fender option classes, or wishes to reuse the rule in some other model, the rule may not operate correctly.

```
If value(*.wheels.wheelSize) == literal("17in")  
and value(*.fender.wellSize) < literal(17)  
    set _isVisible=0
```

Use path information only if you want to access one specific instance of a property, and then only if it isn't possible to make a new property type to hold this value. If you must reference a property's path name, it is often better to use relative pathnames rather than absolute pathnames.

Constraint Tables vs. Rules

This section explains the trade-off between using constraint tables to limit customer choices vs. using rules. Constraint tables limit a customer's choice of one or more option items based on the customer's choice of another option item. For example, the choice of an exterior color for a car might limit the choice of interior colors.

Constraint tables work best for simple validation, for example, an option item does or doesn't work with another option item. Simple constraint tables are easier to maintain than rules. However, large, complex constraint tables can be hard to maintain and can lead to performance issues.

Constraint tables are turned into rules internally.

Rules are best for expressing complex validation issues, and are more versatile than constraint tables. While both constraint tables and rules can display error messages, you can also create rules to set properties or make choices.

Modular Development

This section explains some of the techniques for simplifying model creation and maintenance. Selecting the appropriate technique may have a significant impact on model performance.

- ▣ Using Option Class Groups, Option Item Groups, and Sub-assemblies:

This technique works well when a group of options is repeated in many different models.

For example, suppose that every computer you sell includes a list of hard drives that the user can choose from. Creating Option Class Groups, Option Item Groups, and Sub-assemblies allows the modeler to create and maintain common information in one place, then use it in many places.

One drawback is that this technique can lead to overly large models if a sub-assembly is included in the same model many times.

- ▣ Sub-model punch-in and punch-out:

This technique is useful when a configuration contains a selection that is also configurable. You can use sub-model punch-in and punch-out to nest complex configurations within one overall selling model.

One drawback is that all copies of the configured item will have the same configuration.

- ▣ Dynamic instantiation:

This technique allows multiple instances of a configured item within a single model. Each instance can have a different configuration

Tools

Modeling can be a time consuming and tedious exercise, but in the end the correctness of the modeling and the scalability of the created solution are key to the success of the project. To aid in creating scalable and correct models, we have developed a collection of tools that can be used in various phases of development to guide the modeler. During development, the trace log and the model reporting tool can help the modeler determine which models to debug. Before pushing models into production, their scalability and stability can be tested using the load testing platform. Finally, during execution, the model cache status page can provide insights into the model's usage of the system, and the log analyzer can be used to make sense out of megabytes worth of log information.

Using the Trace Log

The trace log shows the execution of the rules engine. This is often, though not always, the most time consuming part of each request that the configurator makes to the server. The trace log is designed to provide the information necessary to debug rules that are misbehaving and to track the execution time of rules, so always start your debugging by reviewing the trace log.

You create trace logs using the Visual Modeler. To do so:

1. Go to Model Group navigation and navigate to the model you wish to debug.

2. Select the model from the Models and Groups panel.
3. The model displays in the Model Preview tab.
4. Click the Test icon.
5. The model runs in a separate window.
6. Click Debug.

The trace log appears in a separate window.

The log consists of two sections. The first section is the rule firing trace and the second section is the property pool as it exists at the end of rule firing.

The following illustration shows a section of a sample rule firing trace.

Rule Firing Trace		
#	(ms)	Result
0	0	Applying picks
1	0	Firing phase [0]:begin
2	0	Firing rules on MXDS-7500.Disk Drives
3	0	MSG_E_Available_HDD_Slots ==> fires on TRUE - priority = 50
4	0	Property not found [MX75_HDD_Ordered or MX75_Bays_Available], taking null action
5	0	took 0ms.
6	0	Firing rules on MXDS-7500.Memory
7	0	MSG_E_MX75_Memory_Software_Check ==> fires on TRUE - priority = 50
8	0	TESTING:sum(MX75_Mem_Ordered) <sum(MX75_Mem_Required) [nullreturn=false]
9	0	FALSE: 0.0 < 0.0
10	0	FALSE: sum(MX75_Mem_Ordered) <sum(MX75_Mem_Required) [nullreturn=false]
11	0	took 0ms.
12	0	Firing rules on MXDS-7500.Placeholder for auto memory selection
13	0	ASG make placeholder invisible ==> fires on TRUE - priority = 50
14	0	Left side property [MX75_Mem_Auto_Select] not found, taking null action
15	0	took 0ms.
16	0	Firing rules on MXDS-7500.AutoMemory
17	0	EXP_MX75_Automatic_Memory_Selection ==> fires on TRUE - priority = 50
18	0	Left side property [MX75_Mem_Auto_Select] not found, taking null action
19	0	took 0ms.
20	0	Firing rules on MXDS-7500.Software.Application
21	15	EXP_MX75_Fire_Wire ==> fires on TRUE - priority = 50
22	15	Left side property [MX75_Video_Editing] not found, taking null action
23	15	took 15ms.

