JCC LogMiner Loader

Version 3.6.0

Released February, 2019
Documentation Released March, 2019

From the JCC Toolset for Oracle Rdb

JCC Consulting, Inc.
600 Newark Road
P.O. Box 381
Granville, Ohio 43023 U.S.A.

Contact Information

JCC Consulting is eager to hear your comments, questions, and examples. Send these to us in any of these ways:

E-mail	JCC-LMLoader@JCC.com
Phone	+1 (740)587-0157
FAX	+1 (740)587-0163
Post Office	LogMiner Loader JCC Consulting, Inc. Box 381 Granville, OH 43023

Notices

The information in this manual has been carefully checked and is believed to be accurate. However, changes to the product are made periodically. These changes are incorporated in new publication editions. JCC Consulting, Inc. may improve and/or change products described in this publication at any time. Due to continuing system improvements, JCC Consulting, Inc. is not responsible for inaccurate information that may appear in this manual. For the latest product updates, consult the JCC Consulting, Inc. web site at **www.jcc.com** or consult JCC in any of the ways indicated in the **"Contact Information" on page 3**. In no event will JCC Consulting, Inc. be liable for direct, indirect, special, exemplary, incidental, or consequential damages resulting from any defect or omission in this manual, even if advised of the possibility of such damages.

This product should not be used in any industry or fashion for which the underlying database products are not valid.

In the interest of continued product improvement, JCC Consulting, Inc. reserves the right to make improvements in this manual and the products it describes at any time, without notices or obligations.

This file and the LogMiner Loader software are confidential and proprietary to JCC Consulting, Inc. All documentation and the LogMiner Loader software are provided on an AS-IS basis.

Trademark Acknowledgments

JCC Toolset and JCC LogMiner Loader are trademarks of JCC Consulting, Inc. LogMiner, Oracle Rdb, Oracle 12 (and multiple other versions), MySQL, and Tuxedo are trademarks of Oracle Corporation. Windows (in its multiple versions) and SQL Server are trademarks of Microsoft Corporation. MariaDB is a trademark of MariaDB Corporation. Teradata is a trademark of Teradata Corporation. DB2 is a trademark of IBM Corporation. OpenVMS is a trademark of Hewlett Packard Enterprise (HPE). OpenVMS is also available from VSI (VMS Software Inc.) which holds the copyright to versions created by VSI.

Uses in this Document

In this document, Oracle Rdb and Oracle's other RDBMS are referred to frequently and need to be distinguished. Consequently, Oracle Rdb is referred to as "Rdb" and Oracle's other database product (whatever its version) is referred to simply as "Oracle." Similarly, Oracle Rdb LogMiner is referred to as "Rdb LogMiner" or as "LogMiner." The JCC LogMiner Loader is sometimes referred to simply as "the Loader".

Disclaimer

Table of Contents

CHAPTER 1

Using the Documentation and Kit Resources

JCC's LogMiner Loader is a flexible, powerful tool. What you need to learn from the documentation will vary with your background and your intended use. This section offers an expanded look at the contents of the documentation so that you can choose what you need.[1]

The chart to follow shows, in blue, the section that will be of interest to anyone who is curious about the Loader. This section also appears, in essentially the same format, on the JCC web site. Managers and others who want to understand the Loader's impact without getting into details will find this section useful.

The sections that have information valuable for anyone using the Loader are colored green. Installation and set up are colored green, as are the basic sections and the index. Database Administrators will need some of these. Systems Architects will definitely need the information in these sections.

1. DBAs might think of this as MetaDocumentation.

Examples are colored red. These can be very helpful. Look for the ones that apply to your use.

Advanced chapters are colored yellow.

Of the remaining sections, the choice of which you need will be determined by the uses that you intend to make of the Loader. For example, you will need one (and, perhaps, more) of the chapters on targets. You will probably need the chapter that explains modes or the one on the continuous mode or both.

TABLE 1. Guide to the Documentation

Chapter # and Short Name	Purpose and Audience
1 Guide to the documentation	This chapter is an expanded Table of Contents.
2 Introduction	Discussion of JCC's LogMiner Loader suitable for anyone who wishes a general understanding of its purpose and uses. See also http://www.jcc.com/LML.for this information.
3 Basics	Explanation of JCC's LogMiner Loader and Oracle Rdb's LogMiner.
4 Installation	How to get and install the Loader kit, plus licensing and other topics.
5 Continuous LogMiner and Loader	Most applications require Continuous mode for "near realtime" operation. This chapter elaborates the requirements and uses of Continuous mode for the LogMiner and Loader.
6 Modes of Operation	Different modes of operation and the requirements and uses of each mode, including Continuous, Static, and Copy.
7 Further Preparation	Post installation steps that finish the set-up.
8 Target Rdb	Required if and only if your target will be an Rdb database.
9 Target Oracle	Required if and only if your target will be an Oracle database.
10 Target JDBC	Required if and only if your Loader target will be JDBC to work with a JDBC driver to publish to an end target..
11 Target XML	Required if and only if your target will be XML to a file or to your own API.
12 Target Tuxedo	Required only if your target will be Tuxedo.

TABLE 1. Guide to the Documentation

Chapter # and Short Name	Purpose and Audience
13 Control File	Use, organization, and options of the Control File. A Control File is required for all uses of the Loader. Your application will not require you to control all of the options, but the chapter will help you decide which ones you want to control.
14 Monitor	Use and structure of the Monitor and the logs and other tools that help you analyze your application.
15 Performance	Options and tips for tuning performance.
16 Administrator	Additional assistance for the Loader Administrator.
17 Advanced Concepts: Schema and Data Transforms	Schema changes and data transforms and lists and examples to illustrate possible architectures.
18 Data Pump	Tool for repairing or initially loading the target database with data.
Examples	Reorganizing a database.
	Slaving an Oracle (or other) target.
	Additional architectures for the Loader, examples and tools.
	Extended examples and tools
Appendix - More Resources	A detailed listing to the contents of the Loader kit and what each component does, plus the directory tree for the examples that come with the kit. A list of the operator alarms that are possible. See the Administrator chapter and the Control Chapter for more on controlling where these messages show. A list of logical names that you can use to control the operation of the Loader. Frequently asked questions ... and the answers. The index.

This complete documentation is available on-line in Adobe PDF format. Of course, you can also print and bind the PDF. To find specific topics, use the table of contents, the index, or the search tools built into Adobe Acrobat.

If you still can't find what you need or any portion is unclear, please inform the support desk. Your questions may help improve the product or the documentation.

Other Documents and Resources

There are also other resources.

Kafka Option

Kafka is also available as a Loader target, but is separately licensed and documented.

Release Notes

This documentation is only updated for major releases. See the release notes for point release.

Blogs

The blogs available at

http://www.jcc.com/resources/jcc-blogs-menu/blog-jcc-logminer-loader-hints

are used by JCC to provide information relevant to the JCC LogMiner Loader community. The topics covered include frequently asked questions and especially important or time sensitive information.

Presentations

JCC Consultants and customers using the JCC LogMiner Loader have given presentations at various times.

CHAPTER 2

Introducing JCC's LogMiner Loader

JCC's LogMiner Loader is a fast, powerful, flexible tool to reflect data changes from a source database to multiple targets. The source is an Rdb database. Loader targets can be other Rdb databases, Oracle databases, customer supplied APIs in XML format, a Tuxedo application, a JDBC target or a Kafka messaging system.[1] Note that using a JDBC driver as the Loader target opens a wide range of end targets. Some of the compliant JDBC end targets are discussed in this documentation.

The Loader is extensively tested and tuned for all recent versions of Rdb, Oracle, OpenVMS, many of the end targets of the Loader JDBC target, a variety of network products, operating systems, and other companion products.

Testing includes random generation of failure scenarios, a wide range of Loader run-time parameters, and a combination of interfaces and options. Testing methods are discussed more completely in

1. The Kafka Option is a separately licensed and documented option.

http://www.jcc.com/lml-testing and notes on the companion products that are tested are included in http://www.jcc.com/lml_prod_compat.

Who Uses the JCC LogMiner Loader?

Those who use the JCC LogMiner Loader recognize that the information stored in their databases is one of their *most valuable resources*. They cannot afford additional impacts on the production database resources, but they need one or more of the following:

- To have a composite of regional databases
- To have multiple copies of the corporate information to scatter to remote locations
- To be able to use the data warehousing tools of Oracle or other platforms
- To provide web access without performance impact on the production databases
- To apply tools not available in the source database environment
- To reorganize a huge, critical, and overworked database without major downtime
- To add coherency to an environment with more than one database platform
- To archive data
- To audit data changes
- To convert to another platform without interruption in support
- To provide realistic data for serious application testing

Those who use the Loader have some of the largest, highest throughput databases in the world, as well as some of the most stringent demands for timeliness.

Architecture

An Oracle Rdb database may be configured such that the database engine logs all changes made to the database into a special file called the after image journal. The

after image journal (or AIJ) contains records that indicate the status of each and every database object at the completion of a transaction. The Oracle Rdb LogMiner tool uses the after image journal to extract the final copies of data rows as they appear at the end of each transaction.

The LogMiner output represents the changes to the source database. The JCC LogMiner Loader enables a database administrator or someone identified as the Loader Administrator to apply the contents of the LogMiner output to one or more targets.

When run in continuous mode, the Loader control process coordinates the actions of all the Loader processes and of the LogMiner. In continuous mode, the LogMiner Loader updates the target in near realtime.

Fault Tolerance

The JCC LogMiner Loader and the Oracle Rdb LogMiner write entire transactions and do not lose transactions.

The Loader is tolerant of environmental and downstream difficulties. The Loader can be stopped to resolve difficulties. The Loader recovers from exceptions and interruptions without losing data.

The Loader does not interfere with systems operations such as backup. The Loader - running with the Continuous LogMiner - can resume in the backed up AIJs and automatically switch to the live AIJ, after completing processing of the backed up AIJs.

Configuration Options

The JCC LogMiner Loader and the Oracle Rdb LogMiner are most often used in continuous mode, as a “live feed.” They can also be run in a mode that uses only backed up AIJs or in a mode, Copy Mode, that takes output from the LogMiner and applies it as if it were running “live.” Copy Mode is excellent for testing and for environments that do not provide fast, reliable network between the source and target.

The Loader may be configured to write all changes to the target or can be limited to a subset of tables, a subset of columns, and/or a subset of actions (insert, update, delete). A row can be included or excluded based on a filter applied to the row. Additional transforms are supported.

Monitoring, performance related characteristics, database administration options, and others are all configurable.

Monitoring

The JCC LogMiner Loader collects statistics and can display them in an on-line monitor, a file, or the log. The Loader also manages statistics from the LogMiner.

There are several styles of output available for the Loader statistics, including an on-line full-screen display, a Comma Separated Values (CSV) output, and output suitable to interface with "T4 and Friends," the Total Timeline Tracking Tools from OpenVMS Engineering.

