Comergent eBusiness System

Release 7.1

Best Practices Guide

Sterling Commerce
An IBM Company

Comergent eBusiness System Best Practices Guide

Documentation part number: 1-7.1-8-02

Preface

Welcome to the Comergent eBusiness System. This *Best Practices Guide* and the associated documentation provides all the information required for you to implement the Comergent eBusiness System at your enterprise.

Purpose

This guide provides a detailed description of how best to implement the Comergent eBusiness System and the key factors that affect a successful deployment.

Audience

This guide presupposes an advanced level of information systems knowledge, familiarity with basic network and database concepts, and Java for certain implementation steps.

Conventions

Throughout this guide, we will use the following conventions shown in Table 1, "Conventions", on page iv:

TABLE 1. Conventions

Type	Convention
File names	**Sample.txt**
Paths and directory names	**/top_level/next_level/next_level/destination_directory/**
Sample code extracts	`public void method(String s)`
Values to be provided	*`<value supplied by developer>`*

Comments

We welcome your feedback. Our aim is to provide our customers with the best quality documentation possible. Let us know about any inaccuracies or missing information in our documentation. We also welcome suggestions for enhancements to our documentation. Our email address is:

`support@comergent.com`

Contents

CHAPTER 1

Introduction

This guide provides a description of the best practices that Comergent Technologies, Inc. development and professional services engineers have identified in implementing the Comergent eBusiness System.

CHAPTER 2, “Security Best Practices”, describes a best practices security model for a Comergent eBusiness System implementation, including role definition and security policies, information assets, and protecting your information assets and critical functions. This chapter also outlines some threat scenarios.

CHAPTER 3, “Backup and Recovery Best Practices”, describes best practices for developing backup and recovery strategies.

CHAPTER 4, “Database Management Best Practices”, describes best practices for managing the data in your database.

CHAPTER 5, “Performance Best Practices”, describes best practices for ensuring the optimal performance of your Comergent eBusiness System, including basic principles, how to isolate performance problems, performing load testing, and using the Log Analyzer tool.

CHAPTER 6, “C3 Configurator Best Practices” describes best practices for designing and implementing the Comergent eBusiness System application model, including project planning considerations, the art of model-making, business rule design, and performance considerations.

CHAPTER 2

Security Best Practices

Introduction

This chapter outlines a Best Practices security model behind the Comergent eBusiness System.

A Web-based application operating as an e-commerce site is likely to be regulated by industry consortiums or must comply with widely-accepted good development practices. In general, most of these situations will involve some guidelines that attempt to not only help develop robust applications, but also provide a relatively secure environment that can adequately safeguard the use of personal or financial information at the site. The range of regulatory or best practice guidelines is fairly extensive. Some provide regulations for which compliance is mandatory for doing business in a particular market; some provide corporate-wide process and procedures that aid in specific goals, such as protecting personal information; some are more focused on good software engineering practices. Some examples include:

- The Open Web Application Security Project (OWASP)

  OWASP is a development community that recommends software best practices.

- ISO 17799 - ISO/IEC Security Standard

  ISO 17799 is a standard establishing guidelines and general principles for initiating, implementing, maintaining, and improving information security management in an organization.
- Payment Card Industry (PCI)

  PCI is an industry consortium that regulates sites that accept and process credit card transactions.
- Sarbanes-Oxley (SOX)

  SOX is a US legal requirement for public corporations.
- American Institute of Certified Public Accountants SAS 70 (AICPA)

  AICPA's Statement on Auditing Standards no. 70 (SAS 70) is an auditing standard by which organizations can assure that they have adequate safeguards and controls in place for hosting or processing data belonging to their customers.

Guidelines and practices included through these and other bodies range from good software engineering practices to building secure networks to corporate accounting practices. In keeping with this environment, this chapter presents the basic building blocks that are likely to be needed to operate a robust and secure site. Not all aspects of all organizations' guidelines will apply to any one site, but it is likely that some of these guidelines could be mandatory. Use this chapter as a starting point in providing a compliant solution, but more in-depth practices may be needed.

This chapter covers:

- "Role Definition and Security Policies" on page 4
- "Information Assets" on page 8
- "Protection Mechanism for Information Assets" on page 11
- "Protection of Critical Functions" on page 12
- "Threat Scenarios" on page 13

Role Definition and Security Policies

Administration Model

This section describes the entities assumed to be present in the administrative domain in which the Comergent eBusiness System resides, including networks, servers, and administrative roles. This is likely not an exhaustive list. It is likely that various network devices will exist within this environment, and perhaps other servers.

Networks

The following network zones are assumed to exist. These networks are connected to themselves as outlined below through gateways.

- External Network: Directly visible from the Internet. Hosts the Web server with static content. The External network is accessible to the Internet through a firewall. It is assumed that the firewall and appropriate standard security practices are sufficient to prevent shell level access from the Internet. The external network has a gateway to the De-Militarized Zone (DMZ) that permits highly controlled access from the Web server to the application server.
- DMZ: This network is not directly visible from the Internet. A constrained gateway permits the Web server residing on the External Network to access the Application server residing on this network and another, similar, gateway permits access from the DMZ network to the Internal Network. Web server routes messages to the application server through dedicated ports.
- Internal Network: Not visible from the Internet, nor from the External Network. Database resources reside here. Application servers in the DMZ connect to Database Servers in this network through a constrained gateway.

Servers in the DMZ and Internal Network spaces have private addresses, complying with RFC 1918. In general, only the minimal access needed to operate the systems should be allowed.

Servers

The following servers are assumed to exist. The term “server” here represents a software application that is more or less continuously listening on one or more network ports, responding to requests received on the ports. Software servers reside on computer hardware. Generally, though not necessarily, there will be a one-to-one relationship between a server software system, and a server hardware entity.

- Web server resides in the External Network. It responds to HTTP and HTTPS requests from the Internet or internal corporate intranets.

- Application server resides in DMZ. Some HTTP and HTTPS requests are delegated to the Application server for dynamically generated response. The Application server maintains connections with the Database server.
- Database server resides in the Internal Network or DMZ.

Roles

This section describes roles within the administrative context of the Comergent System. These are roles assigned to data center personnel acting as employees or agents of the Enterprise. They are distinguished from the roles of individuals who interact directly with the Comergent eBusiness System Web application (online users). Online users have capabilities managed directly by two Comergent System Entitlement services. Dispatch (or “MessageType”) Entitlement service manages page flow privileges. The Access Policy service manages fine-grained data-level access.

- Database Administrator
  - Responsible for Database Servers.
  - Can log into database server.
  - Can read, create, update, or delete databases, database tables, indexes, and other database resources.
  - Can create backups and restore from backups.
  - Can create Database users and manage them.
  - Sets the application user access and authentication.
  - Does not have root level authority in server OS.
  - Does not have direct access to Application Server machine (or network).
  - Does not have Comergent application access
- System Administrator
  - Responsible for Server Hardware, and Server Software.
  - Has root access to server machines within their zone of responsibility.
  - Has the authority to start and stop server processes.
  - As root, can read, write, update, or delete files in file systems.
  - Can back up and restore files. Can create OS-level users and manage them.

    - Does not have access to log in to database server.
    - Deploys the Comergent eBusiness System application in the production infrastructure.
    - Loads the minimal data set in the XML files to the database.
    - Does not have Comergent eBusiness System access.
- Developer
    - Responsible for preparation of deployment Web archives ("WAR" files).
    - Has the authority to create Web archives representing the Comergent eBusiness System executable.
    - Can set properties and business rules governing Comergent eBusiness System operation, including properties that configure access to the database, properties that configure the JCE Key store, and so on.
    - Has the authority to create or modify the initial Comergent eBusiness System dataset. This dataset is a part of the deployment archive.
    - Does not have any kind of access to the Production Database or Application servers.
    - Does not have username/password to access production Comergent eBusiness System.
    - Does not move code from development/QA environments to production.
- Network Administrator
    - Configures and manages network.
    - Has authority to create and assign network resources, including domain names, IP addresses, firewall level, and so on.
    - Does not have database access.
    - Does not have Comergent eBusiness System application access.