The rule firing trace has three columns:

- ≡ A sequence number, useful for communicating with others about rule issues. It's easy to tell someone, "See line 42 where it says Xxx?".
- ≡ Elapsed time. This logs how long it took from the time the log entry was made until the start of rule firing.
- ≡ The body of the trace log. This shows aspects of the rule firing, such as a condition being evaluated, an assignment occurring, the start of a rule or the conclusion of a rule, and so on.

The log shows the number of milliseconds needed to fire a rule after each rule firing entry. The total number of milliseconds needed to run the model is logged at the end of the rule firing trace.

The property pool trace also presents three columns:

- ≡ Name is the full path name to the item and the property on that item.
- ≡ Type is the property type for the named property, such as Numeric, List, or String.

- Value is the value of the property after the rule has fired.

The following illustration shows a section of a sample property pool trace.

Property Pool		
Name	Type	Value
MXDS-7500.CONFIG: FIRST FIRE	Numeric	1.0
MXDS-7500.MX75_Bays_Available	Numeric	2.0
MXDS-7500.MX75_Card_Slot_Available	Numeric	4.0
MXDS-7500.MX75_Mem_Ordered	Numeric	0.0
MXDS-7500.MX75_Mem_Required	Numeric	0.0
MXDS-7500.Service Options.View Service.UI: COLUMN SPAN	Numeric	2.0
MXDS-7500.Placeholder for auto memory selection.UI: CONFIG CELL HTML CLASS	String	configsubcell_plain
MXDS-7500.Accessory Cards Message.Accessory Card Image.UI: CONFIG CELL HTML CLASS	String	configsubcell_plain
MXDS-7500.Microprocessor.UI: CONSTANT GUIDING TEXT	String	Dual processor capable motherboard, supporting Intel processors
MXDS-7500.Microprocessor.UI: CONTROL	String	RADIO
MXDS-7500.Disk Drives.UI: CONTROL	String	RADIO
MXDS-7500.Placeholder for auto memory selection.UI: CONTROL	String	controls/allpicked.jsp
MXDS-7500.Software.UI: CONTROL	String	CHECKBOX
MXDS-7500.Accessory Cards Message.UI: CONTROL	String	controls/allpicked.jsp
MXDS-7500.Accessory Cards Message.Accessory Card Image.UI: CONTROL	String	controls/allpicked.jsp
MXDS-7500.Accessory.Graphic Cards.UI: CONTROL	String	RADIO
MXDS-7500.Accessory.Cards.UI: CONTROL	String	CHECKBOX
MXDS-7500.Accessory.Network Cards.UI: CONTROL	String	RADIO
MXDS-7500.Service Options.View Service.UI: CONTROL	String	CHECKBOX
MXDS-7500.Placeholder for auto memory selection.UI: DEFAULT SELECTION	String	no
MXDS-7500.Accessory Cards Message.UI: DEFAULT SELECTION	String	no
MXDS-7500.Accessory Cards Message.Accessory Card Image.UI: DEFAULT SELECTION	String	no

Use this log "single user" to get a feel for how extensive the rules are per click. Check how long is it taking to fire the rules. If the answer is more than 100-200ms you may have scalability problems. If you do, use the trace log to figure out if any particular rules are performing badly.

Using the Model Reporting Tool

The model reporting tool can provide an overview of a model's size relative to other models. Use it to help make decisions about which models to test. You can track the test results over time so that you can determine the amount of change to the model.