The monitoring and logging tools can be constrained for normal operation or expanded for testing or resolution of a problem. Each step in the Loader processes can be thoroughly documented. Some indications of the behavior of the source and target are also reflected in the Loader monitor data.

Performance

The JCC LogMiner Loader impact on the system supporting the production database (the source) is generally negligible. JDBC targets depend on running the JAVA engine and their resource consumption may be noticeable.

In continuous mode, the updates to the target are "near realtime." Since the Loader does not get any information until after the commit, large transactions can vary from realtime. Small transactions will appear to be realtime. Throughput in all cases is impressive; as is demonstrated in the case studies.

The Loader supports dynamically adjusting parallel processes. The minimum and maximum number of processes to use in parallel are configurable through the Control File and are also adjustable while the Loader is running.

The Loader provides configurable commit intervals to group multiple source transactions into single target transactions to minimize overhead. If the source becomes quiescent, timeouts prevent the Loader's stalling with a partially filled commit interval.

Many Loader families can run simultaneously from the same source database with differing targets and different configuration choices.

Transforms

The JCC LogMiner Loader supports schema and data transformation. The transforms supported can be combined to solve a wide range of issues with disparate information systems. The Loader can also be used to create intended differences in the source and the target.

Indexing and other physical database parameters can be different on the source and the target.

Even the primary key can be different, although, for updates, there must be some way to uniquely identify a row in the target. Also, the primary key chosen must be unchanging, as the Loader only has available the data as it was after the transaction. The Loader includes a mechanism for using the dbkey from the source as a key in the target to address situations for which there is no reliable natural key.

Configuration choices can limit the tables, rows, or actions replicated and row-based filtering can further refine the subset of data written to the target.

Data in the rows can be transformed using any SQL expression that returns a single value of a single data type. The SQL expression can be any standard SQL, an Rdb built-in function, or a user defined function. The user defined function can be stored in the database used for Loader filters and Loader transforms. Do note that doing extensive transforms and external calls can introduce measurable latency in the Loader's work.

There are additional transforms already defined which can transform dates to meet different standards in different databases, define a value to replace a null value, trim trailing white space (blanks, carriage return, or linefeed) in going to non-Rdb targets.

Values can be materialized to provide virtual columns. The list of possible virtual columns includes such things as commit timestamp and other timestamps, values that can be used to partition the data, defined constants that can be added to the key to aid in rolling up similar databases with overlapping keys, Loadername, LSN (Loader Sequence Number), TSN, and other identifying criteria.

Additional Support for Running the Loader

The JCC LogMiner Loader is fully multi-variant. One version can be run while another is being tested.

There are procedures for generating template Control Files. For many of the Control File options, there are defaults that will be satisfactory, without needing to define the option.

Examples are included in the kit.

There are Loader features that aid testing the overall application and its impact on the database. These include the option of throttling Loader performance or setting Loader performance to emulate realtime. The Loader can, alternately, be set to run faster than realtime by some set amount to test the performance of the overall system in the face of growth.

In addition, the Loader provides operator messages. What triggers an operator message and where the message should be displayed are both configurable. For example, operator messages can be used to provide an alert if the downstream processes are backing up and not able to absorb the Loader output.

Data Pump

Downstream processes may do worse than just slow the throughput. When target databases lose data or are inappropriately updated, it is often possible to trace which data has become corrupted. In these cases, as well as for initial population of the target, the Data Pump of the JCC LogMiner Loader is a valuable resource.

The Data Pump can be configured to use a Structure File of SQL statements and a Driver File with selection criteria. The Structure File can represent hierarchical structures. For example, if a certain block of accounts were corrupted, both the accounts and the child tables of bills and payments may need to be updated.

The Data Pump is packaged with the Loader and takes advantage of the Loader's nature. The Data Pump works by making no-change updates to the source database, causing the Loader to write the unchanged data to the targets.

The Data Pump can also be used for initial population of a target.

Users of the Data Pump are finding it fast compared to alternate approaches. That it is also reliable and configurable is also important.

Companion Products, Versions, and Testing

The JCC LogMiner Loader is rigorously tested, continuously, with an automated, random regression test that generates different options and, also, randomly "attacks" processes to emulate failure scenarios.

Loader testing has included OpenVMS for both alphas and Integrity on both hardware and emulator platforms. The JCC LogMiner Loader is fully supported on Integrity platforms, including the i2 and i4 architectures.

The Loader has been tested with all recent versions of all the companion products - Rdb, Oracle, SQL*net, OpenVMS, Tuxedo - and some of the older versions, as well. The Loader has also been tested with some of the wide range of class 4 JDBC drivers and the products they access. Details are provided in http://www.jcc.com/lml_prod_compat.

Should you have any doubts about your proposed combination of products, please contact JCC (info@jcc.com).

Success Stories

The JCC LogMiner Loader is a general-purpose tool and can assist an Administrator with a variety of functions. The following examples demonstrate a selection of the varied uses of the Loader:

Case 1: Making HUGE Data Volumes Available to Additional Tools

A company wanted to begin using a wider range of products and applications. All of these were to be dependent on the data in the production Rdb databases. The original application had divided the data into almost thirty related Rdb databases because the data volumes, update rates, and query demands were so large that partitioning was deemed necessary to successfully manage the huge data volumes.

The Loader provides XML to a customer-defined API and also provides data to a Tuxedo application. A set of Loaders handle all the transactions for all the source databases while maintaining transaction consistency and interacting with the targets. The Loaders process the data at a peak rate of 2,400 rows per second.

Case 2: Meeting the Challenge of Timely Web Access

The primary Rdb database resides at the central office. Other offices are around the world. The critical challenge came when the production database could not offer information quickly enough to satisfy queries in the remote offices and to capture the business that was available through to online processing.

Using the Loader has permitted distributing multiple copies of the database with up-to-the-second accuracy. The query databases can be tuned differently from the source database to get the best query responses. Bookings are rolling in.

Case 3: Application Testing with Realistic Data

A company is already using the Loader to replicate data to products supporting a variety of specialized functions. The company needs to test a replacement for the downstream database and applications. The Loader is used to run transactions packaged from actual AIJs. These are, then, run in realtime emulation.

They could, alternately, be run to feed the actual data at a configurable pace that exceeds the production rate, thus testing scalability.

Case 4: DB Reorganization when Downtime is NOT Possible

A company did not provide routine maintenance for the Rdb database because the company could not afford to be without it for long enough to accomplish the maintenance. In time, the company "hit the wall," as mixed areas filled up. They exceeded the limits of the database physical design. This was a 60 GB database and traditional reorganization techniques would require 24 hours of down time. Business growth was healthy, but the computer support was already not adequate to handle past growth. Database reorganization could be deferred no longer.

A copy of the production database was made. The reorganization was accomplished on the copy and was extensively tested. Meanwhile, business continued on the production database. When the reorganized database was ready, the Loader was used to apply the data changes that had occurred in the production database since the copy was made. The total downtime for end-user support was thirty-five minutes, including additional testing.[1]

Case 5: Speed and Large Transactions

An enterprise regularly peaks at over 4,000 transactions per second and the Loader maintains near realtime performance.

The DBAs did a convert/nocommit and had to change the truncate table statements into delete from table, resulting in transactions of half a million rows. It took an enhancement to the Loader to provide the performance desired. The Loader, now, changes locking strategies upon encountering extremely large transactions. The threshold for the strategy change is, of course, configurable.

1. "Example: Reorganizing an Rdb Database" on page 529 provides details.

Case 6: Displaying Time Critical Information on a Map

A utility company with all its data in an Rdb database wanted to use a graphics package to display the (service) outage information. The package did not have an interface to Rdb and NOBODY wanted to re-enter data.

Using the Continuous LogMiner Loader, the information is displayed in near real-time, yielding significant improvements in service, communication, and safety.

Given that success, a different application of the Loader was utilized to provide outage data to an SQL Server database that is a source for display on the web.

Following a recent weather related outage that put one third of the customers without power, the press raved about the service provided and did feature presentations on the technology.

Case 7: Building a Coherent Information Resource

A company with a number of regional databases and applications that were aging was able to use the Loader to build a coherent database from the regional databases. Having more timely information available when needed was so successful that they moved on to other applications, including distributing subsets of the data where they needed it. Now, with much better control of their data and application, they are looking at a complete re-write, built, from the beginning, with the Loader as an integral part.

Support

A *Basic* support contract for one year is provided as part of the software purchase. Basic support includes the right to new releases (during the term of the support contract), plus call in and e-mail support during normal JCC business hours.

An upgrade to *Gold* support includes all Basic support, plus 24 X 7 coverage.

Basic or Gold support contracts can be renewed for subsequent years.

JCC Consultants are also available to work with you on-site on a consulting basis.

License, Documentation, and Kit

The distribution kit may be obtained from the JCC web site. Look for options in the light yellow box on the upper right of the web site at

http://www.jcc.com/lml

You can acquire the full kit, this documentation, or release notes for recent versions.

You will need a license key to be able to use the product. Temporary licenses are available for investigation prior to purchase.

CHAPTER 3

Basics

JCC's LogMiner Loader enables high performance, easily achieved solutions to issues requiring moving data from an Oracle Rdb database to a target data store or data transport.

This chapter describes some of the basics in greater detail than provided in the introduction. The chapter will reveal the overall architecture and help you understand the choices that let you tune Loader use to your requirements. Topics also include different modes of operation, the input and output of a Loader session, how the Loader identifies rows in the target, and the concept of quiet points within the AIJs.

This chapter provides an important overview. Later chapters provide greater detail.

Architecture: Rdb and LogMiner

An Rdb database may be configured such that the database engine logs all changes made to a production database into a special file called the *after image journal.* In

this documentation, such a production database is called the *originating database* or *source database*.

AIJ Files

The after image journal (or *AIJ*) contains records which indicate the status of each and every database object at the completion of a transaction. The original purpose of the after image journal is to be able to recover a backup copy of the original database to an appropriate point should a disaster occur.

The Rdb LogMiner uses the after image journal to extract the final copies of data rows as they appear at the end of each transaction.[1]

Each extracted row is stored as a record. The structure of the record includes control information about the row and the transaction as well as the data values in their internal format. The row also includes a null bit vector.[2] Essentially, the row is presented as it is stored in an Rdb database.

Options Files

The LogMiner is controlled with an *Options File*. Note that you can instruct the LogMiner to include all of the tables or only some of them.[3] For most uses, you will also want the LogMiner to include the commit rows.

The Rdb documentation will tell you how to construct the options file. Also, the JCC LogMiner Loader Toolset includes procedures that generate template options files for the LogMiner. “Preparing the Source Database” on page 97 provides more information on setting up the options file.

1. A given row appears in the LogMiner output only once per transaction.
2. In Rdb, the null bit vector for a row indicates which columns, if any, are null.
3. If you are going to exclude a given table from your target, less work is required (therefore, performance is better) if you do not ask LogMiner to provide it.

Architecture: LogMiner and LogMiner Loader

The LogMiner (Rdb) and the LogMiner Loader (JCC) have been developed to work well together. As you become familiar with these products, you will see that the Loader provides controls, monitoring, logging, and other enhancements to complement the LogMiner.