Data Center Roles

The following are assumed about Data Center administrative roles:

- Roles are segregated. System Administrators cannot be Developers, Network Administrators, nor Database Administrators. Database Administrators cannot be Developers, System Administrators, nor Network Administrators, and so on.

- System Administrator Roles are partitioned on network boundaries. A system administrator for the DMZ should not be a system administrator for the Internal Network.
- Data Center Administrators do not have Comergent eBusiness System online administration roles.

Information Assets

Encryption of Persistent Data

This section describes the encryption of persistent data in the Comergent eBusiness System.

1. What data is encrypted?

This is configurable within the Comergent eBusiness System configuration files. Please see other sections for the details of which data fields and the contents of which data fields are encrypted.

2. What encryption libraries are used?

By default, the Sun implementation of JCE is used for encryption, with policy files enabling strong encryption. This is the recommended option.

3. What encryption algorithms are used?

This is configurable among secret key encryption algorithms supported by JCE. The recommended algorithm is AES. The Comergent eBusiness System also supports 168 bit DES encryption.

4. Where are keys stored?

In a password protected JCE Key Store file. By default this file is located in the home directory of the user that initiates the Comergent eBusiness System application, but this can be altered within a configuration file.

5. How is the Key Store password protected?

By default the key store password is encoded in Java. The Comergent eBusiness System supports two phase initialization, if desired, in order to employ a password provided at run time. In this mode of operation, the Comergent eBusiness System will come up partially and await a special HTTPS message containing the key store and key passwords to complete initialization and respond to requests. This technique means that the application server system administrator will not have access to the Comergent eBusiness System password, but it requires more than one person to bring up the Comergent eBusiness System.

6. How are passwords saved within the Comergent eBusiness System?

Either by a cryptographically secure digest (MD5 or SHA-1) or by secret key encryption as described above.

Information Assets

This section introduces the critical information assets that are dealt with by the Comergent eBusiness System.

Account profile

The user's account information is maintained in the Account profile. The user's account is identified by the (unique) account-id. The account profile contains personal financial details such as credit card numbers and BillTo and ShipTo addresses. Implementation can be customized, so this profile may contain additional or fewer details, encrypted fields, and so on.

Transaction and System Log

According to the type of data and level of role separation required, Comergent eBusiness System logging can be configured to capture all transactional and non-transactional requests to file locations outside of the Comergent eBusiness System deployment infrastructure. Certain aspects of order and account history can also be captured in the Comergent eBusiness System database for administrative and end user interaction purposes but do not represent a secure audit trail.

Network considerations need to be included for remote and/or more secure logging.

Key Store File

The Key Store file stores the encryption keys that protect Comergent eBusiness System data. This file normally resides in the home directory of the user that initiates the Comergent eBusiness System Web application, but this can be altered within a configuration file.

Attention: The Key Store file is a requisite to encrypt or decrypt data in the knowledgebase. This Key Store file must be well protected. With this Key Store file, the encrypted data can be read, so it must never be included on the same media as a database backup. If the Key Store file is lost, then data cannot be read from the database.

WAR local key store file

The WAR local key store file stores the keys that protect the database username and password used by the Comergent eBusiness System. It is contained in the Comergent eBusiness System WAR file structure.

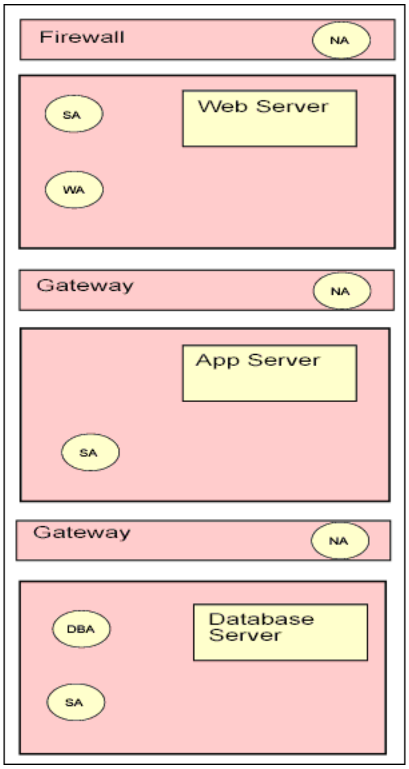
User passwords

Comergent user names, passwords, and user profile information are stored in the user authentication and user contact tables. The passwords are protected as described in “Protection Mechanism for Information Assets” on page 11.

Roles Schematic

The following indicates the spheres of access for the following classes of administrators:

- NA: Network Administrators
- SA: Security Administrators
- WA: Web Administrators
- SA: System Administrators
- DBA: Database Administrators



Protection Mechanism for Information Assets

This section discusses the encryption mechanisms used to protect the integrity of the critical data fields of the Comergent eBusiness System.

Credit Card Information

The following data fields are encrypted using the DES 168 bit encryption by default:

- Credit Card Number
- Credit Card Holder Name

User Passwords

User passwords are protected by 1-way encryption (MD5 or SHA-1 Message Digest).

Only the digests of the passwords are stored in the Comergent eBusiness System database. Alternatively, at the discretion of the customer, the password can be encrypted with DES 168 bit encryption or other encryption.

Protection of Critical Functions

Performing functions such as creating and setting database users and passwords and storing sensitive data in a database are critical from both a security and an operational perspective. Equally critical is protecting the performance of these functions from potential attacks or misuse. This section documents the process of performing these functions and explains how that process prevents attacks or misuse.

Setting Application's Database User and Password

1. The DBA creates a user in the database, assigns a password, and sets appropriate database privileges.
2. The DBA informs the developer.
3. The developer encrypts the password using the key in the WAR local Key Store.
4. The developer initializes connection information in **WEB-INF/schema/DataSource.xml**.
5. The developer prepares the WAR file and gives it to the application server SA.
6. The application server SA deploys the WAR file
7. The application decrypts the password and uses it to connect to the database.

Assertions

- The developer knows the password but does not have access to the database, so cannot use the password.
- The application server SA does not know the password, so cannot access the database.

- The DBA has the password and can access data, but does not have access to the Web application. Sensitive data is encrypted. The DBA does not have access to the system without having access to encryption keys.

Storing Sensitive Data in the Database

1. The developer specifies encrypted fields for sensitive data in the database schema.
2. Where necessary, schema fields are redefined to incorporate correlative data to prevent relocation. For example, account balances may be converted strings combined with an account id to provide the ability to detect field copies.
3. The developer specifies the encryption algorithm and location and type of key store.
4. The developer specifies the key store password.
5. The developer prepares the WAR file and delivers it to the application server SA.
6. The application server SA deploys the WAR file and starts the application server.
7. When sensitive data fields are set, data services invoke the encryption service. Data is encrypted. Eventually, data is persisted to the database.

Assertions

- The DBA has access to the database, but does not have the ability to decrypt sensitive fields.
- The application server SA has access to the key store, but not to the key store password.
- The developer has access to the key store password, but not to the key store.

Threat Scenarios

Transport

Scenario: The Network Administrator intercepts clear text values in communication between the application server and database server, for example, over a JDBC transport stream.

Preventative Measures: Sensitive data is encrypted before transport and would not be usable. The encryption key is never transported.

Restores from Backup

Scenario: The DBA restores database tables from backup to some earlier state in order to create a vulnerability.

Preventative Measures: The Comergent eBusiness System provides no means to prevent this type of attack. Rely upon Data Center practices and procedures.

Detection: It is theoretically possible to detect this attack by correlation to Comergent eBusiness System debug logs and Web server access logs.

Log Files

Scenario: The application server SA edits the Comergent eBusiness System debug log files to conceal some activity. Comergent recommends that operational systems configure debug logging at the “INFO” level. At this level, Web request start and end events, JSP dispatch events, session events, login events, and CRON events are logged. Scenarios such as an attack that could be concealed therefore involve an attack by a logged in and authorized user.

Preventative Measures: The application server SA has root access to the file system. One can not prevent this sort of activity.