Using Load Testing Tools

Load testing tools help you determine how your model will perform once deployed. Before using the load testing tools:

- Understand what is being tested.
- Isolate your test cases so that you know what the impact means (local vs. remote LAN testing, testing with and without clustering, with and without web fronting, and so on).

- Understand that as models change, so must any scripts that you use to perform testing and replay test scenarios.
- If they are more global than the current model group, then define them at the lowest point in the model group tree that is an ancestor of a model where you wish to use the property.

#	(ms)	Result
0	0	Applying picks
1	0	Firing phase [0]:begin
2	0	Firing rules on MXDS-7500.Disk Drives
3	0	MSG_E_Available_HDD_Slots ==> fires on TRUE - priority = 50
4	0	Property not found [MX75_HDD_Ordered or MX75_Bays_Available], taking null action
5	0	took 0ms.
6	0	Firing rules on MXDS-7500.Memory
7	0	MSG_E_MX75_Memory_Software_Check ==> fires on TRUE - priority = 50
8	0	TESTING:sum(MX75_Mem_Ordered) <sum(MX75_Mem_Required) [nullreturn=false]
9	0	FALSE: 0.0 < 0.0
10	0	FALSE: sum(MX75_Mem_Ordered) <sum(MX75_Mem_Required) [nullreturn=false]
11	0	took 0ms.
12	0	Firing rules on MXDS-7500.Placeholder for auto memory selection
13	0	ASG make placeholder invisible ==> fires on TRUE - priority = 50
14	0	Left side property [MX75_Mem_Auto_Select] not found, taking null action
15	0	took 0ms.
16	0	Firing rules on MXDS-7500.AutoMemory
17	0	EXP_MX75_Automatic_Memory_Selection ==> fires on TRUE - priority = 50
18	0	Left side property [MX75_Mem_Auto_Select] not found, taking null action
19	0	took 0ms.
20	0	Firing rules on MXDS-7500.Software.Application
21	15	EXP_MX75_Fire_Wire ==> fires on TRUE - priority = 50
22	15	Left side property [MX75_Video_Editing] not found, taking null action
23	15	took 15ms.

The rule firing trace has three columns:

- A sequence number, useful for communicating with others about rule issues. It's easy to tell someone, "See line 42 where it says Xxx?".
- Elapsed time. This logs how long it took from the time the log entry was made until the start of rule firing.
- The body of the trace log. This shows aspects of the rule firing, such as a condition being evaluated, an assignment occurring, the start of a rule or the conclusion of a rule, and so on.

The log shows the number of milliseconds needed to fire a rule after each rule firing entry. The total number of milliseconds needed to run the model is logged at the end of the rule firing trace.

The property pool trace also presents three columns:

- Name is the full path name to the item and the property on that item.
- Type is the property type for the named property, such as Numeric, List, or String.
- Value is the value of the property after the rule has fired.

The following illustration shows a section of a sample property pool trace:

Property Pool		
Name	Type	Value
MXDS-7500.CONFIG: FIRST FIRE	Numeric	1.0
MXDS-7500.MX75_Bays_Available	Numeric	2.0
MXDS-7500.MX75_Card_Slot_Available	Numeric	4.0
MXDS-7500.MX75_Mem_Ordered	Numeric	0.0
MXDS-7500.MX75_Mem_Required	Numeric	0.0
MXDS-7500.Service Options.View Service.UI: COLUMN SPAN	Numeric	2.0
MXDS-7500.Placeholder for auto memory selection.UI: CONFIG CELL HTML CLASS	String	configsubcell_plain
MXDS-7500.Accessory Cards Message.Accessory Card Image.UI: CONFIG CELL HTML CLASS	String	configsubcell_plain
MXDS-7500.Microprocessor.UI: CONSTANT GUIDING TEXT	String	Dual processor capable motherboard, supporting Intel processors
MXDS-7500.Microprocessor.UI: CONTROL	String	RADIO
MXDS-7500.Disk Drives.UI: CONTROL	String	RADIO
MXDS-7500.Placeholder for auto memory selection.UI: CONTROL	String	controls/allpicked.jsp
MXDS-7500.Software.UI: CONTROL	String	CHECKBOX
MXDS-7500.Accessory Cards Message.UI: CONTROL	String	controls/allpicked.jsp
MXDS-7500.Accessory Cards Message.Accessory Card Image.UI: CONTROL	String	controls/allpicked.jsp
MXDS-7500.Accessory.Graphic Cards.UI: CONTROL	String	RADIO
MXDS-7500.Accessory.Cards.UI: CONTROL	String	CHECKBOX
MXDS-7500.Accessory.Network Cards.UI: CONTROL	String	RADIO
MXDS-7500.Service Options.View Service.UI: CONTROL	String	CHECKBOX
MXDS-7500.Placeholder for auto memory selection.UI: DEFAULT SELECTION	String	no
MXDS-7500.Accessory Cards Message.UI: DEFAULT SELECTION	String	no
MXDS-7500.Accessory Cards Message.Accessory Card Image.UI: DEFAULT SELECTION	String	no