Architecture: Applications

The LogMiner and LogMiner Loader combination has the flexibility to support a wide range of applications. This chapter defines some of the choices that you will make.

One of your first choices is likely to be which target data store you will use. See "Loader Targets" on page 34 and the sections referenced there for more on targets supported.

"Success Stories" on page 22 provides examples of uses of the Loader and some specific success stories. These may help you to understand some of the options that are available. "Using the Documentation and Kit Resources" on page 11 can help you identify additional documentation and kit resources.

The power and flexibility of the Loader product imply that you make choices, if you wish to do anything other than straight replication. This chapter is intended to help you understand the most basic of those choices so that you can define your architecture.

Modes of Operation

Most frequently, the LogMiner and the Loader operate in Continuous, near real-time, mode. Other modes are possible. Choice of mode depends on your goals and resources. Database reorganization and testing are two of the applications that call for other modes of operation. Certain data volumes and/or resource constraints may also recommend against Continuous Mode. The uses and requirements for the modes are discussed further in "Modes of Operation" on page 77 and in "Continuous LogMiner and the Loader" on page 69.

The *combined* architecture of the Rdb LogMiner and the JCC LogMiner Loader is designed to function without interfering with the normal operation of the source database.[1] This basic principle applies to the LogMiner and Loader running in any of the modes.

Using Continuous Mode

The LogMiner can run against live database AIJs to produce a continuous output[2] stream that captures transactions when they are committed.[3] Most applications, today, use Continuous Mode to provide seamless, near realtime[4] updates to one or more targets.

The continuous Loader session may be shut down, database work and even AIJ backups may proceed, and the Loader session may resume later without losing information. If AIJ backups have occurred while the Loader was shut down, the LogMiner and Loader will be started pointing at the backup AIJ files. The Loader and LogMiner conspire to replay the backed up AIJs in the correct order and switch to the live AIJs when the backup information is exhausted.[5]

Continuous mode is discussed further in “Continuous LogMiner and the Loader” on page 69 and throughout the document.

Continuous Mode is the default.

1. The LogMiner does attach to the source database and can inhibit AIJ backup in certain circumstances. The JCC LogMiner Loader provides a method for avoiding this difficulty. See “Loader Heartbeat and AIJ Backup” on page 475 for a more complete description of the problem and the solution.
2. All recent versions of Rdb support Continuous LogMiner (CLM). The earliest versions to support it were Rdb Version 7.0.6.3 (with a special LogMiner patch) or Rdb 7.1.0.2.
3. The continuous output stream, if requested, includes additional commit records which the JCC LogMiner Loader uses to determine the end of a transaction.
4. JCC uses the term “near realtime” rather than “realtime” to describe operations because the LogMiner cannot acquire the information until the source database commit.
5. “Rdb Issues” on page 412 in the chapter for Administrators provides details of the architecture and techniques for restart.

Using Static Mode

The first production application of the Loader was to provide database reorganization with minimal downtime. Database reorganization continues to be an important application of the LogMiner and the Loader. However, improvements in operation in Static Mode have been made since the reorganization used as an example.

Static mode - in both the original and improved versions - is also discussed in “Modes of Operation” on page 77.

Using Copy Mode

The Loader can also be configured to use a hybrid of static and continuous. In copy mode, the Loader uses a pre-processed (static) LogMiner output file, but applies the results to the target in an imitation of continuous update. This mode is excellent for testing applications without disrupting production. (See “Loader Tools for Testing” on page 482.) Copy Mode is also valuable when there are reasons to manage source activities separately than target activities.

Control File

The Control File provides “knobs and levers” to specify how the JCC LogMiner Loader will run. In other words, the Control File determines how the JCC LogMiner Loader will transform the LogMiner output for the target. It also controls options that tune performance, logging, and failure modes. This is how you distinguish your use from the defaults.

The Control File may specify that tables in the target database are completely synchronized with the source database, including all updates, inserts, and deletes. Alternatively, it may specify that a target table include the most recent version of all deleted rows with sufficient added information to provide a complete audit trail. Other combinations are possible and can be tuned to the solution required.

Control Files are inherently complicated and lengthy. They specify all columns in a table, the expected data types of those columns, whether the column is part of the primary key, and what should be done in the case of delete operations. They also specify a wide range of other items.

To provide an easier start, the JCC LogMiner Loader Toolset includes procedures that generate template Control Files. The procedures can also be used to generate template LogMiner *options* files to control the Rdb LogMiner. These templates may require further editing to meet your exact requirements.

The generated Control Files are designed to cause the JCC LogMiner Loader to maintain the rows in the target database as a complete replica of the corresponding rows in the source database. To achieve other behavior for the target, the Control Files must be edited. The chapter "Control File" on page 217 provides the details that you need to create and edit the Control File.

The Control File is required.

Logical Names for Control

Logical names provide additional control. These are mentioned throughout the documentation, as the topic warrants. They are also summarized in the appendix in "Logical Names" on page 585.

The Loader kit includes a tool for managing logical names used with the Loader. See "Logical Name Controls for Loader Procedures" on page 463.

Loader Input and Output

The diagram summarizes the input and output for the JCC LogMiner Loader.

The output from the Rdb LogMiner is required as input, as is the Loader Control File and some logical names. These are introduced in the preceding.

The Loader will output to the target defined in the Control File. It will also write to a checkpoint file[1] and read from the file for restart. The Loader also outputs log

1. For Rdb and OCI Loader targets, the checkpoint is written to the database instead of a file.

files[1] and can be directed to display status information for the online monitor.[2] The output is further introduced in the following sections.

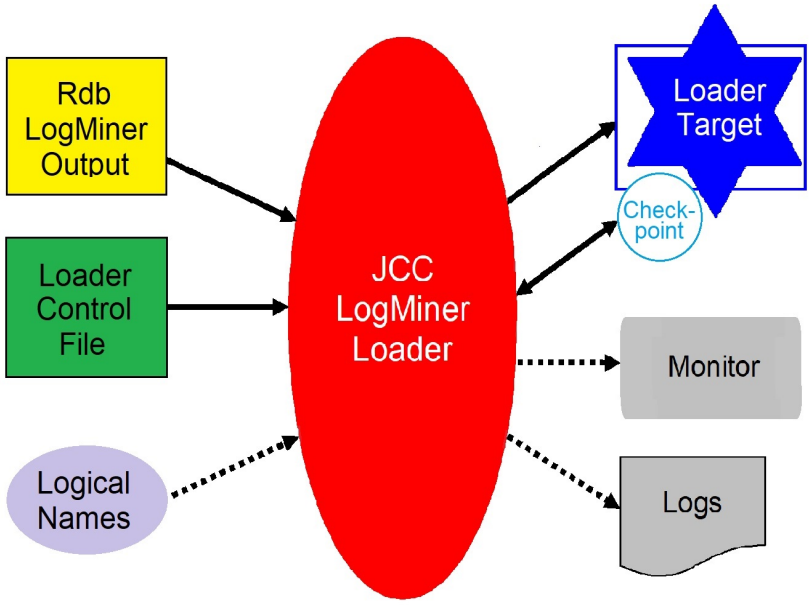


FIGURE 1. JCC LogMiner Loader Input and Output

1. For a further discussion of log files and which log files are maintained, see
2. For a complete discussion of the statistics monitor see

Loader Targets

The JCC LogMiner Loader supports an extensive range of targets and output formats. The direct targets of the Loader include Rdb, Oracle, JDBC, Tuxedo, file output, and tools for Kafka.[1] Through the JDBC support, any database with a supported class 4 JDBC driver can be an end target. The Loader also supports a range of transports and formats including OCI, XML, JSON, and Avro (Confluent Kafka). A customer supplied API can absorb Loader output and process it further before writing it to the end target.

Most of the end targets run on operating systems other than OpenVMS.

Consult the blog http://www.jcc.com/lml-prod-compat for specifics on version combinations that have been tested.

A summary of the Loader targets, transports, and formats is shown in the chart.

TABLE 1. Targets, Transports, and Output Formats

Loader Target	Transport	Format	End Target
Rdb	Rdb	Rdb native	Rdb
Oracle	OCI	Oracle native	Oracle
Tuxedo	Tuxedo	Tuxedo native	any Tuxedo target
JDBC	JDBC	JDBC driver specific	any database with a supported class 4 JDBC driver, including Rdb, Oracle, SQL Server, Teradata, Ingres, Postgres, MariaDB, and others.
File		XML JSON (untested)	File or any target supported by a customer-supplied API
Kafka	Kafka	Avro JSON XML	Any “data lake” or other repository written to by Kafka messaging systems

The physical organization of the target may be similar to the source database, or it may be quite different. You may elect to maintain tables corresponding to all tables

1. The Kafka Option is separately licensed and separately documented.

in the source database or to maintain only a subset of tables. You may maintain all columns or only a subset of the columns in any of the tables to be mined. You may add materialized columns such as commit timestamp or entire materialized tables. You may filter rows based on criteria in the row using an SQL predicate. You may write data from a given table to several different tables or, given appropriate keys, combine data from different tables or different databases. You may accomplish a variety of transforms of the source data. Your choices are indicated with the Loader's Control File. For more information on using the Loader to transform your data, see "Schema and Data Transforms" on page 489.

The choice of target will determine which sections of the documentation and which examples are helpful to you. The choice will also determine which of the keywords you use in the Control File.

Transactions and Recoverability

Transactional consistency and recoverability are protected by the LogMiner and the Loader.

Transactional Consistency

The LogMiner stamps each record in the output with the identity of its transaction. This transaction stamp is used by the Loader to provide transactional consistency in the target. The Loader always works within transactional boundaries.

Commit Interval

To improve performance, the Loader allows the DBA (DataBase Administrator), Loader Administrator, or other person responsible for running the Loader to specify a *commit interval.* The commit interval is the number of source transactions that are to be bundled into a single Loader transaction. The value for the commit interval may be specified in the Control File.

It is important to realize that the Loader cannot predict which tables are going to be processed at any point in time. If Loader commit intervals are too small, excessive I/O can result. If they become too large, excessive numbers of locks will be required. Larger commit intervals will also require more virtual memory.

Retries

There can, of course, be no guarantee that the Loader will not encounter lock stalls and deadlocks or that the network or target machine will not fail. The architecture is intended to provide some fault tolerance and to provide recoverability if the fault is beyond the Loader's control.

The Loader can encounter deadlock or some other system problem that might cause the transaction to fail. The Loader will automatically detect deadlock and rollback and retry the commit interval.

It is also possible that the target database is down or that the non-database target does not respond. The Loader will retry. How many retries occur is specified in the Control File. (See "Keyword: Output_failure" on page 278.)

Recovery

Of course, it is also possible that the Administrator will need to shut down the Loader for some reason. Whether the shut down is intentional or a result of an unanticipated event, it is important that the Loader resume operations without losing any updates.