Bogus Account to Access Customer Records

Scenario: The Comergent eBusiness System administrator creates a super CSR who has access to all accounts. The Comergent eBusiness System administrator uses super-CSR to get credit card info, change balances, charge customer credit cards, and so on.

Preventative Measures: CSRs should not be able to see the unmasked or unencrypted credit card numbers. Any activity that involves charges against the valid customer credit cards will be logged to the account transaction log.

Detection: All account activities are logged in the account transactions log table. Unusual activity from a specific CSR can be detected by reviewing the logs in a timely manner.

Credit Card Number Theft

Scenario: A CSR or CSR manager gets a customer’s credit card number(s) and commits fraud.

Preventative Measures: When a customer's credit card number is displayed to the CSR, it is masked so that only the last four digits are displayed, and the Card Verification Value (CVV) number is not stored in the database or log files.

CSR's only see the entire credit card number and associated information when they create the account or edit the credit card information on behalf of the customer. There is no practical way to prevent the CSR from stealing this information.

DBA Password Theft

Scenario: A System Administrator obtains the DBA password and performs unauthorized activities on the database.

Preventative Measures: Use best practices rules for password creation and appropriate policies to expire and rotate the passwords periodically. As the Comergent eBusiness System money fields are encrypted using keys and combinational fields, the DBAs will not be able to decrypt and modify the values.

Detection: Review of the database level audit files can detect such activities.

CHAPTER 3

Backup and Recovery Best Practices

Introduction

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 chapter covers:

- "Deployment Architecture Overview" on page 18
- "Infrastructure" on page 18
- "Backup Strategies" on page 20

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.

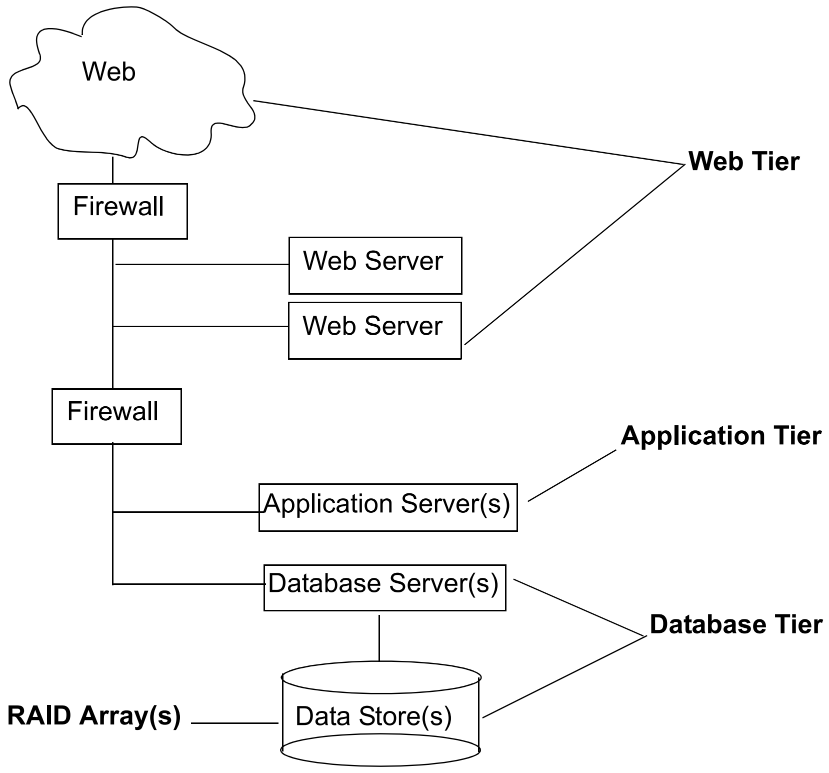


FIGURE 1. Typical Deployment Infrastructure

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. See CHAPTER 2, “Security Best Practices” for more information about 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.

CHAPTER 4

Database Management Best Practices

Introduction

This chapter describes database management best practices and the activities that support those practices. Database management practices protect your data, ensure data integrity, and address database performance issues. Database management activities include:

- Archiving active data
- Monitoring database tables
- Sizing database tables and purging old data at intervals dictated by business requirements or regulatory audit compliance, such as Sarbanes-Oxley or HIPAA
- Updating indexes and managing database statistics

For recommended database archival practices, see CHAPTER 3, “Backup and Recovery Best Practices”.

This chapter covers:

- “Archiving Data” on page 24 presents general guidelines for archiving data.

- "Monitoring Database Tables" on page 24 presents guidelines for monitoring and sizing database tables, key tables to monitor, guidelines for purging data, and creating and using history tables to retain old data online in compliance with regulatory or business requirements.
- "Updating Statistics" on page 26 presents scripts for updating the database statistics for an Oracle and a SQL Server database.

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). See CHAPTER 3, "Backup and Recovery Best Practices" for more information about database backup strategies.

Monitoring Database Tables

Auditing compliance requirements such as Sarbanes-Oxley and HIPAA may dictate how long you must store production data online. In general, performance starts to degrade when the number of rows in a database table grows to between 50 million and 100 million rows. To avoid performance problems, monitor the size of your database tables and start transferring data to history tables when the number of rows in a particular table reaches about one million rows. Data that is no longer needed by your application, such as data that is over one year old, should be considered historical data and transferred to history tables. History tables are tables that are online but that are not accessed by the Comergent eBusiness System application and therefore do not affect performance.

Key Tables To Monitor

While you should monitor the size of all the Comergent eBusiness System database tables (including custom tables added during your implementation), the following tables tend to grow quickly and should be monitored on a daily basis.

CMGT_CART_CONFIGURATION

CMGT_CART_CONFIGURATION_LINES

CMGT_CART_LINES

CMGT_CATALOG_TO_CART_LOG

CMGT_CATALOG_TO_CONFIGURE_LOG

CMGT_NOTE

CMGT_OIL

CMGT_OIL_HEADER

CMGT_OIL_LI

CMGT_ORDER_ADDRESSES

CMGT_ORDER_EXTN

CMGT_ORDER_LI_EXTN

CMGT_ORDER_LI_SHIP

CMGT_ORDER_SERIAL_ITEMS

CMGT_QUOTE_EXTN

CMGT_RFQ_EXTN

CMGT_RFQ_LI_EXTN

Purging Data

Business requirements and regulatory compliance requirements will dictate how long you must retain data, whether online or in offline archives. Establish notification protocols and procedures to handle purging data appropriately for your requirements.

Creating and Using History Tables

The names of history tables are appended with _H. See “Key Tables To Monitor” on page 24 for more information about the key database tables to monitor.

Some history tables are created during the Comergent eBusiness System implementation, such as CMGT_INVOICE_LI_H, CMGT_OIL_LI_H, and CMGT_ORDER_LI_EXTN_H. If you need to trim the size of a database table that does not already have an associated history table, then you can create one. To transfer historical data to a history table:

1. Create a new table with exactly the same structure as the table whose size you need to trim. The name must be of the following format:

   *TABLE_NAME*_H

   TABLE_NAME is the name of the table to contain the historical data and _H indicates that this is a history table, not to be used in indexing and not to be accessed by the Comergent eBusiness System application. The name should be similar to the name of the original table. For example, the history table CMGT_INVOICE_LI_H contains historical data for the CMGT_INVOICE_LINES transactional table.
2. Transfer the historical data to the history table from the original (transactional) table, using business criteria to determine which data is no longer needed. For example, you might place data that is more than a year old into a history table.

The following sample SQL statement transfers historical data to a history table:

```
INSERT INTO CMGT_TABLE_NAME_H SELECT * FROM CMGT_TABLE_NAME WHERE
ACTIVE_FLAG = ‘N’;
```

You can also use the DBMS export command, import the data to the appropriate history table, and then delete the transferred data from the original table.

The following sample SQL statement deletes transferred data from the original table:

```
DELETE FROM CMGT_TABLE_NAME WHERE ACTIVE_FLAG = ‘N’
```

You can still see and access any history tables if necessary by doing SQL queries on the database. When the data ages past its required retention time, you can use standard tools to purge it. If you do not have auditing compliance requirements, other business decisions may determine when you should purge your data.