Use this log "single user" to get a feel for how extensive the rules are per click. Check how long is it taking to fire the rules. If the answer is more than 100-200ms you may have scalability problems. If you do, use the trace log to figure out if any particular rules are performing badly.

Using the Model Reporting Tool

The model reporting tool can provide an overview of a model's size relative to other models. Use it to help make decisions about which models to test. You can track the test results over time so that you can determine the amount of change to the model.

Using Load Testing Tools

Load testing tools help you determine how your model will perform once deployed. Before using the load testing tools:

- Understand what is being tested.
- Isolate your test cases so that you know what the impact means (local vs. remote LAN testing, testing with and without clustering, with and without web fronting, and so on).

- Understand that as models change, so must any scripts that you use to perform testing and replay test scenarios.

Cache Status

- `cmd=configstatus` shows the current contents of the cache

Performance

Rules

- Excessive paths to items:
- A rule that adds memory by:

```
totalMem = value(*.adapter.1.memory) + value(*.adapter.2.memory) +  
value(*.adapter.3.memory) + value(*.adapter.4.memory)
```
- will perform much more slowly than:
- `totalMem = sum(memory)`
- If the memory property exists in other places for other uses so that `sum(memory)` would produce the wrong value, then introduce additional properties on the adapter items 1-4 called `adapterMemory`, and use:
- `totalMem = sum(adapterMemory)`.
- This is much less maintenance effort than maintaining:

```
totalMem = value(*.adapter.1.memory) + value(*.adapter.2.memory) +  
value(*.adapter.3.memory) + value(*.adapter.4.memory)
```
- Write rules to fire only when they are needed:
- A rule that assigns `totalMem = sum(mem)` only needs to fire if `count(mem) > 0`

Properties

- Define properties at the correct position in the model group hierarchy:
 - If they are local only to this model, then define them in the model.
 - If they may be used by other models within this model group, then define them in the current model group.
 - If they are more global than the current model group, then define them at the lowest point in the model group tree that is an ancestor of a model where you wish to use the property.

Archiving Data

Managing your data is key to protecting your business. Archive your production data regularly and set up primary and secondary locations for storing your database archives, preferably off-site. Establish a regular schedule of archival activities, including daily incremental backups and weekly full backups (more often if your business volume demands it).

Updating Statistics

Updating the database statistics allows the database query optimizer to re-examine database indexes and re-compute the most efficient paths for retrieving data. This section presents two scripts: one to update statistics for an Oracle database, and one to update statistics for a SQL Server database.

Please consult your DBA to get the statistics updated on the tables properly or refer to your database documentation for further help.

Updating Statistics For an Oracle Database

The following example shows how to update statistics for an Oracle database at the schema level. Replace *schema name*, *owner name*, and *table name* with the appropriate schema, owner, and table names.

```
EXEC DBMS_STATS.GATHER_SCHEMA_STATS(  
  ownname=> 'schema name' ,  
  cascade=> TRUE,  
  estimate_percent=> DBMS_STATS.AUTO_SAMPLE_SIZE,  
  degree=> DBMS_STATS.AUTO_DEGREE,  
  granularity=> 'AUTO',  
  method_opt=> 'FOR ALL COLUMNS SIZE AUTO');
```

The following example shows how to update statistics for an Oracle database at the table level.

```
exec dbms_stats.gather_table_stats(  
  ownname=> 'owner name',  
  tabname=> 'table name',  
  estimate_percent=> DBMS_STATS.AUTO_SAMPLE_SIZE,  
  cascade=> DBMS_STATS.AUTO_CASCADE,  
  degree=> null,  
  no_invalidate=> DBMS_STATS.AUTO_INVALIDATE,  
  granularity=> 'AUTO',  
  method_opt=> 'FOR ALL COLUMNS SIZE AUTO');
```

You can also update statistics at the database or indexes level depending on your requirements.