To provide recoverability, a Loader-specific *checkpoint*[1] is maintained. At each Loader commit, the Loader records, in the *checkpoint file* or the *highwater table*, the current high-water mark.[2] During recovery this table (or file) is read and the Loader begins transmitting data at the transaction immediately subsequent to the last transaction committed.

For applications that use the highwater table, the update of that table is part of the overall update and the Loader has a completely accurate record of the last transaction.

There are two scenarios that may cause the Loader to re-send some data.

---

1. For database targets, the checkpoint is usually maintained in a table that is added to the target database. Other checkpoint mechanisms are also available. See "Keyword: Checkpoint" on page 231.
2. This consists of the LSN, TSN, AERCP, and other data. For additional information, see "Displaying Checkpoint Information" on page 372 in the Monitor chapter or "Rdb Issues" on page 412 in the chapter for Loader Administrators.

1. For applications that use the checkpoint file, the Loader updates the file after receiving the acknowledgment of receipt from the target. Since the issue that caused failure may have developed after the target received and recorded the update, but before the Loader received and recorded the acknowledgment, it is possible that the target may, upon restart, receive duplicate transactions.
2. For multi-threaded use of the Loader, the lowest thread LSN (Loader Sequence Number) is used to determine the start point. This may result in re-sending some data.

In all cases, *the architecture prevents the target's missing any transactions.*

Performance

The Loader's job is to transfer, to the target, updates that may have been written to the source by many tightly-tuned processes. The Loader needs to transfer these updates with perfect accuracy. The Continuous LogMiner Loader also needs to do this without falling behind. There are a variety of mechanisms included with the Loader to enhance and tune performance.

Performance within the Loader

One powerful mechanism for enhanced throughput is to work with parallel threads.[1] To examine how to specify threads in the Control File, see the sections "Keyword: Thread" on page 292 and "Keyword: Parallel" on page 279. To learn about monitoring threads see "Monitoring an Ongoing Loader Operation" on page 313, particularly "State Report Examples" on page 339.

Control of commit intervals ("Commit Interval" on page 35) and many other options enhance Loader performance. To examine other performance enhancements, see "Performance Considerations" on page 381.

1. The Loader does not use OpenVMS P-threads. Loader "threads" may be more properly designated as parallel processes. See "Keyword: Parallel" on page 279 for more information on the Loader's use of threads.

Performance and Companion Products

In the chapters on targets of the Loader and elsewhere, there are examples of Loader tuning options that are added in response to testing performance of the target products. For example, sections that contain valuable performance information related to specific companion products include these.

- "Keyword: Sort" on page 283 for performance tuning for OCI.
- The Oracle target chapter, the section "Reserved Words" on page 138 and others.
- "Systems Tuning Using JDBC as the Loader Target" on page 148.
- "Source Columns with TINYINT Data Type" on page 166 for suggestions when using JDBC targets.

System Performance

Both the Performance chapter and the chapter for Loader Administrators offer additional information.

Monitoring

The LogMiner Loader offers a variety of monitoring options. Realtime on-line display is available in a variety of formats. Other options for output include T4 and CSV, each of which have extended opportunities.

Information is also available in the logs and the Loader manages logs for both the LogMiner and the Loader. Using the Control File, you can request additions to the log.

Note that, just as the Loader includes options for improving overall system performance, the monitoring also provides tools for viewing more than just the Loader. The information provided is sufficient to understand details of performance throughout your system from the source database to the target. Further, T4 output from the monitor can be combined with T4 output from OpenVMS and Rdb.

See "Monitoring an Ongoing Loader Operation" on page 313 and "Keyword: Logging" on page 256 for more information.

Identifying Rows in the Target

Rows that are to be updated in the target must be correlated unambiguously to rows in the source database. That is, the Loader must be able to unambiguously identify which row is to be updated or deleted. In some database environments, primary keys are available for every table. In others, specifying a particular row within the table is problematic. The JCC LogMiner Loader must work with both scenarios.

Therefore, the Loader provides mechanisms for identifying a specific row within a table. The first is taken from data modeling theory, the second relies on certain behaviors in an Rdb database, a third mechanism is available if the IDENTITY attribute can be used in the source.

From data modeling theory. Each table will contain a column or set of columns that uniquely identify a row. These columns are considered to be the primary key.

These columns may or may not be indexed. They usually are for performance, but indexing and primary key definitions are not necessarily the same thing, despite the fact that they are sometimes used as synonyms.

In order to transport row changes to the target database, the Loader must rely on all columns in the key to be unchanging and not null. If any field in the primary key may change or may be null, [1] the Loader must rely on an alternate method. The Loader only has the key to identify the row. If the new values for columns of the key do not match the old, the Loader cannot identify the original row in the target.[2]

From Rdb internals. Each Rdb row has a unique internal address called its database key, or dbkey. The Rdb LogMiner provides the dbkey in its output. The JCC LogMiner Loader can correlate the dbkey value with a special column in the target database, the *originating_dbkey.*[3]

---

1. The restriction on nulls is inherited from Rdb, but is also consistent with good practice and the SQL standard.
2. For example, targets that use a date column as part of the key where that column might have less precision in the target than the source column cannot be guaranteed to get the expected results.
3. When the target is Rdb, by default, this column is an 8-byte string. Alternately, it can be BIGINT for compatibility with tools that need that. When the target database is Oracle, this column is a NUMBER. When this value is requested for other targets, the dbkey of the source record will be interpreted as a large number.

Side effects can occur if the originating dbkey approach is required. If any action causes a row to change its dbkey, that action may cause the target database to be maintained incorrectly. Some examples of such invalidating actions are:

- Data in the source database is reorganized through any of these operations
  - Export-import operation
  - Unload-truncate-load sequence
  - Alter storage map reorganize.
- Data rows are updated through read-delete-insert sequences
- Data rows are deleted and then space is reclaimed by another row. The second row may have the same dbkey as the first row, but represent a very different row semantically.
- If the Control File specifies a Rollup operation or a NoDelete operation, the Loader will not delete the first instance of the row and this would lead to confusion on updating any new rows with the same dbkey.

Should the Administrator elect to use the dbkey approach with an Oracle database or other target, tools and procedures included in the kit can be used to extract Rdb dbkey values, include them as numeric columns in the target and generate the appropriate table and index definition scripts in Oracle SQL.

JCC recommends that whenever possible the primary keys of tables be regular data columns instead of the synthesized dbkey columns.

Identity Attribute. An IDENTIY attribute is a shorthand mechanism for adding and maintaining a unique id generator for any table. This feature of Rdb Release 7.1.0.2 and later relies on the SEQUENCE and AUTOMATIC column features and can be specified by CREATE or ALTER TABLE.

Using this mechanism can provide a column within the source table that can be replicated and used as the primary key on the target. The identity column does not have to be the primary key in the source for the column to provide the primary key for the target, nor does the source have to have any indices or constraints added.

Using an identity column in the source to provide a primary key for the target avoids the issues associated with relying on dbkeys. However, in some environments, it is unacceptable to make any changes to the source database. If existing programs or tools look at all columns, this change may cause chaos. Even in environments that permit changes to the source database, adding the identity column

will add eight bytes to each row and may cause fragmentation. Even so, the identity attribute is often the best solution and the required changes[1] are acceptable.

Summary

One of these mechanisms must be chosen for each table for which the Loader must identify rows to update or delete.

Inserts and Updates

LogMiner for Rdb accesses the database after image journal (AIJ)[2] to obtain a list of all changes committed to a source database. These changes are characterized by the LogMiner as either updates or deletes.

The LogMiner characterizes insert operations as updates. Accordingly, the JCC LogMiner Loader cannot, a priori, distinguish between update and insert operations based on the data in the LogMiner output.

For replication to an Rdb, an Oracle, or JDBC target, the JCC LogMiner Loader will first attempt to update a row by the specified primary key and, should that operation fail, it will then insert the row. To preserve efficiency in this operation, a multi-statement SQL procedure is computed once per table. This statement is then executed once per row. The result of this is that the LogMiner Loader will perform precisely one interaction with the target per row in the output.

A similar process is used when maintaining an Oracle target database. Similar dynamic SQL procedures are created, complied and executed. By default, the Oracle paradigm requires that these procedures be prepared several times over the course of a connection. However, the JCC LogMiner Loader may be instructed to sort all rows in an input transaction by the name of the table being maintained. In

1. Regular use of the Loader writes nothing to the source. Using the identity attribute to provide a key is not something that causes the Loader to modify the source, but it may require the Database Administrator to modify the source.
2. The AIJ used may be the backed up AIJs or the active AIJ or a combination, depending on the Loader mode chosen and the circumstance.

this way, efficiencies in preparation of rows may be achieved leading to greater throughput for the SQL*net connection. [1]

Replication and Other Options

One option that may be specified in the Control File is to determine whether the table is to be exactly replicated or whether some other combination of actions should be written to the target. To replicate a table is to get the same table.[2]

It is possible to decide, separately for each table, whether the target table is to receive, or not, the updates, the inserts, and/or the deletes. All combinations are possible with the Loader, although some of them are more likely than others to be useful. In addition, to 'replicate' which means 'insert,update,delete', the Loader recognizes the named combinations: 'rollup' and 'audit'.

It is also possible to decide that all tables should be treated the same for choosing to replicate, audit, or whatever and there is a "wildcard" option to more easily enter this in the Control File.

To 'rollup' is to preserve all rows, even deleted rows. For rollup, a record in the LogMiner output that is flagged as a delete will be changed into an update. The result of this will be that the last version of a row *known to Rdb* will be left in the target database.[3] If you use rollup, you will probably also want to materialize the action to serve as a logical delete flag.

'Audit' means 'insert,noupdate,nodelete'. Audit writes each insert, update, and delete as an insert. If you use audit, you may want to materialize the action, the commit timestamp, and the session user to complete your audit information.

1. See "Keyword: Sort" on page 283 for more information. The sorting described here is BY_RECORD. Since that is the default, you don't actually need to include the Sort keyword, but understanding it will help you understand the Loader output.
2. See REPLICATE as the action parameter for "Keyword: Table" on page 286.
3. If a key is reused for a different row, then rollup will not function as you probably desire. For this reason, rollup cannot be done on tables that are synchronized through the dbkey mechanism, but the issue can apply to natural key columns, as well.

There are other combinations that are possible. The answer may be tailored to your application. See “Schema and Data Transforms” on page 489 for further discussion, “Keyword: Table” on page 286 for details of how to specify which actions to include and “Keyword: VirtualColumn” on page 299“Quiet Points” on page 101 for a discussion of how to add materialized data.

Quiet Points and AIJs

The LogMiner and the Loader are transactionally consistent. That is, they work only with whole transactions. AIJ backups can be done such that a transaction starts in one backup file and continues in the next. There is no harm in this, once the LogMiner is in operation. However, to start the LogMiner, it is necessary to begin on a *quiet point* boundary. This is not difficult to do and is described in “Quiet Points” on page 101.

Restrictions

Restrictions are discussed here. Other comments on how to make specific things work well are included throughout the documentation. The chapter “Aids for the Administrator” on page 409 provides in-depth coverage of a number of details.