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_STATUS.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 }
```

CHAPTER 5

Performance Best Practices

Introduction

This chapter describes performance optimization best practices. It discusses the three primary questions relevant to project implementation:

- How do you build a performance-optimized system?
- How do you verify what you have built?
- How do you solve problems that arise?

This chapter covers the following topics:

- "Performance Concepts" on page 29
- "Building a Performance-Optimized System" on page 30
- "Verifying Performance" on page 32
- "Solving Performance Problems" on page 39

Performance Concepts

This section describes performance optimization concepts. Subsequent sections go into these concepts in more detail.

1. Set performance as the number one priority from the very beginning of the implementation process.
2. Ensure that your infrastructure is sound. This will help in isolating problem areas when your project enters the testing phase, and if problems arise after your project goes live.
3. Know your system components. Take an inventory of all components that make up your system, check that all components are functioning properly, and keep the inventory up-to-date.
4. Establish a performance baseline.

   Establishing a performance baseline and taking regular baseline measurements allows troubleshooters to return to the last known good state and to measure the effect that each component has on overall system performance.
5. Set realistic performance expectations.

   Setting realistic performance expectations allows you to recognize when performance tuning has been successful.

There are two types of performance optimization:

- On-going: an established, daily routine of monitoring and recording your deployment's performance.
- Troubleshooting: identifying and resolving problems that affect the ability to conduct business on your e-commerce site.

As the system is deployed and continues to evolve, monitor and verify its performance against the goals set during the implementation process. Establishing an on-going performance optimization routine allows you to avoid troubleshooting situations while providing a way to easily track changes in performance and make improvements as needed. Use a log analysis tool such as the one described in "Log Analyzer Tool" on page 69 to measure system performance and to assist in performance optimization and troubleshooting.

See "Solving Performance Problems" on page 39 for information about troubleshooting performance problems.

Building a Performance-Optimized System

Ensuring that your infrastructure is sound, knowing your system components, and taking baseline performance measurements all factor into building a performance-optimized system. You should also have an end-game plan. What are the expectations for performance metrics? Are the expectations reasonable?

Performance Is the Priority

Building in performance from the very beginning is preferable to, and more efficient than, troubleshooting performance problems down the line. Before you add a component to the system, take baseline performance readings. After you add the component, check the component's effect on overall performance. Examine data such as the number of SQL queries, total time spent servicing requests, how many times requests go to the database or introduce client-side redirects, and compare with the baseline. Proceeding in this way allows you to measure the effect that a particular component has on the overall system.

Ensuring a Sound Infrastructure

Ensuring that your infrastructure is sound before you start a Comergent eBusiness System implementation assures that you are building on a solid foundation.

Taking Inventory of System Components

Take an inventory of your system's components and keep the inventory up to date. Doing this will help in identifying performance bottlenecks.

A system inventory includes:

- An up-to-date network diagram.
- An inventory (even if it is hand-drawn or just a list) of the Web server, Web browsers, Web containers, application server, database, router, firewalls, and a list of the operating system, CPU types, available memory, and so on. Such an inventory will be useful when you are evaluating possible bottlenecks.
- Note applications, including the version number and any installed patches.
- Note your network bandwidth.
- Ensure that integration points are configured correctly. For example, ensure that third-party applications such as a pricing or shipping application or web services such as a geocode lookup service, are configured correctly and functioning properly.
- Having a current list of contacts who are responsible for supporting each component of your system and keeping the list up to date.

Establishing a Performance Baseline

The purpose of taking baseline measurements is twofold:

- To allow troubleshooters to return to the last known good state of the system.
- To isolate the performance impact of each system component and account for how that component contributes to overall performance.

Taking baseline performance measurements is not a one-time activity, performed only before you start implementing a system: it is also part of daily performance monitoring activities. Establishing a baseline performance record and monitoring performance on a daily basis allows you to identify and resolve trouble spots early.

Use a log analysis tool to assist in monitoring performance and use the tool at all phases of your implementation: to take baseline performance measurements, to verify your built system's performance, as part of monitoring performance on a daily basis, and for troubleshooting. For example, use the log analysis tool to generate daily reports, then compare the reports day by day. Log analysis tools capture exceptions which you can then use as a to-do list to resolve performance issues. The daily reports are useful during troubleshooting: if you keep a daily log with running commentary, confirming the problem and identifying the bottleneck is easier in a troubleshooting situation.

Setting Realistic Performance Expectations

Taking baseline performance measurements allows you to set realistic performance goals that are achievable given the system infrastructure and its components.

For example, taking baseline measurements of an existing system, excluding any Comergent eBusiness System components, will assist in determining realistic post-Comergent-deployment performance goals. If response time is one second without Comergent, it would be unrealistic to expect that response time will remain one second once Comergent is deployed.

Verifying Performance

Once you build your system, verify its performance and do any needed performance optimization tuning to bring performance in line with your goals. Load testing verifies system performance under load. Use a log analysis tool to measure system performance and to assist in performance optimization.

Load Testing

Load testing is the process by which you measure how the implementation performs under load and determine whether the production implementation will meet its performance requirements.

Load tests should be repeatable: that is, you should document your starting point sufficiently to allow re-creating the test, either in the same environment or in a different one. Make a complete copy of your last good environment, from the Web server to the database image. This ensures that you can fall back to the last known test run that had good results, and that you have something reliable to compare against to check your progress.

Use techniques such as stubbing and simplifying to troubleshoot problems. Stubbing or simplifying execution can provide good insight into load issues. Stubbing is particularly useful if you are using a third-party service, such as SOAP calls to a map service, and you need to quantify its contribution. See “Using Stubs to Isolate Performance Issues” on page 38 for more information.

You should also:

- Have a network diagram of the testing infrastructure.
- Know the single-user timing before you allow multi-user load testing. If you don't know the single-user timing for a particular test scenario, find out. If the single-user timing is already bad (with some requests taking more than a second), do not proceed to multi-user load testing.
- Do not allow other external processes to run during load testing. If the load test result is bad, how do you isolate the cause of the problem, whether the cause is your Comergent deployment, or other processes taking memory or CPU cycles?
- Start in a clean state. For example, a testing scenario progressively adds items to a user cart. Subsequent logins to display the carts will get progressively long. Try to re-set the database if you can. Restart the Web and application server before each load test if you can.
- Always archive good results and establish your baseline values. Once you are getting good results, archive and save everything that is needed to replicate the same environment. Doing so will come in handy if you need to go back at some point, for example if there are changes to your environment and your load testing results are now bad.
- Start small. Too often, the load tester wants to start at the end: the so-called most realistic scenario (which is often the complicated scenario). It

is best to start simple and build up. For example, try the following progression:

- Only static pages
- Dynamic pages: display only the home page
- Simple scenario: Log in, then log out
- Read-only: Log in, display an order, then log out
- Simple configuration: as an anonymous user (or a test user), search a configurable item, configure the item, but don't add the item to a cart

If all of the above items perform well, then try one of the more complicated scenarios.

- Know the effects of third-party software on your deployment. If third-party integration is involved, test it separately to prove that it functions correctly outside of your Comergent eBusiness System deployment. At a minimum, use the stubbing technique to bypass the third-party software and eliminate any effects it might have on your deployment's performance.
- Be prepared to collect sufficient data while the load test is running. To put the load test results into context, you need to know the following:
  - CPU usage on all servers (Web, application, database)
  - Web server log (for example, the mod_jk log for Tomcat)
  - Application server log
  - Network interface statistics, such as:

  ```
  netstat -i
  ```

Preparation

Before you start load testing, consider the following:

1. Is the application server configured to accept a reasonable number of incoming requests? For example:
   - For WebLogic, check the work thread settings. For information about WebLogic settings, see:

   ```
   http://e-docs.bea.com/wls/docs81/perform/WLSTuning.html
   ```

   - For Tomcat, check the value in conf/server.xml for maxThreads:

```
<Connector port="8580" maxHttpHeaderSize="8192"
    maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
    enableLookups="false" redirectPort="8443" acceptCount="100"
    connectionTimeout="20000" disableUploadTimeout="true" />
```

2. Check your test script scenario. Is the think-time reasonable? (3-7 seconds for the extreme case; 7-15 seconds is more reasonable.)
3. Are you testing using one user? Multiple users?
4. Is your environment in a clean state before you start the test? Remember to reset the database after running a test. For example, if you use one user to create 1000 carts, be sure to reset your environment before you re-run the test.
5. Where is the client in relation to the server? For example, running the test from home using a DSL line is not optimal; the line is unlikely to be able to handle large amounts of transmitted data.
6. What is the network diagram? Do you know all the touch points? Is there a proxy server somewhere that might limit the number of requests that the network can handle?
7. While the test is running, have someone monitor the application server: CPU, threads, memory usage, and (for WebLogic), the work queue.
8. Is there anything else running on this application server instance at around that time (other than the Web application) that might affect such resources as CPU cycles or threads?
9. If the results do not make sense, then try a simpler case. For example, have the test client retrieve static HTML pages. This will remove the Comergent implementation from the equation. When you do that, check whether or not you see the same performance degradation.
10. Do you need to monitor the database server as well? Is the database loaded? Is the CPU okay during the test?
11. If the Web application uses a third-party service (mapping, pricing, credit card processing, and so on), ensure that this is not the choke point.
12. Has this infrastructure been load tested before for other Web applications? How was that experience?
13. What are the JVM memory settings?
14. Is the problem area using out-of-the-box code or custom code?

Basic Sequence of Testing

The following is a sample load testing check list for a customer using WebLogic and a load testing tool on Linux and Windows.

Before the Load Test Run

1. Capture machine information, including CPU, memory, JDK, and JVM settings:

    - On Windows, retrieve CPU information using this tool:

    ```
    http://www.cpuid.com/cpuz.php
    ```

    - On Linux, retrieve CPU information using the following command:

    ```
    cat /proc/cpuinfo
    ```

2. Synchronize the machine clocks among the application server, database server, and load testing tool controller. Synchronization does not have to be exact but it will help later to correlate events if the clocks are close.
3. Construct a simple network diagram of where each of the servers are: load testing tool controller, Web server (if any), application server, and database server. Note any possible choke points, such as firewalls or a small pipe (for example, a DSL line from home).Use the ping command to find out how many network hops there are between any two servers.
4. Everyone should know and understand the load testing scenario. For example, a simple scenario might be login, browse the product catalog, and configure model XYZ. One should be able to duplicate the same scenario using a Web browser in a single user test.
5. Use a Web browser to go manually through the test scenario once. Make sure that there are no errors both in the Web browser and in the Comergent eBusiness System application log file. Make notes of elapsed times for each click. This can be used later as one of the baselines.

    The following are tools that can help tracking elapsed times as well as other statistics such as number of requests per click and response size:

    - For FireFox: https://addons.mozilla.org/firefox/1743/
    - Use a proxy: Contact Comergent Technologies, Inc. for information about using an HTTP proxy tool.

6. You will need access to all servers so that CPU usage can be monitored while load testing is running.

During the Load Test Run

During the run, perform the following tasks:

1. Connect to each server: Web server, application server, database server, and load testing tool controller. Use appropriate tools, such as TaskManager, to keep an eye on CPU usage on all layers: Web, application, and database, while the load test is running.
2. Using the load testing tool, run the test once for one user. Make sure there are no errors both in the load testing tool and from the log file. Compare the timing results against the manual Web browser case. If the results are significantly different, then you need to reconcile them before running a multi-user load test.
3. For each load test run, note:
   - start/end time
   - number of vusers (virtual users)
   - startup delay (starting N users for M seconds)
4. Make sure that log files are preserved if the server is restarted between runs.

After the Load Test Run

Consider the following questions as you evaluate the test results.

1. Is customer timing matched up with your log timing?

   Use a log analysis tool to determine whether the customer timing matches the log timing.
2. What is the incoming request rate? Use the application server access log to estimate the number of incoming requests per second/minute. Is the rate reasonable? Ask the Web administrator if the infrastructure is capable of handling that rate.
3. What if you use jLoad to test? Do you see the same degradation?
4. Is this slowdown across the board or occurring in just a few cases?
5. Is the page in question large? Does it use a large number of SQL persists or restores? Check the log analysis report to be sure.
6. Forward the load test report and other information such as start/end time, number of vusers, and startup delay to Comergent Technologies.
7. Forward the Comergent eBusiness System logs to Comergent Technologies for log analysis. Include information about CPU usage (for example, 10% usage

on the load test controller, 80-90% CPU on application server, 30% on database server).

Ensuring Load Test Repeatability

You may need to re-create your performance testing in either the same environment or in a different one. Make sure that you establish a base camp: this is a complete copy of your last good environment, from the Web server to the database image. Having a base camp ensures that you can fall back to the last known run that had good results, and that you have something reliable to compare against to see if you are making progress.

Establish your base camp as soon as you have a good load-testing run. Archive everything that will be needed to replicate the same environment. At a minimum, save the Web application as follows:

- The Web server. The easiest method is to jar up the whole Web server directory.
- The Application server. The easiest method is to jar up the whole application server directory.
- Save a binary database image and verify that you can restore the database from it.

Database Tools

If you are using an Oracle database, then familiarize yourself with tkprof. Contact Comergent Technologies to obtain a tool that can be used to display tkprof output.