Updating Statistics For a SQL Server Database

The following example shows how to update statistics for a SQL Server database at the table level. Replace *table name* and *index name* with the appropriate table and index names.

```
UPDATE STATISTICS ON <table name> [ . <index name> ]  
  WITH FULLSCAN {, NORECOMPUTE }
```

JVM Memory and Tuning Guidelines

When you encounter memory-related issues, adjusting the JVM memory settings can get you back to a sane, working environment. This section presents guidelines for JVM memory settings and performance tuning. You should be familiar with your JVM and servlet container environment to apply these guidelines.

Adjusting JVM Memory Settings

In general, you should allocate as much memory as possible to the JVM running your application server. You can do this by setting the JVM memory configuration as follows:

- -Xmx should be between 80% and 100% of the machine's physical memory. If you set -Xmx too small, the application server may fail with an OutOfMemory error. If you set -Xmx too large, the memory's footprint is larger and you run the risk of the Java's heap being swapped out, causing other performance problems.
- -Xms should be approximately half of the -Xmx setting. If you have historical data about your application server's stable memory-usage point, then set -Xms to be around that value.

Another option is to set -Xms to the memory-usage value observed at the end of InitServlet. This will ensure that, at a minimum, DEBS initialization will complete with as little garbage collection as possible. To get the memory usage value, perform the following steps:

- a. Set -Xms to be the same as -Xmx
 - b. Start up your Visual Modeler deployment and wait until initialization is complete
 - c. Access your e-commerce site's home page
 - d. Open the **debs.log** file in a text editor and examine the log entry that is similar to the following:
2003.03.18 ... END Request ... Mem=129380744/388726784/391291344 ...
 - e. The first number after **Mem=** is the current memory usage after initialization. Set -Xms to that number: in the above example, use the value -Xms128m.
- -XX:MaxPermSize controls the allocation size for system-like reflective objects such as Class and Method. Its recommended initial value is 128m.

For Web applications, the allocated space fills up quickly because the *.jsp files are converted into *.java files, then into *.class files which are loaded into the memory space specified by -XX:MaxPermSize. Starting with Java version 1.4.2, you can use -XX:+PrintGCDetails to monitor the details of this space, named permanent generation.

Be careful not to make memory-related changes that might contradict what the application server currently supports. DEBS needs to co-exist with the application server in the same VM so when in doubt, double-check the application server documentation or contact your application server's Support organization. For example, suppose that the current application server documentation states that the JVM setting -server is not supported. In that case, don't set -server.

As a last troubleshooting resort, start the VM with no additional arguments and incrementally add one argument, restart, observe the results, then add one more, continuing until you get good results.

Additional Performance Tuning

Additional performance tuning can be done around Java garbage collection activities and by adjusting memory settings for other areas, such as for threads, JVM stacks, or native structures or code. Use the Log Analyzer tool or check the **debs.log** file directly to make observations and determine performance problem areas.

Tracing Garbage Collection Activities

If you observe unexplained pausing, then it is possible that the VM is being paused for a full garbage collection. To confirm that this is what is happening, use the JVM setting `-verbose:gc` to enable recording of garbage collection events in the **debs.log**. Garbage collection events are of the following types:

```
[GC 325816K->83372K(776768K), 0.2454258 secs]
[Full GC 267628K->83769K(776768K), 1.8479984 secs]
```

A minor collection should be less than half a second. A major garbage collection should be less than three seconds. Anything more than three seconds indicates an out-of-range condition and should be looked into.

Other garbage collection trace settings you might want to look into are:

- The JVM `-server` setting: this setting adjusts some initial Java heap settings so that they are more appropriate for a server environment. Set the `-server` value unless your application server does not support it.