Restrictions Inherited from LogMiner and Rdb

All restrictions that apply to the Oracle LogMiner for Rdb apply to the JCC LogMiner Loader. For instance, segmented string columns or vertically partitioned tables are not supported, nor is table truncation or some other delete operations for which details of the rows are not journaled. No column in the primary key may be NULL or may be changed. Other restrictions derived from the Rdb LogMiner also apply. Please reference the Oracle Rdb documentation or VMS HELP for a complete list of these.[1]

1. In addition, use of the LogMiner and Loader, by default, does not support distributed transactions, although the TID can be supplied as a materialized value. See “Keyword: VirtualColumn” on page 299 for additional notes on materialized columns.

Operational requirements for LogMiner are discussed in “Enabling the LogMiner” on page 102 and elsewhere.

Some additional restrictions applied in early versions of the Rdb LogMiner. You are advised that you should verify that you are running the latest patch levels of the Oracle Rdb LogMiner code. Contact Oracle Support to obtain a proper copy.

Known Rdb Issues

Any Rdb issues, that impact the functioning of the JCC LogMiner Loader, that are not resolved prior to publication of this documentation, will be reflected in “Aids for the Administrator” on page 409.

Restrictions for Some Targets

Some targets impose their own restrictions. These and recommendations for best use are included in the sections on specific targets and, where applicable, in descriptions of particular choices for keywords in the Control File.

Restrictions Imposed by the Environment

You may also have restrictions based on environmental factors or company policy.

Operational Restrictions

Quiet point AIJ backups are required for starting the LogMiner, initially. The Loader will start with the first transaction in the backup AIJ immediately following the quiet point backup request.[1] Restart will also use a quiet point. However, restart will use the highwater record of the last quiet point whether it was requested or occurred naturally. Naturally occurring quiet points are referred to as Micro Quiet Points and are surprisingly frequent. Note that daily quiet point backups will, in some circumstances, reduce the time taken to find the restart point. However, the Loader can start in the middle of a journal and that journal need not start on a quiet point.

There are also locking considerations with LogMiner and backups.

1. It is recommended that the AIJ backup that precede this one be removed to a different location to avoid confusion.

- The LogMiner should not be started or restarted when there is an AIJ backup being processed. This implies that surprise AIJ backups (whether triggered by a Database Administrator or by automated processing can have serious consequences.[1] The Loader kit includes a tool [2] to determine whether it is safe to begin backup. This tool can be built into the backup routines when automated backup tools are not used.
- The LogMiner, while running, can block backups. The LogMiner maintains a lock on the last location in the AIJ that it is reading. This prevents an RMU/BACKUP/AFTER command from processing. If there is no activity in the journal that is pertinent to the LogMiner's work (as defined by the Options File), the lock will not advance and the backup will time out. The Loader includes a solution to this. That solution is called "heartbeat". To learn more about heartbeat, see "Loader Heartbeat and AIJ Backup" on page 475.

Loader Expectations

In addition, please note that the Loader requires certain set-up and management.

- The AIJ files, including AIJ backups, must be preserved until processed.
- The Loader must be able to identify a row in the target to be able to do anything beyond inserts.
- The Control File, target, and installation must meet the standards described in this document.
- Rdb must be running on the computer that hosts the source and the computer that hosts the Loader. (For continuous operation, this will be the same system or the same cluster. For static LogMiner operation, they might be different.)

Limits

Loader families can be limited by how fast the target can accept the data changes. Performance can also suffer from limitations imposed by the network or other aspects of the environment. To date, such performance difficulties, that are beyond the Loader's control, are the only limit that is evidenced for properly tuned Loader families.

---

1. The difficulty with automated backups is also discussed in the chapter for Administrator "Automated AIJ Backups" on page 483 and the blog www.jcc.com/lml-abs-bad .
2. See "Knowing Whether the AIJ Is Processed" on page 417.

CHAPTER 4

Installation

This chapter tells you what you will need to know to install the JCC LogMiner Loader and how to acquire a copy of the Loader and of this documentation. It also discusses the JCC LogMiner Loader license key.

The JCC LogMiner Loader may be installed as the standard and only version or may be installed for multi-version operation, while continuing to use earlier versions.

This chapter also references blogs describing the interaction of the Loader with other software products and how to run the Loader for best results with different variants of the related products. See “Software Versions and Related Products” on page 49.

Additional preparatory steps are discussed in “Post-Installation Preparation” on page 99.

Getting a Copy

The JCC LogMiner Loader kit is provided as the following files.

- JCCLML_MM_mm_DOC.PDF
- JCCLML_MM_mm_pp_AXP.ZIP[1]
- JCCLML_MM_mm_pp_IPF.ZIP
- JCCLML_MM_mm_pp_axp.exe
- JCCLML_MM_mm_pp_ipf.exe

These are available from the yellow box in the upper right-hand corner of

http://www.jcc.com/lml

The first file is a PDF version of this document. You may read it on-line with Adobe Acrobat or you may print it. (Before printing, note that it is several hundred pages and includes color diagrams and color-coded examples.)

The next two files are OpenVMS zip archives, each containing an OpenVMS save-set of the JCC LogMiner Loader distribution.[2] The kit should be unzipped on OpenVMS. The last two files are the executables.

For a complete list of the kit contents, see “Kit Contents - Directories and Files” on page 573.

Privileges

JCC recommends that the installation be done from the system account.

The following privileges are required for running the Loader and for the installation and the IVP (Installation Verification Procedure):

1. Read “MM_mm_pp” as the Major, Minor, and patch designations of the version number.
2. Where the source database is on an alpha, you will need the AXP version. Where the source is on Integrity, you will need the IPF version.

- SYSNAM privilege to create system logical names
- PRMMBX privilege to create permanent mailboxes
- SYSGBL privilege to create a system global section
- PRMGBL privilege to create a permanent global section
- SYSLCK privilege to create system wide locks

Software Versions and Related Products

JCC's LogMiner Loader has been tested, over time, with many combinations of software versions. To acquire the most recent notes on versions and related products, consult the blog[1]
http://www.jcc.com/lml_prod_compat

Particular notes on JDBC targets are also included in
http://www.jcc.com/lml-jdbc-targets

Notes included in the blogs or in this document are correct to the best understanding available at the time they are written. However, JCC strongly suggests that you access the vendor's own documentation for notes on any software to support the target that you choose and/or on OpenVMS issues.

To properly test the Loader requires attention to operating systems and software for both the source and the target configurations. For more on the automated, randomized, continuing regression testing, see
http://www.jcc.com/lml-testing

When new versions of OpenVMS or Rdb are released, a Loader version that supports the new release is generally available within days. The Loader is tested on alpha, Integrity, and alpha emulators. JCC even maintains a VAX to test special architectures for retrieving data from a VAX. Indeed, one of the trickier questions is how far back we can go in offering support and you will see that reflected in our chart with notes on minimum versions.

1. The URL includes underscores that may be hard to see.

Testing source versions is a limited challenge, compared with testing versions for the target. Again, we have issues of both operating systems and end software. For Oracle targets, we even have issues of compatibility between the Oracle version available on OpenVMS and the Oracle version to be used with the target.

Restoring the Save Set

To begin, get the zip archives from the website and unzip them on the proper platform or get the self-extracting .EXEs. If you have not yet done these steps, see "Getting a Copy" on page 48.

If you will be running with multiple versions installed simultaneously, see "Multi-Version Support" on page 56 for comments on special multi-version installation.

The saveset file is a standard OpenVMS backup saveset. As such, it may be restored to any disk and directory tree required. Restoring this saveset will result in a directory sub-tree containing the JCC kit.

Create the Directory to Use

Work with the JCC LogMiner Loader will utilize an OpenVMS directory structure. Directories can be named to suit the installation and referenced appropriately. To create the option of having more than a single version of the Loader on a system at once,[1] JCC recommends creation of a top-level directory with a name that includes the software version. For example

```
$ create/directory <disk>:[jcclml_MM_mm_pp]
```

If, however, only a single version of the Loader will exist on the system at a time, creation of a directory without the version number will be adequate. For example

```
$ create/directory <disk>:[jcclml]
```

1. See "Multi-Version Support" on page 56.

Command for Installation

The OpenVMS command to restore the save set is:

```
$ backup jcclml_MM_mm_pp.sav/save_set <disk>:[<dir>…]/new_version
```

Instead of MM_mm_pp, use the Major, Minor, and patch designations of the version that you will be using. Instead of <disk>, use the relevant disk drive name. Instead of <dir> use the name of the top level directory selected for this data. The ellipses (...) and the "/new_version" are required. Failure to include these will fail to create the necessary directory tree.

Choosing the Directory

JCC recommends that the directory be specified in the root directory on the volume ([000000]). See also "Name Changes for Stored Procedures" on page 65 for a review of directory tips and "Startup and Directories" on page 53 for a discussion of the importance of following expected placement.

Choosing the Disk

The JCC LogMiner Loader files are not large. However, the Loader has numerous logging options. Depending on which of these options you find useful, the Loader may write voluminous log files. The disk you choose should have sufficient space to accommodate those files.

Loader log files can be re-directed by defining the logical name JCC_tool_logs in the Loader process context.

See also "Naming and Placing the Log Files" on page 460.

Loader Start Up

The start up command has two optional parameters.

Syntax

If you are running one version of the Loader and running on integrity or running on alpha with the default variant, you do not need either of the optional parameters.

```
jcc_tool_com:jcc_dba_startup [p1[ p2]]
```

P1. Parameter 1 establishes whether this is the standard (**S**) or variant (**MV**) version. Standard is the default. As described in "Multi-Version Support" on page 56, the Loader can be installed with more than one version active. That is indicated with P1, the first parameter of the startup command.

P2. Parameter 2 establishes whether the **Standard** (linked with p-threads), **EV6**, or **ST** (without threads) version is to be used. Standard is the default. See "Variants for Alpha" on page 52.

Variants for Alpha

Note that this section applies only to installations for which the Loader is run on alpha or an alpha emulator.

There are three possible variants for the Loader when running on an alpha.

- standard
- EV6
- ST (no P-threads)

Standard is required in some circumstances. EV6 is alpha architecture specific. ST may be desirable in some installations of the Loader and is the only valid variant with OpenVMS version 7.2-2 and earlier.

Certain products, Oracle for instance, require that the Loader be linked with p-threads. Newer versions of OpenVMS and Oracle use p-threads in a fashion that is not supportable with older versions of the Loader.

Since the Loader itself does not use p-threads, it is possible to link without them when using Rdb or API as a target and using Alpha. The version of the Loader linked without p-threads is called ST. Having it, protects backward compatibility for OpenVMS and it is recommended when using Rdb or API targets and Alpha.

To use the non-threaded images, the startup procedure must have the p2 parameter set to ST. The kit may be installed as either a standard or multivariant kit. The following example installs the Loader as multivariant and non-threaded.

```
@jcc_tool_com:jcc_dba_startup MV ST
```

Oracle targets are not compatible with the ST version. EV6 and ST versions are mutually exclusive.