Log Analysis

Use a log analysis tool to get independent confirmation from your load test results. For example, if the load test results on the client side say that the average response time is 20 seconds, are you seeing the same results on the server side from the log analysis report? See “Log Analyzer Tool” on page 69 for more information about using a log analysis tool.

Using Stubs to Isolate Performance Issues

Remember the power of stubbing and simplifying. When things are bad, stubbing or simplifying execution can provide good insight into load issues. Stubbing is particularly useful if you are using a third-party service, such as SOAP calls to a map service, and you need to quantify its contribution.

For example:

- If you suspect that the pricing is taking too much time, add a stub to always return 0.00.
- If a particular JSP is taking too long, replace the JSP with one that simply displays "Hello world".
- Move the load testing driver to the same server as the Web or application server to reduce network traffic.
- Hitting the application server build-in listener directly instead of the Web server to remove the Web server contribution.

Network Considerations

It is possible that network issues between one or more of the layers are contributing to performance issues. Contact Comergent Technologies, Inc. to obtain information about performing sanity checks and using the HTTP 1.1 expiration and response compression facilities to improve performance.

For example, try this approach to test for network latency:

1. Run the log analysis tool to confirm system slowness. Compare the timing record in the log file against what users claim to experience.
2. Take stock of the internal network architecture, including the proxy server, firewall, and virus scanning on the firewall, proxy, and on the user desktop. This will help account for other possible contributing factors.
3. Take the basic network transfer measurement.

   Contact Comergent Technologies, Inc. for more information about taking the basic network transfer measurements.
4. In general, moving to an Apache/Tomcat stack can help because you can implement some additional optimizations such as compression and expiration.

   Contact Comergent Technologies, Inc. for more information about implementing additional optimizations.

These optimizations can reduce the number of requests/bytes transferred and thus will speed up response time.

Solving Performance Problems

If the ability to conduct business on your e-commerce site is at risk or severely impaired, you must first confirm the problem, perform triage to determine the nature of the problem, and then resolve the problem as quickly and efficiently as possible.

The following lists the steps to take during the problem analysis process:

- Use a log analysis tool to confirm the problem.
- Establish the priority of the problem.
- Identify the last known acceptable checkpoint.
- Identify the bottleneck, formulate a modification plan, and establish a fall-back procedure.
- Evaluate the solution risk vs. reward.
- Provide feedback to Support.

Having a baseline is an important tool for successfully identifying the cause of performance problems. The baseline also helps you to determine when the system last performed acceptably, and when the slowdown started.

The following sections describe the problem analysis process.

Confirming the Problem

To confirm the problem, look for the following types of indicators:

- A spike in response time. The response time chart provided by log analysis tools gives an overview of your system's responsiveness. Check for sudden increases in response times.
- An exception list. Most log analysis tools automatically compile a list of exceptions. Check for such lists and examine their timestamps to determine when the problem started.
- Bad log entries. Were there bad log entries during a particular period of time?

Is This a New Problem?

This step is where an established baseline becomes critical. Having the baseline allows you to more readily correlate problems with specific events, such as software updates or an increase in requests.

To determine whether or not this problem is new, do the following:

- Compare the current log analysis report with baseline report(s) from the same time period yesterday, last week, or even last month as needed.

- Identify when the degrade in response time started and note performance indicators during that time. Did memory use spike? Was there a spike in requests?

Using Performance Metrics

Certain metrics can help you to compare current performance with baseline values, and to help resolve the problem.

The following are some metrics you can use to compare current performance with baseline values.

- Total number of requests: was there any change in load?
- Response time: what is the percentage of requests under *n* seconds?
- Request load: was there a change in the total number of concurrent requests?
- Session load.

The following are some metrics you can use to help resolve the problem.

- Memory usage: note consistently high memory usage, large swings in memory usage, and whether the system is operating near the maximum allocated memory.
- Thread usage: high thread usage is a candidate for further examination.
- Request load: is the request load higher than expected?
- Note the worst offenders: which requests have the highest response times, use the most resources, or perform unusually high numbers of inquiries to the database?
- Response time per message type: is the problem specific to a particular message type or to a small set of message types?
- Hanging requests: requests hanging or timing out are candidates for further examination.

Troubleshooting Check-List

- Use a log analysis tool to confirm the problem and identify the bottleneck. A daily log report with running commentary is useful for this part of the process.
- Look for out of range values, including large numbers of SQL statements, HTTP requests, or large datasets.

- Perform a quick sanity check of all your deployment layers and components, including the Web browser, network bandwidth, front-end Web server, application server, and database.
- Check your resource usage: CPU, memory, and threads. If your system is hanging, do a Java thread dump.
- If you have third-party software, check for new **.jar** files in **WEB-INF/lib/**. Ensure that any third-party software is performing correctly.
- Change one variable at a time.
- Try simplified, stand-alone (non-DEBS) tests to obtain the lowest response-time number.
- Communicate: let everyone know what you are doing so that you do not duplicate work.
- Before applying a change, consider whether it is worth the risk. Always have a fall-back plan.

CHAPTER 6

C3 Configurator Best Practices

Introduction

Mapping eBusiness requirements to the features provided by Comergent eBusiness System's C3 Configurator can be daunting. Understanding how C3 Configurator works in a variety of circumstances and studying examples demonstrating common usage patterns can help.

This chapter covers:

- "Planning Considerations" on page 44 describes planning considerations for designing and implementing a Comergent eBusiness System application model.