There is a known problem with the `-server` setting related to a bug in JIT (just-in-time) compilation which causes the value used by the data service to change unexpectedly. The result is that DEBS will fail to initialize (InitServlet fails). Contact your representative to learn how to disable JIT compilation for certain methods.

Some application servers recommend using the VM setting `-server`. In particular, the value of `-XX:NewRatio` for `-server` is 2 (the default value for the `-client` setting is 8). For more information about the `-server` and `-client` settings, see the Sun documentation at the following URL's:

http://java.sun.com/docs/books/performance/1st_edition/html/JAppHotspot.fm.html#998292

http://java.sun.com/docs/books/performance/1st_edition/html/JAppHotspot.fm.html#998359

- The `-Xincgc` setting: this setting enables incremental garbage collection. Setting `-Xincgc` reduces large pauses due to full garbage collection. When you use this setting, bear in mind that you are shifting the time spent to perform one major collection to several minor collections. There is an overhead cost associated with this shift, usually around 10%.
- If you are getting `OutOfMemoryError` messages, the you should first increase the value of `-Xmx`, ensuring that `-Xmx` is no more than the value of the machine's physical memory. If it appears that you are getting `OutOfMemoryError` messages when the current heap usage (where new objects are allocated) is nowhere near the value of `-Xmx`, then there is a possibility that other areas of memory allocation are exhausted. Examine the Log Analyzer report and check the following possible areas:
 - ⌚ Due to Classes: try setting `-XX:MaxPermSize=128m`
 - ⌚ Due to Threads: try adjusting the stack using `-Xss=512k`
 - ⌚ Due to JVM Stacks: try adjusting the stack using `-Xss=512k`

- Due to Native data structures: try adjusting the OS swap size
- Due to Native codes: try adjusting the OS swap size

Log Analyzer Tool

The Log Analyzer is an open source tool that can help with your analysis of the Visual Modeler **debs.log** entries. The tool provides a view of key performance indicators: threads, memory, requests, and sessions, as well as response times sorted by user and request type.

Using the Log Analyzer as part of a daily routine of monitoring your deployment provides these advantages:

- Daily log analyzer reports add reliability and stability to your deployment. The generated data can provide an early warning about potential problems, making it possible to prevent outages. For example, using the daily log analyzer reports, you can pro-actively plan to re-start an application server when it reaches near-maximum memory usage.
- Daily log analyzer reports provide the basis for troubleshooting a current problem. By examining the reports, you can determine when the problem started and correlate it with events such as an OS upgrade.
- Daily log analyzer reports provide a focal point for making incremental improvements. By reviewing the log analyzer report daily, you can generate a to-do list to plan when to restart your application server, clean up any exception lists, track down hanging threads, or to provide feedback to developers about long-running requests or requests that are using substantial resources, such as returning large rows from a database.

Contact your representative to obtain the Log Analyzer tool. The Log Analyzer is a **.jar** file that can be saved and unjarred in any convenient location. The Log Analyzer expects that the format of DEBS log entries is similar to the following:

```
<YYYY.MM.DD HH:MM:SS:mss ThreadName:LogLevel:LogTag:messages>
```

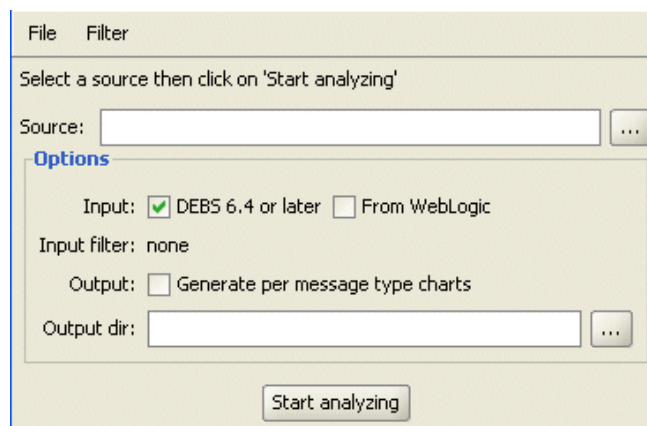
For example:

```
2006.10.12 06:00:00:171 Env/http-8580-Processor48:INFO:WrappingFilter ...
```