Display

The variant is displayed wherever the Loader version is displayed.

Command Line Commands

Use the jcc_dba_startup procedure[1] to instantiate the JCC LogMiner Loader variant that best meets your needs. Once that is done, no other changes to your environment are required to run the instantiated images. The command line commands detect the instantiated variant and execute the correct images. Referencing special images directly is not supported. Instead, use these commands on the command line.

- jcc_version
- jcc_continuous_logminer_loader
- jcc_logminer_loader
- jcc_lml_dump_checkpoint
- jcc_lml_show_locks
- jcc_lml_statistics

Startup and Directories

Placement of the JCC_DBA_Startup procedure is critical. Logical names are based on the directory that is the usual location. Two behaviors can lead to invalid logical names.

1. The procedure is copied to another directory and executed.

1. See also “Set-Up for the Standard Version” on page 55.

2. The backup command for the Loader installation is executed without creating the full Loader directory tree.

Invalid logical names result from either case. Therefore, with current versions of the Loader, the procedure is enhanced to analyze the directory specification and alert the user if it is being run from an inappropriate directory tree.[1]

The JCC LogMiner Loader License Key

You will need a license key for the JCC LogMiner Loader. The license key will be created specifically for you by JCC. You must add this key definition to the Loader startup procedures after installation and prior to performing the Loader startup procedures.

Getting the License Key

There are two forms of license key that you might obtain.

- A permanent license key is available with purchase of a Loader license.
- The LogMiner Loader license scheme allows JCC to issue temporary licenses so that you may review the product before purchasing it. Should you receive such a temporary license, the Loader will broadcast warning messages to the system operator one week prior to the expiration date for the license. When you have examined the product and arranged the purchase with JCC, you will receive a permanent license.

Applying the License Key

The license key is in the form of an OpenVMS logical name. This logical name (JCC_LogMiner_Loader_Key) must be defined exactly as it is sent to you. Therefore, before the JCC Loader installation procedure is executed, copy the procedure <disk>:[<dir>.com]JCC_LML_license.com. Place it in the <disk>:[<dir>.local]

1. See “The JCC LogMiner Loader License Key” on page 54 for correct creation of the directory tree and Figure 1, “Directory Tree for Installation,” on page 56 for a schematic representation of the directory tree created.

directory. Edit the copy that you just made to replace the logical name definition with the key supplied to you by JCC. [1]

Groundwork for Communication

The examples in this chapter and those in other chapters require having run the procedure JCC_TOOL_COM:JCC_LML_USER. Following running this procedure, it is no longer necessary to use

- the DCL symbol "@" for JCC DCL procedures
- the run command for JCC executables

JCC_TOOL_COM:JCC_LML_USER is supplied with the kit. All processes that use any of the JCC LogMiner Loader tools will require that this command has been executed prior to the tool use.

Set-Up for the Standard Version

One of the restored files is a startup procedure. This is located in the [.COM] directory and is called JCC_DBA_STARTUP.COM. This procedure should be executed in the system startup procedure so that it is run each time the system starts.[2]

After restoring the save-set, the following command should be executed manually:

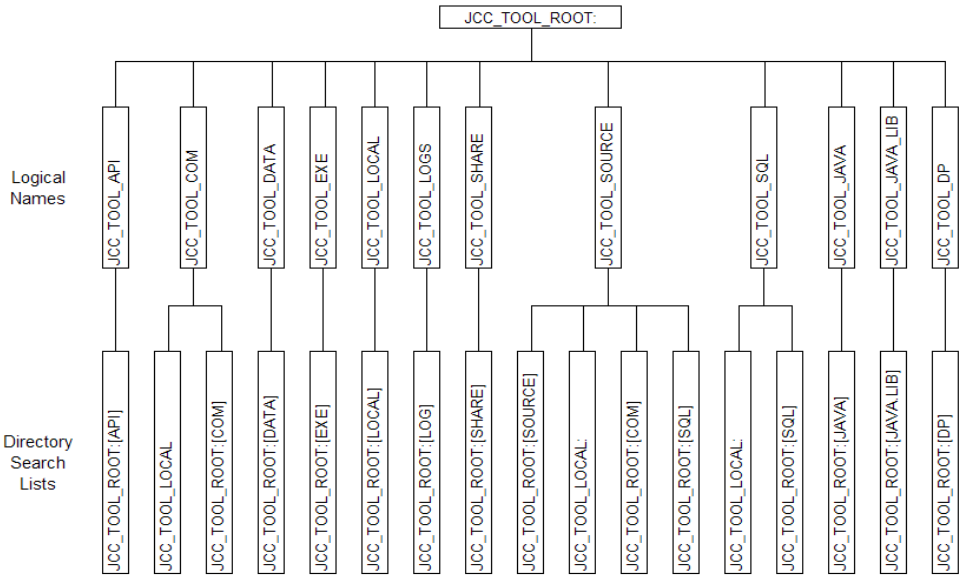
```
@<disk>:[<your directory path>.com]jcc_dba_startup
```

This command procedure will create several system logical names and will install several images. The command procedure establishes the directory tree shown in Figure 1, "Directory Tree for Installation," on page 56.

Do not move or copy this file to another directory before executing it. The startup procedures are self-referencing and must not be moved from the JCC directory tree.

1. See also "Multi-Version and the License Key" on page 57.
2. Startup will also need to be executed after any installation of a new version of the Loader.

FIGURE 1. Directory Tree for Installation



There is also an Examples subdirectory that includes examples for each of the targets. These are from JCC regression testing and are both complete and very well tested. Type the following to see what is available:

```
$ Dir JCC_TOOL_ROOT:[EXAMPLES...]
```

Multi-Version Support

The Loader supports installation and simultaneous use of multiple versions of the Loader.[1] Multi-version support makes it possible to test a new version of the Loader without disrupting current work.

If you only run one version at a time, you may skip this section in its entirety.

1. At most, one of the versions installed can be a pre-V2.0 version of the Loader.

Multi-version support requires an MV directory tree for each variant. Set up the MV directory tree and prepare to run multiple versions with the following steps:

1. Create an MV directory tree.
2. Install each version 2.0 or later Loader into the MV directory tree you will use with it. The following provides an example.

```
$ create/dir/prot=(w:rew) DPA100:[JCC_CLML_v2_0]
$ backup/log JCCLML_02_00_07.SAV/save DPA100:[JCC_CLML_v2_0...]*.*;/new_ver
%BACKUP-S-CREDIR, created directory DPA100:[JCC_CLML_V2_0.API]
     o
     o
     o
```

3. If one of the versions that you will be running is earlier than version 2.0 and you haven't already done so, copy two procedures from the MV directory tree to the standard directory tree.[1]
   - JCC_LML_USER.COM
   - JCC_ADD_LOGICAL.COM

See also "Manually Execute the Startup" on page 60 for the command to run the startup.

Multi-Version and the License Key

The startup procedures search for and execute the local startup procedure in the [...LOCAL] directory. If you are starting both a standard and a variant version of the Loader, then this procedure will be executed in the context of both startups.

The Loader license key logical name is defined within the context of the system logical name table. This means that the second definition will override the first definition. There is no problem with this if both definitions are correct. However, if a temporary license was used with one version of the Loader and a different version needs to be installed with a permanent license key, it may not be possible to do a multi-version installation in the manner described here. The solution is simple:

1. For example,

```
$ copy/log DPA100:[JCC_CLML_v2_0]JCC_LML_USER.COM PRINCE$DKBO:[LML.local}
%COPY-S-COPIED, DPA100:[JCC_CLML_v2_0]JCC_LML_USER.COM;8 copied to
PRINCE$DKB0:[LML_LOCAL]JCC_LML_USER.COM;8 (11 blocks)
$ copy/log DPA100:[JCC_CLML_v2_0]JCC_ADD_LOGICAL.COM PRIN-
CEE$DKB0:[LML.local]
%COPY-S-COPIED,DPA100:[JCC_CLML_v2_0]JCC_ADD_LOGICAL.COM;7 copied to
PRINCEE$DKB0:[LML.local]JCC_ADD_LOGICAL.COM;7(6 blocks)
```


Copy the permanent license key to the installation containing the temporary license.

Multi-Version and Swapping Between Loader Versions

When running with multiple versions of the Loader, you may need to toggle between versions. Do this by executing the JCC_LML_USER.COM procedure. This procedure accepts one parameter.

- If the parameter is 'S' or if none is specified, the standard version is used.
- For the MV installation, the parameter is the Loader version number. (For example, for version 3.0, the parameter is "3.0".)

This procedure

- Binds the correct set of logical names to each process.
- Appends the paths JCC_TOOL_EXE: and JCC_TOOL_COM: to the logical name DCL$PATH to simplify using JCC command procedures and executables.

The example shows swapping between two versions, one installed as the standard version and one that is a 3.0 version. The differences in input and output when toggling to version 3.0 are highlighted in red.

```
$ @jcc_tool_com:JCC_LML_USER s
Setting JCC LogMiner Loader version Standard
$ install list jcc_tool_share:jcc_logminer_loader_base_share.exe

DISK$RAID0-07:<JCC_CLML.SHARE>.EXE JCC_LOGMINER_LOADER_BASE_SHARE;1
                    Open    Shared        Lnkb1
      o
      o
      o
$ @jcc_tool_com:JCC_LML_USER 3.0
Setting JCC LogMiner Loader version 3.0
$ install list jcc_tool_share:jcc_logminer_loader_base_share.exe

DISK$RAID0-07:<JCC_CLML.SHARE_V3_0>.EXE JCC_LOGMINER_LOADER_BASE_SHARE;1
                    Open    Shared        Lnkb1
```

Tailoring Procedures

In the figure “Directory Tree for Installation” on page 56, note the JCC_TOOL_LOCAL directory. New releases of the Loader are installed “on top” of existing versions and overwrite all JCC files *except* the local directory. Procedures that you have tailored and placed in this directory will not be overwritten. Note that it is searched first.

However, if you use variants[1] of Loaders in different directories or use multiple versions[2] simultaneously, there are some added complications. For variants, the logical name JCC_TOOL_LOCAL will differ across variants. If you migrate a Loader family from one version to another, you will find a need to change file location or copy some files. An alternative with the advantages of using JCC_TOOL_LOCAL that avoids changes when varianted versions are used and their status changes is to create a directory which you can use for files specific to a single Loader family. Then, only one edit is required to change to a new version.

Installation in a Cluster

The Loader may be installed in a cluster if the installation directory is accessible by all nodes in the cluster.

If the cluster has a mixed architecture, it will be necessary to install multiple kits, one for each architecture.

System Startup

The system startup procedure must be modified to include starting the Loader.

For the first time that the Loader is run, the command can be given manually as shown in “Loader Start Up” on page 51.