- "Models" on page 45 provides an overview of models and best practice design principles.
- "Properties" on page 50 provides tips and procedures for working with properties.
- "Rules" on page 56 describes how to define and use rules to promote efficiency in your model.
- "Performance" on page 63 describes considerations for writing models that provide the best performance.

About Absolute and Relative Paths

This chapter 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 product. When you sit down to plan a Comergent eBusiness System implementation, you start by considering how to design a model of the product. There is no one “right” way to model a particular product. 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.

- 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 product 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 chapter 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.

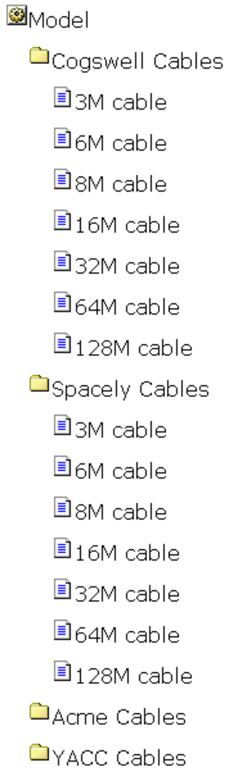
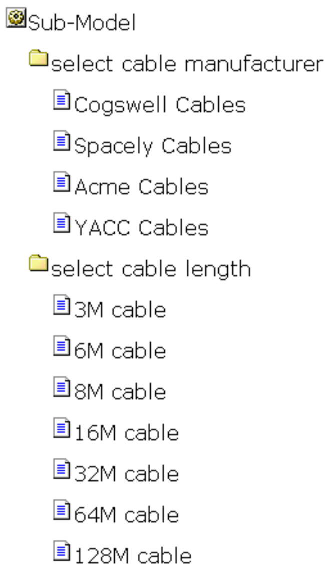


FIGURE 2. **Cable Inventory Model by Supplier**

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.

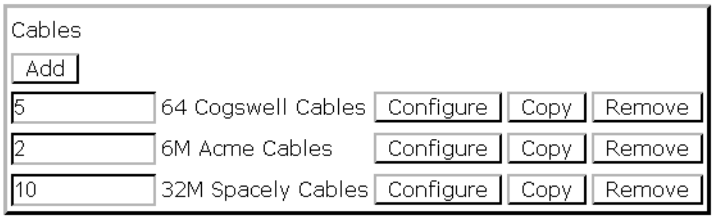
A better alternative is to create a submodel that allows the user to select a cable manufacturer and length, then use dynamic instantiation to let the user add as many different cable types and lengths as necessary, as shown in the following figure.



Sub-Model
- select cable manufacturer
  - Cogswell Cables
  - Spacely Cables
  - Acme Cables
  - YACC Cables
- select cable length
  - 3M cable
  - 6M cable
  - 8M cable
  - 16M cable
  - 32M cable
  - 64M cable
  - 128M cable

FIGURE 3. **Dynamic Cable Inventory Model**

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



Cables

Add

5	64 Cogswell Cables	Configure	Copy	Remove
2	6M Acme Cables	Configure	Copy	Remove
10	32M Spacely Cables	Configure	Copy	Remove

FIGURE 4. **Cable Selection Screen**

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.

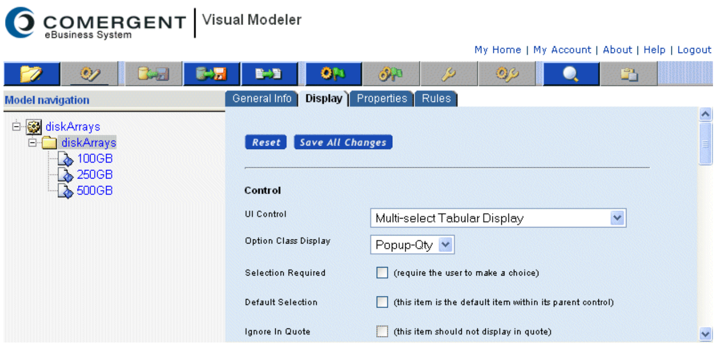
Some modelers do not like the placement of the quantity box, so instead use the User Entered Values (UEV) control to display an edit field next to the item in which users can enter a quantity. The problem is that the behavior of the popup-qty control differs significantly from the behavior of UEV controls: 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 Edit Model icon. The Model navigation page appears. 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.



Control

UI Control: Multi-select Tabular Display

Option Class Display: Popup-Qty

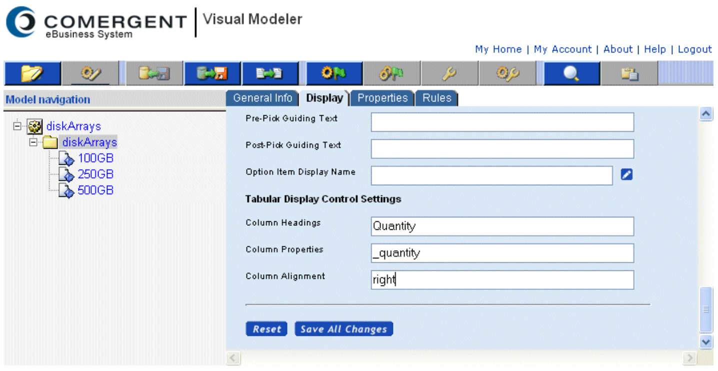
Selection Required: (require the user to make a choice)

Default Selection: (this item is the default item within its parent control)

Ignore In Quote: (this item should not display in quote)

FIGURE 5. **Setting up a popup-qty control: selecting Multi-select Tabular Display**

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



Pre-Pick Guiding Text:

Post-Pick Guiding Text:

Option Item Display Name:

Tabular Display Control Settings

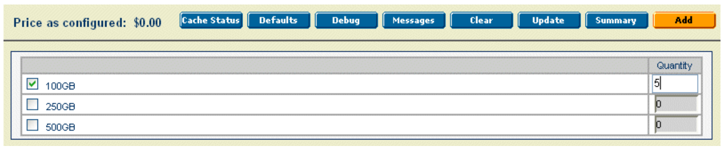
Column Headings: Quantity

Column Properties: _quantity

Column Alignment: right

FIGURE 6. **Sample Column Headings, Properties, and Alignment Settings**

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



	Quantity
✓ 100GB	5
250GB	0
500GB	0

FIGURE 7. **A Sample Product Configurator Page Showing popup-qty Controls**

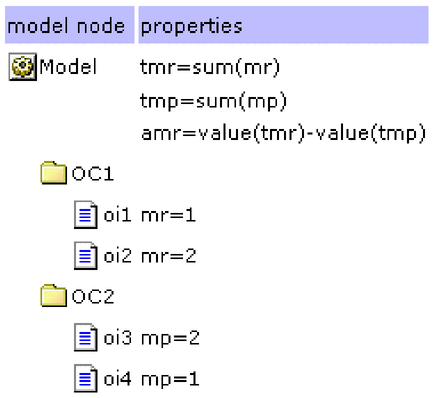
Properties

Properties are ubiquitous in the ***C3*** 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 product, 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:



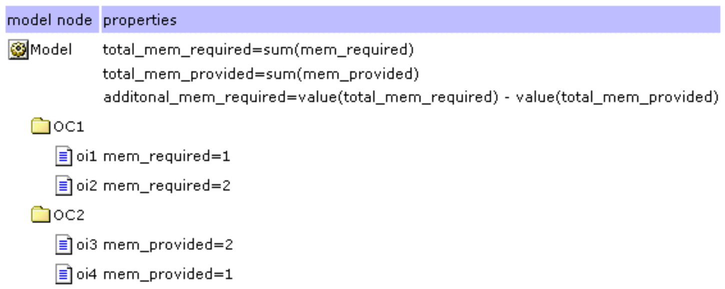
model node	properties

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

FIGURE 8. **Using Cryptic Property Names**

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)
	additonal_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

FIGURE 9. **Using Meaningful Property Names**

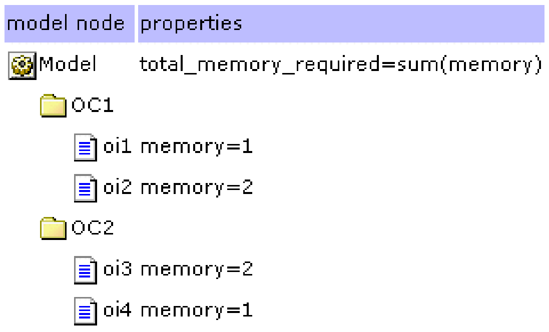
Do Not Use the Same Property to Mean Two Different Things

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 is 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

FIGURE 10. Using One Property Name For Different Purposes

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 page 56 on why using paths to specific instances of properties can be a bad idea.)

Define Properties at the Appropriate Level in the Model Hierarchy

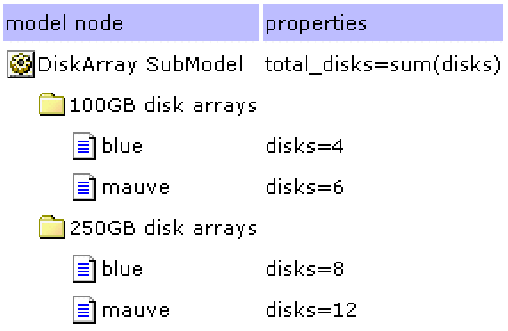
Properties may defined at any level in the model group hierarchy, from the root model group level to the individual model level. Where a property is defined determines which models can see and make use of the property. A little thought during the design of your models will speed model development and 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 groups 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

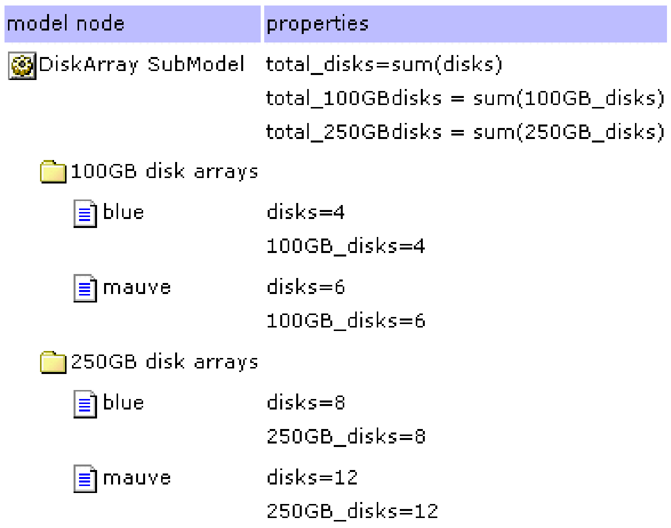
Multiple properties with the same value can sometimes make a model easier to build and maintain.This concept may be confusing at first and is best demonstrated by an example. Suppose that you are building a model that allows the user to choose from a selection of disks arrays. Each type of disk array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

FIGURE 11. **Using Multiple Property Names to Clarify Purpose**

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).



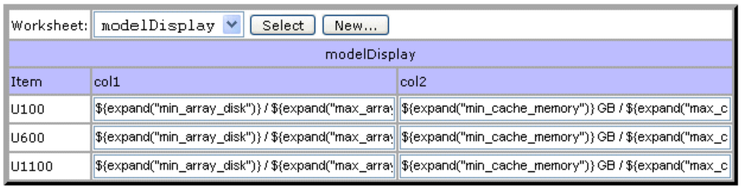
model node	properties
DiskArray SubModel	total_disks=sum(disks)
	total_100GBdisks = sum(100GB_disks)
	total_250GBdisks = sum(250GB_disks)
100GB disk arrays	
blue	disks=4
	100GB_disks=4
mauve	disks=6
	100GB_disks=6
250GB disk arrays	
blue	disks=8
	250GB_disks=8
mauve	disks=12
	250GB_disks=12

FIGURE 12. Using Additional Properties for Simple Calculations

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

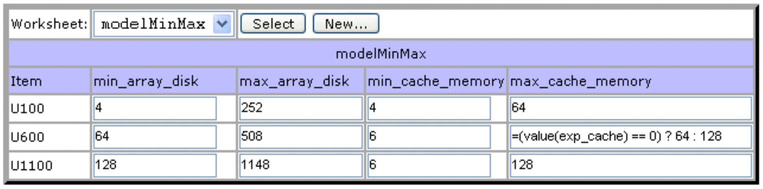
When developing a model, it is often necessary to assign the same set of properties to multiple option classes or option items. Worksheets are very useful in this case, since they allow you to rapidly set the values for a particular property on any number of option classes or items. this is especially 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	${expand("min_array_disk")} / ${expand("max_arra	${expand("min_cache_memory")} GB / ${expand("max_c
U600	${expand("min_array_disk")} / ${expand("max_arra	${expand("min_cache_memory")} GB / ${expand("max_c
U1100	${expand("min_array_disk")} / ${expand("max_arra	${expand("min_cache_memory")} GB / ${expand("max_c

FIGURE 13. Using Worksheets to Simplify Property Assignment

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	=(value(exp_cache) == 0) ? 64 : 128
U1100	128	1148	6	128

FIGURE 14. Worksheet Showing Min and Max Properties For Each Option Item

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 product 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 products 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 product'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 axel 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(axelTurnFactor) * value(wheelBaseTurnFac-
tor)* sum(turningElements)
```