Since the process of analyzing a log file can be memory-intensive, specify as much JVM memory as possible to avoid OutOfMemoryError messages. For example, start the Log Analyzer as follows:

```
java -Xms256m -Xmx512m -jar logAnalyzer-1.1.1-SNAPSHOT-app.jar
```

The Log analyzer initial screen is displayed as shown in the following figure:



Enter the following information:

- Source: enter the full pathname of the location of a DEBS log file, or directory containing multiple DEBS log files.
- Input:
 - ⋮ DEBS 6.4 or later is automatically checked. If you are analyzing a log file from a pre-Release 6.4 Visual Modeler, then uncheck this checkbox.
 - ⋮ From WebLogic: click this checkbox to indicate that the log files are generated from a Oracle WebLogic application server.
- Output: click the Output checkbox to generate a response-time chart grouped by message type.
- Output dir: enter the full pathname of the directory to contain the report output.

Click Start analyzing to start the log analysis process. The Log Analyzer displays messages as it progresses, then places the log analysis output in the specified output directory when it finishes.

Setting Up Log Analyzer Daily Reports

This section describes a procedure for automating daily log analyzer report generation. The procedure described here uses Ant, since Ant is portable, well-used, and has good documentation. Ant is available from <http://ant.apache.org>.

The goals of this procedure are to:

- Set up a cron job to run reports nightly, and organize output by date (year/month/day) to ease navigation.
- Compress log files when possible to save space.
- Set up the automation in a way that is easy to duplicate so that log files from multiple deployments can be hosted from a single log server.

To automate log analyzer report generation, you need:

- Java and Ant
- Read access to the DEBS log files
- Write access to the report output directory, *<out.dir>*. The contents of *<out.dir>* are accessible via a Web server.

Daily Reports Workflow

The following describes the general workflow for automating log analyzer daily reports.

1. DEBS generates log files to the application server or servlet container **logs** directory.
For example, the **logs** directory in a Tomcat deployment is *<tomcat-home>\logs*.
The log file is named **debs.log.n**, where *n* is a number. For example, *debs.log.1*, *debs.log.2*, and so on.
2. Set up a cron job to run daily (perhaps very early in the morning) to concatenate all the log files from the log directory into a temp file.
3. From the temp file, extract yesterday's log entries into the log analyzer output directory using the directory naming pattern *year/month/day/log.suffix*.

4. The **year/month/day/log.suffix** file is further compressed using **gzip** to save space.
5. Start the log analyzer to parse the **year/month/day/log.suffix.gz** file and generate the report to the **year/month/day/html/** directory.

Setting Up the Daily Reports

1. If you have not already done so, contact your representative to obtain the following files:

- ⋮ The Log Analyzer **.jar** file
- ⋮ **logAnalyzer-daily.xml**
- ⋮ logAnalyzer-daily.properties

2. Save the Log Analyzer files to a temporary location.
3. See "[Configuration](#)" for information about configuration values.
4. Use the following command to run the daily log analyzer report:

```
ant -Dproperties.file.name="logAnalyzer-daily.properties" -f
logAnalyzer-daily.xml
```

5. Examine the output. The location is similar to the following:

```
sites/default/app-server/logAnalyzer-out.d/dailySplit/YYYY/MM/DD/html/index
.html
```

Recommended directory layout

The following figure illustrates the recommended log analyzer directory layout. This layout is especially recommended if you plan to host log files from multiple sites.

```
# where to keep the log analyzer jar file
bin/
  logAnalyzer-1.1.1-SNAPSHOT-app.jar

# ant script
logAnalyzer-daily.xml

# sites data
sites/
  site1/
    ...
  site2/
    ...
  siten/
    logAnalyzer-daily.properties
    app-server/
      logs/
        debs.log
        debs.log.1
        debs.log.2
```

Site information is kept under the **sites** directory, which contains a directory for each site. The site directory name can be any unique string; the example above uses *site n* , where *n* is a number: site1, site2, and so on.

Each site directory contains a **logAnalyzer-daily.properties** file that contains that site's specific settings.

Each site's log files are kept in the **siten/app-server/logs/** directory.

The **sites** directory is read-only. Output is written to the **siten/app-server/logAnalyzer-out.d** directory.