1. See “Variants for Alpha” on page 52.
2. See “Multi-Version Support” on page 56.

The following example for inclusion in system startup assumes that you installed into [JCC_CLM] as the root.[1]

Modify System Startup

Edit system startup to include starting the Loader. At system boot time, execute JCC_DBA_STARTUP.

```
$ @DPA100:[JCC_CLML.COM]JCC_DBA_STARTUP
JCC Tool Root will be: DPA100:[JCC_CLML.]
    o
    o
    o
```

Manually Execute the Startup

When your other preparations are complete, manually execute what you have added to the system startup, namely:

```
$ @DPA100:[JCC_CLML.COM]JCC_DBA_STARTUP
JCC Tool Root will be: DPA100:[JCC_CLML.]
    o
    o
    o
```

Multi-Version

If you are running more than one version of the Loader simultaneously, you will need to add the startup for each version to the system startup. For example, to start version 3.5 as an alternate version, you will need to add

```
$@DPA100:[JCC_CLML_V3_5.COM]JCC_DBA_STARTUP MV
JCC Tool Root will be: DPA100:[JCC_CLML_V3_5.]
    o
    o
    o
```

You will also need to execute the command manually when you have completed your preparations.

In both systems startup and your manual execution use whatever version is appropriate instead of the “3_5” that is shown.

```
$@DPA100:[JCC_CLML_V3_0.COM]JCC_DBA_STARTUP MV
JCC Tool Root will be: DPA100:[JCC_CLML_V3_0.]
```

1. See Figure 1, “Directory Tree for Installation,” on page 56 for a diagram of the complete directory tree established.

-
-
-

Installation Verification Procedure

A small installation verification procedure (IVP) is provided with the kit. You should run this to ensure that the kit is installed properly.

To run the IVP execute the following command:

```
$ jcc_logminer_loader_ivp
```

Note that this IVP does not attempt to perform a complete verification of all functional components of the kit. It uses the sample personnel database defined by the Rdb kit and slaves a second copy of that database to the first. It then performs a few updates and runs the LogMiner and the Loader to load that second copy with changes made to the first. Finally, it performs differences on the tables.

Not all issues are resolved by the IVP, but additional protection is added, as issues are discovered.

If the user defines a symbol that modifies the delete command such that some files are not deleted, the IVP overrides the user definition.

The IVP provides two additional checks: Run Local and Run User Procedure.

Run Local

The Loader checks to ensure that the IVP is running from a locally mounted disk to provide support of creating and altering the databases required for the IVP. If the IVP is not running from a locally mounted disk the following message will be displayed.

```
This procedure must be run on the same node as that
on which the Loader software is installed.
Local node: <current computer>
Installed node: <remote mounted disk node>
```


Run User Procedure

The Loader verifies that the JCC_LML_USER procedure has been executed prior to running the IVP. This ensures the appropriate environment to perform the requested verification. If the procedure has not been run, the following message will be displayed.

```
This procedure requires the JCC LogMiner Loader
environment be set. Please execute jcc_lml_user
procedure for this process before running the
the IVP.
Use either:
$ @jcc_tool_com:jcc_lml_user
or, for multiversion support:
$ @jcc_tool_com:jcc_lml_user <LML version>
```

```
Creating the originating database and enabling logminer

Backing up database that was just created

Restoring the backup to establish the target database

Removing constraints and triggers in the target database

Adding high-water table to target database

Adding DB-key column to tables

Creating JCC LogMiner Loader control file
    1       !
3997 substitutions
5 substitutions
No substitutions
JCC_IVP_ROOT:[IVP.WORK_FILES]MF_PERSONNEL.INI;2 73 lines
%PURGE-I-FILPURG, JCC_IVP_ROOT:[IVP.WORK_FILES]MF_PERSONNEL.INI;1 deleted (525 blocks)

Creating Oracle LogMiner options file
%PURGE-I-FILPURG, JCC_IVP_ROOT:[IVP.WORK_FILES]MF_PERSONNEL_LM_UNL.OPT;1 deleted (525 blocks)

Backing up the AIJ file now

Running Oracle LogMiner
%RMU-I-NOSEGUNL, Table "RESUMES" contains at least one segmented string column
%RENAME-I-RENAMED, JCC_IVP_ROOT:[IVP.WORK_FILES]MF_PERSONNEL.RDB_LOGMINER_UNLOAD;1 renamed to
JCC_IVP_ROOT:[IVP.WORK_FILES]MF_PERSONNEL_01.JCC_LOGMINER_LOADER_INPUT;1
%DELETE-I-FILDEL, DISK$USER:[JCC]MF_PERSONNEL.LM_OPT_2002-02-18_11_08_41;1 deleted (35 blocks)

Starting comparison of the two databases
Files seem to compare --> JCC_IVP_ROOT:[IVP.WORK_FILES]WORK_STATUS.DIFF;
Files seem to compare --> JCC_IVP_ROOT:[IVP.WORK_FILES]EMPLOYEES.DIFF;
Files seem to compare --> JCC_IVP_ROOT:[IVP.WORK_FILES]JOBS.DIFF;
Files seem to compare --> JCC_IVP_ROOT:[IVP.WORK_FILES]DEPARTMENTS.DIFF;
Files seem to compare --> JCC_IVP_ROOT:[IVP.WORK_FILES]JOB_HISTORY.DIFF;
Files seem to compare --> JCC_IVP_ROOT:[IVP.WORK_FILES]SALARY_HISTORY.DIFF;
```

FIGURE 2. Sample Run of the IVP

When the IVP has been successfully run, you are assured that all of the files are present that you need to start work with your own database and the Loader.

The IVP and the Rdb Version

If you are running multiple versions of Rdb on your system, you should be aware that the IVP will run under your current default version. You may display this version with the command:


```
$@sys$library:rdb$shover
Current SYSTEM Oracle Rdb environment is version V7.1-411 (MULTIVERSION)
```

If you wish to run the IVP using a different version, you should set your Rdb version appropriately. Please note that the IVP does not make any attempt to validate that you are running under a version that supports LogMiner.

Cleaning Up After the IVP

The IVP generates a number of files and databases during its execution that are specific only to Rdb‘s sample Personnel database. Because these files can serve as examples for constructing your own Loader activities, they are left on your system.

You can remove these files from your system by executing the following command:

```
$ clean_up_ivp
```

Exception Messages

For a list of exception messages, their meaning, and what action can be taken to rectify the problem, see the file

`jcc_tool_source:jcclml_msg.doc`

Notes for the Systems Manager

Collected here are a few comments that come from answering systems questions and tuning the Loader to specific installations.

Quotas

The VMS System Service Manual documents the following as quotas that are pooled.

- BYTLM
- ENQLM
- FILLM

- PGFLQUOTA
- PRCLM
- TQELM

If you are running in a tightly constrained system with other jobs running, you may need to tune these. PGFLQUOTA is a particularly important parameter, as it is shared between all processes in a Loader family. If your transaction size is large or if you use multiple Loader threads, it is important that you tune this value.

Directory Security

The Loader provides some security for directory trees and provides the option of site specific changes.

The default security eliminates world write capability to most of the files and directories in the Loader directory tree.

You may set the security model for the Loader directory tree by using the procedure jcc_tool_com:jcc_tool_security.com. If the default security model for the Loader is not acceptable, then jcc_tool_com:jcc_tool_security.com should be copied to jcc_tool_local:jcc_tool_security.com and modified to provide the required security. If jcc_tool_local:jcc_tool_security.com exists, it will be used during startup instead of the default procedure jcc_tool_com:jcc_tool_security.com.

Name Changes for Stored Procedures

If the JCC LogMiner Loader has been in use at your site since before version 3.3.1, you may want to be aware of some deprecated names.

Executing the procedure JCC_TOOL_SQL:VMS_FUNCTIONS.SQL against the source database is optional, but enables additional functionality. With Release 3.3.1 of the JCC LogMiner Loader, these procedures are replaced (and renamed) to avoid overlap with different procedures with the same name that are in use at some Loader sites. The affected procedures are:

- JCC_CREATE_LOG_MINER_OPT_FILE.COM
- JCC_CREATE_LOG_MINER_TUX_FIELD_DEF.COM
- JCC_GENERATE_ORACLE_TABLE_SQL.SQL

If you require the functionality of these procedures, then the new version of JCC_TOOL_SQL:VMS_FUNCTIONS.SQL must be applied to the source database.

Note that the original procedures (named GET_SYMBOL and SET_SYMBOL) may be deleted, if desired. They are not deleted by Loader procedures due to naming overlaps at some Loader sites.

Review and Tips — Directories and Files

A bit of review of directory placement may help forestall problems. Here are some directory tips:

- Install the Loader in the root directory on the volume that you will be using. The resulting directory tree is shown in Figure 1, "Directory Tree for Installation," on page 56.
- Loader log files can be re-directed by defining the logical name JCC_tool_logs in the Loader process context.
- Copy <disk>[<dir>.com]JCC_LML_license.com, place the copy in the <disk>[<dir>.local] directory and edit the copy to replace the logical name definition in that procedure with the one provided by JCC as your license key.
- Also use the <disk>[<dir>.local] directory for any special purpose procedures that you add. Many of the Loader's logical names are search lists which translate to the local directory first and then to the generic directory. Special procedures which are not placed in the local directory may be overwritten by a later patch or upgrade of the Loader.
- Use the startup command without moving it to another directory. See "Startup and Directories" on page 53.
  ```
  @ddcu:[<your directory path>.com]jcc_dba_startup
  ```
- Create extra directory trees when using variants.
- Use the logical name JCC_AIJ_BACKUP_SPEC to specify to the Loader where backups may be found for restart. This logical name may be defined as a searchlist and a searchlist with wildcards may be required to find all the backup files, particularly if your AIJ backup files bear multiple naming conventions.

Notes for the Loader Administrator

Different installations require different precautions and tuning. See “Aids for the Administrator” on page 409 for an assortment of topics.

One topic that is critical to all installations is management of the AIJ files. See the sections “The After Image Journal - AIJ” on page 416 and “Safety Test for AIJ Backup” on page 418 and follow the practice described there in “Use” on page 419.

Defaults

There is a great deal of flexibility in the Loader. Keywords in the Control File and logical names give you the opportunity to modify how the Loader works and what it does. This can be a powerful advantage in solving your particular issues.

The Loader also includes carefully chosen defaults. These defaults are often the correct choice for your application.

Without understanding the options and the advantages and disadvantages to the option chosen, overriding the defaults can lead to unexpected behavior and confusion. The Administrator should rely on the default unless there is a clear need for another alternative.

CHAPTER 5

Continuous LogMiner and the Loader

Coupled with Rdb's Continuous LogMiner, JCC's LogMiner Loader can run continuously.[1] In continuous mode, transaction commits in the source database initiate work in the LogMiner which initiates work in the Loader. Changes to the source are published to the target in "near realtime." The LogMiner Control Process manages it all.

Other modes are discussed in "Modes of Operation" on page 77.

"Near Realtime" Operation

The *Continuous LogMiner (RMU CLM)* processes data from the AIJ file(s) when a transaction is committed. The *Continuous LogMiner Loader Control Process* coordinates the interaction of the CLM and the Loader by causing the CLM to output to an OpenVMS mailbox. The Loader takes the CLM output and — under the direction of your Control File — writes it to the target.