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

The equivalent rule is as follows:

```
if (value(axelTurnFactor) >= 0 or value(axelTurnFactor) < 0)
and (value(wheelBaseTurnFactor)>= 0 or value(wheelBaseTurnFactor)<
0)
and (sum(turningElements) >=0 or sum(turningElements) <0)
      turningRadius = value(axelTurnFactor) * value(wheelBaseTurn-
Factor)* 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 product will have the same configuration.

- Dynamic instantiation:

  This technique allows multiple instances of a configured product 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.

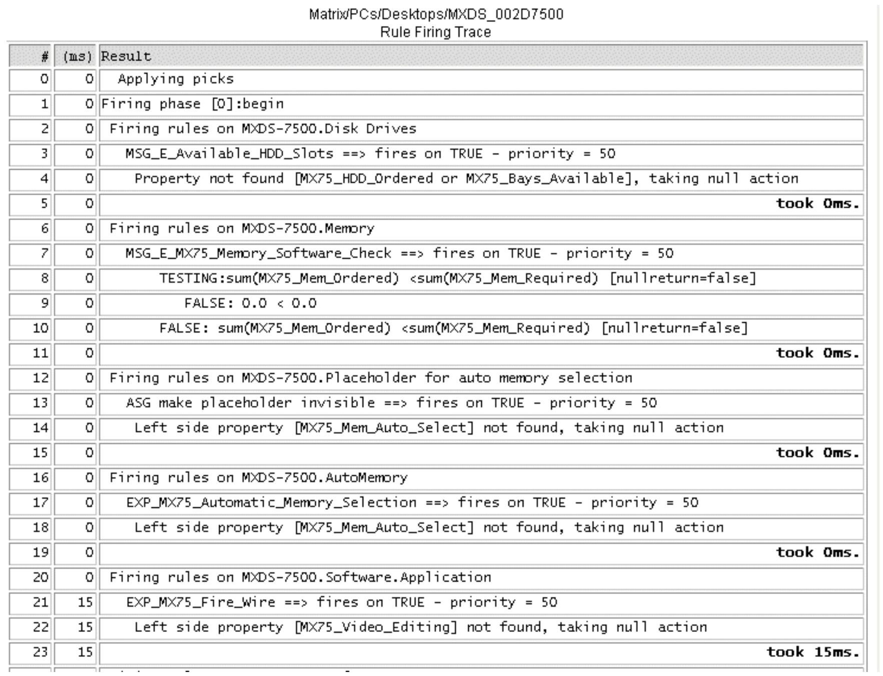
   The model displays in the Model Preview tab.
3. Click the Test icon.

   The model runs in a separate window.
4. 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.



Matrix/PCs/Desktops/MXDS_002D7500
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.**

FIGURE 15. Sample Rule Firing Trace

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.

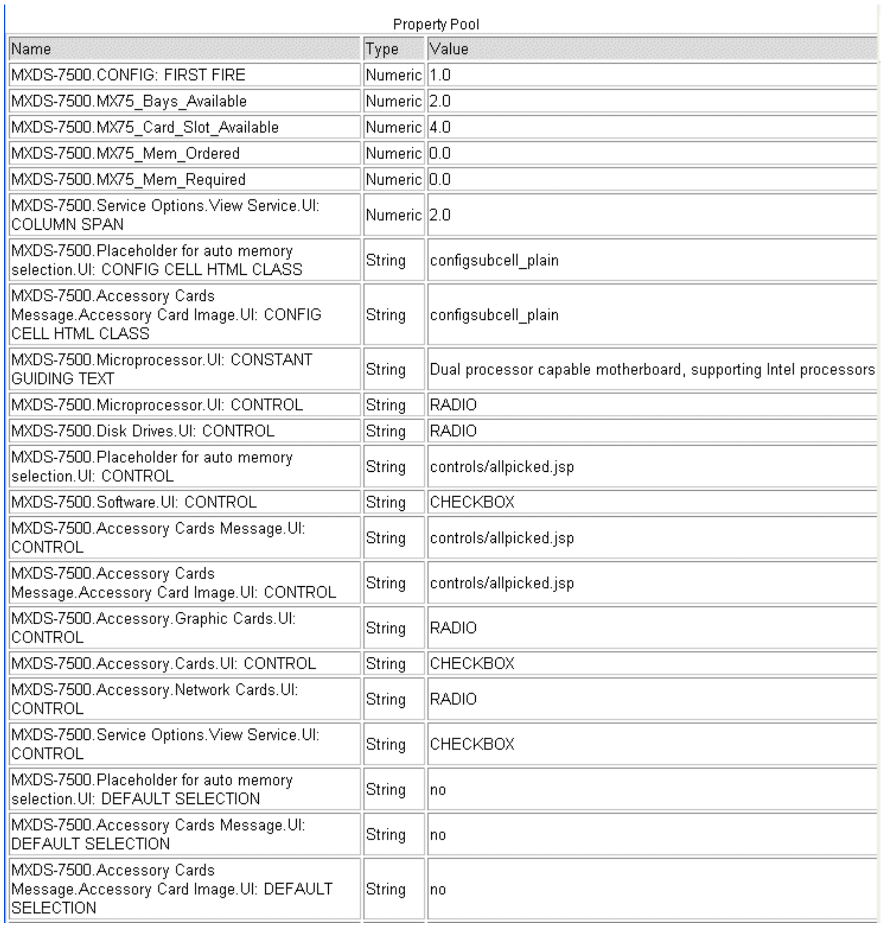
The log shows the number of milliseconds needed to fire a rule after each rule firing entry. The total number of milliseconds needed to run the model is logged at the end of the rule firing trace.

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

FIGURE 16. **Sample Property Pool Trace**

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.

For more information about load testing, see "Verifying Performance" on page 32.

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 that 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.

APPENDIX A

JVM Tuning and Log Analysis

Introduction

This chapter describes the following performance optimization items:

- "JVM Memory and Tuning Guidelines" on page 65 describes general guidelines for Java Virtual Machine (JVM) performance tuning. You should be familiar with your JVM and servlet container environment to apply these guidelines.
- "Log Analyzer Tool" on page 69 describes an open source log analysis tool, Log Analyzer. Contact your Comergent Technologies representative for information about obtaining this tool.

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 Comergent eBusiness System 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 Comergent Technologies 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/JPAppHotspot.fm.html#998292`
  `http://java.sun.com/docs/books/performance/1st_edition/html/JPAppHotspot.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 Comergent eBusiness System **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. Contact your Comergent Technologies representative for information about obtaining this tool.

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 Comergent Technologies 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>
```

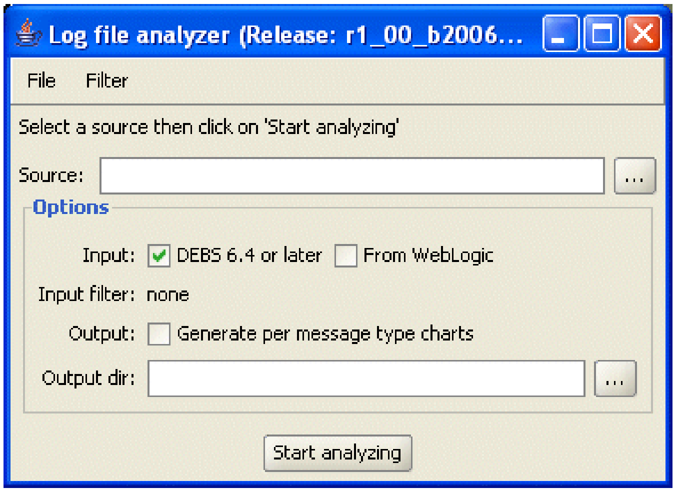
For example:

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

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 following screen displays:



Select a source then click on 'Start analyzing'

Source:

Input: [✓ DEBS 6.4 or later] From WebLogic

Input filter: none

Output: Generate per message type charts

Output dir:

FIGURE 17. **Log Analyzer Initial Screen**

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 Comergent eBusiness System, then uncheck this checkbox.
    - From WebLogic: click this checkbox to indicate that the log files are generated from a BEA 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 Comergent Technologies 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” on page 74 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
```

FIGURE 18. **Recommended Log Analyzer Directory Structure**

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 **site*n*/app-server/logs/** directory**.**

The **sites** directory is read-only. Output is written to the **site*n*/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
  ```

Index