Using the above layout, you can start a cron job with just the site name. For example, for a site named **bbfb-01**:

```
# Tell Ant to set the site.name and use a build script name:
# logAnalyzer-daily.xml
ant -Dsite.name=bbfb-01 -f logAnalyzer-daily.xml
```

If you rename **logAnalyzer-daily.xml** to **build.xml**, then you can then skip the **-f logAnalyzer-daily.xml** argument. For example, for a site named **bbfb-01**:

```
ant -Dsite.name=bbfb-01
```

Configuration

Deployment-specific settings are set in a property file. The default property file is **sites/\${site.name}/logAnalyzer-daily.properties**. You can also set the property file name at the command line as follows:

```
ant -Dproperties.file.name="path_to_file.properties" ...
```

The following lists the configuration properties in the **logAnalyzer-daily.properties** file.

- **log.dir**: the full path to the location of the directory containing the DEBS log files. For example:

```
# default is ./logs
log.dir=/home/hle/tomcat/logs
```
- **out.dir**: where to write the generated reports. For example:

```
# default is logAnalyzer-out.d
out.dir=/home/hle/public_html/logAnalyzer-out.d
```
- **logAnalyzer.jar**: the location of the logAnalyzer **.jar** file. For example:

```
# default is ./logAnalyzer-1.1.1-SNAPSHOT-app.jar
logAnalyzer.jar=target/logAnalyzer-1.1.1-SNAPSHOT-app.jar
```
- **is.weblogic**: true if the log files was generated by WebLogic. For example:

```
# default is false
is.weblogic=true
```
- **genChart.perMessageType**: false to skip messageType charts generation. For example:

```
# default is true
genChart.perMessageType=false
```
- **log.prefix**: the DEBS log prefix. You rarely have to change this. For example:

```
# default is debs.log
log.prefix=Midwest.log
```
- **target.date.offset**: Auto-set the target.date. The default is 1, which means yesterday. For example, set target.date.offset to 7 to extract log files for a week ago:

```
# default is yesterday: 1
target.date.offset=7
```

- target.date: Limit processing to log entries for this day. The most likely usage for this setting is to manually re-generate an old set of log files. For example:

```
# default is yesterday (auto-evaluated)
```

```
target.date=2006/07/24
```

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual

Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS

FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

J46A/G4

555 Bailey Avenue

San Jose, CA__95141-1003

U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available. This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are

fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© IBM 2011. Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. 2011.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "[Copyright and trademark information](http://www.ibm.com/legal/copytrade.shtml)" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates. Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium and the Ultrium Logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Connect Control Center®, Connect:Direct®, Connect:Enterprise, Gentran®, Gentran:Basic®, Gentran:Control®, Gentran:Director®, Gentran:Plus®, Gentran:Realtime®, Gentran:Server®, Gentran:Viewpoint®, Sterling Commerce™, Sterling Information Broker®, and Sterling Integrator® are trademarks or registered trademarks of Sterling Commerce, Inc., an IBM Company.

Other company, product, and service names may be trademarks or service marks of others.

A

- Analyzing debs.log 32
- Automating Log Analyzer reports 33

B

- Backup and recovery strategies 8
- Backups
 - checkpoint backup 8
 - daily backup 8
 - full 8
 - incremental backup 8
 - weekly backups 8

D

- Data
 - protecting 7
 - timeline for recovery 8
- Deployment architecture 6
 - build environment 6
 - QA area 6
 - staging area 7

I

- Incremental garbage collection
 - Xincgc 30
- Infrastructure
 - application tier 7
 - database tier 7
 - typical 7
 - Web tier 7

J

- JVM
 - server 30
 - verbose
 - gc 30
 - Xincgc 30

K

- Key performance indicators 32

L

- Log Analyzer tool 32
 - directory structure 34
 - properties file 35

M

- Memory allocation
 - areas to check 30
- model size 10

P

- Performance
 - memory issues 29
 - out of memory error 30
- Performance tuning
 - garbage collection 30
- properties 14
 - defining at correct location 26
- property names 14

R

- Recovery policies 8
- Recovery scenario 8
- Redundancy 7

S

- size of models 10
- Strategies
 - backup and recovery 8
- submodels 12

U

Update statistics
Oracle 28
SQL Server 28

X

-Xmx setting 30