1. Continuous operation requires the Rdb Continuous LogMiner kit (in Rdb 7.0.6.4 or 7.1.0.2 or later) from Oracle and the JCC LogMiner Loader kit (version 1.2 or later) from JCC.

The CLM and the Loader are designed to run and keep running. Continuous operation is the norm.

The operation can be almost realtime, but is never quite realtime (unless nothing is happening). The Continuous LogMiner (CLM) doesn't produce output until a transaction is committed and the LogMiner Loader (LML) cannot process the CLM's output until it is available. Processing will often be perceived as realtime.

CLML Architecture

The CLM may read either from backup AIJ files or the live AIJ files. During catch up, it will read from the backup AIJ files and switch to the live AIJ files when all backup files are completed. It processes the backup AIJ files in the correct journal sequence. (See "Finding AIJ Backups" on page 71 and "Shutting Down Continuous LogMiner" on page 75.)

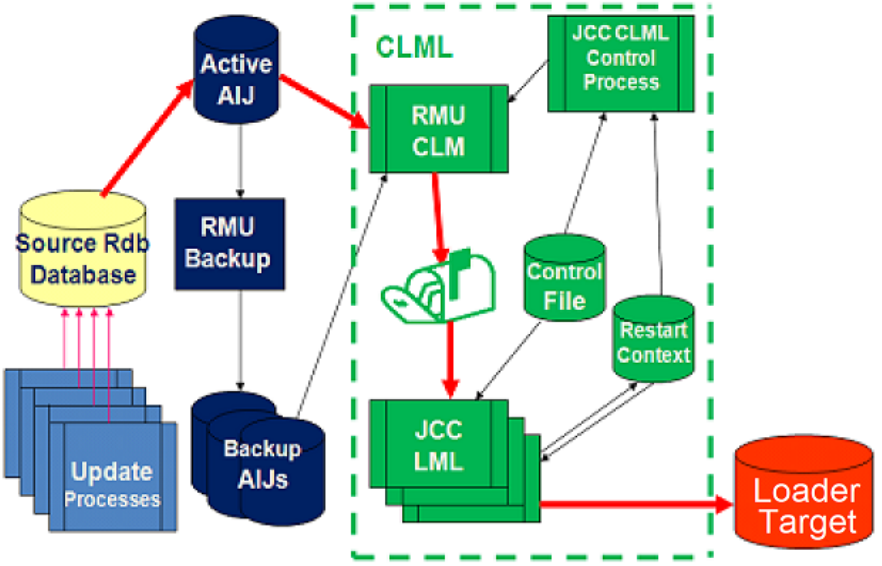


FIGURE 1. .CLML Architecture

The heavy red arrows in the figure show the flow of data during normal operation.

Of course, some aspects of the diagram are simplified. You can have multiple Loaders running at the same time and each of them can have one or many (up to 32) parallel threads. See “Multiple CLM Processes” on page 71 and “Parallelism and Loader Threads” on page 383 for more on these topics. Also, the Loader target may be a database itself or may be a transport to an end target.

Multiple CLM Processes

If you wish multiple CLM processes to read from the same AIJ backup files at the same time, define the logical name

```
$ DEFINE JCC_ADD_CLM_SHARED_READ TRUE
```

Finding AIJ Backups

The AIJ backup files must be specified to the LogMiner. This is usually done by specifying a logical name containing wildcards which appropriately map the backed up AIJ files. For backup AIJs in multiple directories, use a search list form of the logical name to specify the files in the different directories. Examples are:

```
$ define aij_dir disk1:[dir1], disk2:[dir2]
$ define jcc_aij_backup_spec aij_dir:*.aij
```

See also “Searchlist for AIJ Backup Files” on page 417.

Enabling Continuous LogMiner

Before you begin, the source database needs to be enabled for Continuous LogMiner. To do this use[1]

1. See also “Enabling the LogMiner” on page 102 and do not forget the follow on steps described both here and there.


```
$ rmu/set logminer /enable/continuous <database name>
```

You should backup the AIJ after issuing the command, You should also remove all AIJ backups from the AIJ backup directory after configuring the database.

Running Continuous LogMiner and the Loader

There are some requirements that are specific to running the *continuous* LogMiner and the Loader.

Required Privileges

The account under which the Loader is run must have SYSNAM and PRMMBX privileges or the equivalent.

Other Requirements

To run the Continuous LogMiner requires:

- A version of Rdb that supports the Continuous LogMiner. That's version 7.0.6.4 or version 7.1.0.2 or later.
- Multiple journals. That is, if you are running with a single file AIJ, you must establish multiple journals. This may require an export-import of your database. JCC recommends that, if you must make this change, you go to a multi-file model for ease in future management.

To run the JCC Continuous LogMiner Loader requires version 1.2 or later.

Procedure

To run the Loader starting in the active AIJ, use the syntax

```
$ JCC_RUN_CLM_LML                    -
<source database name>               -
<LogMiner options file>              -
<LogMiner Loader Control File>
```

This is also the command for resuming a session.

Parameters

Each of these parameters is required and must point to a valid file.[1]

<Source Database Name>. The source database name specifies the name of the source database that the continuous LogMiner is running against.

<LogMiner Options File>. The LogMiner options file is the name of the LogMiner options file that describes the tables to be mined. The LogMiner options file is required by the CLM and must contain a complete specification for the output. For use with the Loader, the options file specifies that the LogMiner should write everything that it writes to the same output file. An example of the LogMiner options file might be

```
table=account, output=rdb_logminer_output_file
table=account_contract,output=rdb_logminer_output_file
table=bill, output=rdb_logminer_output_file
table=customer, output=rdb_logminer_output_file
o
o
o
```

See "Rdb LogMiner Options File" on page 102 for a discussion of using the procedure provided with the kit to generate a template options file for your source database, using

```
$ JCC_CREATE_LOG_MINER_OPT_FILE <source database>
```

<LogMiner Loader Control File>. The LogMiner Loader Control File must specify the name of the target database. It also specifies many additional aspects of how the Loader session should operate. See "Control File" on page 217.

Overrides

There are, actually, three optional parameters for the run command. These are not covered here because they should *NOT* be used in most circumstances.

1. In Copy and Static mode, this same command is used. Even though each of the parameters do not always point to something that is actually required as input, the parameter must still be specified and must point to a valid file. See also "Running the LogMiner Loader" on page 84.

In certain limited circumstance, it is necessary to override[1] the default behavior built into the Loader. These circumstances are:

1. If the DBA is doing a big database reorganization that is not journaled, the Administrator must ensure that all LogMiner activity is completed before starting the reorganization. The Administrator can, then, shut down the Loader and LogMiner, backup the journal, and restart (in the live journal) using the overrides.[2]
2. If a development environment has gotten thoroughly confused, it may be necessary to use the override.
3. If you need to start for the *first time* on a source database and need to start in the *backed up AIJs.*[3]
4. For special applications that do not need "old" data.

In all other circumstances, JCC advises ignoring the override parameters.

Log Files

The Control Process acts as the logging sink for the CLM and LML processes. Each of these sends data to the standard sys$output and sys$error, using a mailbox for each process. The Control Process writes this data to the log files.

- jcc_tool_logs:jcc_run_clm-<loadername>.log
- jcc_tool_logs:jcc_run_lml-<loadername>_n.log, where n represents the thread[4] number (using 0-9 and a-v)

Re-opening the Logs

You may request that the CLML Control Process re-open — or close and re-open — the log files from the LogMiner and Loader processes. To do so, use the following command.

```
$ JCC_CLML_REOPEN_LOG <loadername>
```

1. See "Overrides in the Run Command" on page 427, *IF* circumstances 1 or 2 apply.
2. See "Other Overrides" on page 432.
3. See "Starting for the First Time in the Backed Up AIJs" on page 430.
4. Threads support parallel Loader streams and are discussed in numerous contexts later in this document. In particular, see "Parallelism and Loader Threads" on page 383.

Shutting Down Continuous LogMiner

The Continuous LogMiner and Loader are designed to run and keep running. Occasionally, though, you will want to shut things down. It may not even be that the LogMiner or the Loader develop a problem. You may have a maintenance reason for shutting down, such as down-time for the source database; or whatever is receiving the data may develop a problem. In any case, there must be a way to shut down.

In general, you should not terminate the Loader or LogMiner jobs manually. For predetermined shutdown, you should use the procedure supplied with the JCC LogMiner Loader kit.

Shut Down Command

To shut down, use the command

```
$ JCC_CLML_SHUTDOWN <loadername>
```

The Loader works in an environment that includes other products. Sometimes, there are issues with the other products or with environmental conditions that cause the Loader to stall. Should a stall condition occur due to circumstances in the environment or in other products, your manual intervention may be required.

Control Process Follow Up

Whether the CLML is shut down on purpose or through some exception condition, it is necessary that the shut down be as graceful as possible. If either CLM or the Loader fails, the Control Process shuts them both down and drains and closes the mailbox that they use to communicate. The Control Process shuts the operation down in a fashion intended to support resumption of activity at a later time.

Managing AIJs During Shutdowns

Note that AIJs are required for the LogMiner and the Loader to work. If shutdown conditions require that the active AIJ be backed up, the backed up AIJ files must be retained on-line and must be available in the path defined by JCC_AIJ_BACKUP_SPEC. See also “Finding AIJ Backups” on page 71.

Determine AIJs Needed for Restart

Determine which backup AIJ files are no longer needed by examining the AERCP that is current in the LogMiner. The AIJ sequence number embedded in this value represents the earliest AIJ file required for restart. In order to ease this inspection, both the JCC_LML_statistics utility and the JCC_LML_dump_checkpoint (in writing to the log) will format and display the current AERCP. See “Online Statistics Monitor” on page 314 and “Displaying Checkpoint Information” on page 372.

Restart/Recovery

See “Restart, Recovery, and Shutdown” on page 420 in the chapter for Administrators for more information.

CHAPTER 6

Modes of Operation

The Oracle Rdb LogMiner and the JCC LogMiner Loader are, generally, run in continuous mode. Continuous mode is discussed in detail in "Continuous LogMiner and the Loader" on page 69. Continuous Mode is not, however, the only option.

This chapter discusses each of the modes and the factors that determine which mode is used, as well as the set up and execution for each.

The Oracle Rdb LogMiner and the JCC LogMiner Loader are designed to work together. How they work together is determined by the mode chosen. The LogMiner output provides the input to the Loader. The LogMiner can use only the backed up AIJ files or can use backed up AIJ files and active AIJs. The LogMiner can write to a file or to a mailbox. The LogMiner can include a record for each commit or not. The mode determines which of these choices are suitable for the LogMiner, as well as how the Loader utilizes the LogMiner output.

History

A bit of history may help if you have or run into older uses of the LogMiner and Loader.

Original

Static mode was all that was available in Version 1.0. For the initial release, the LogMiner used the backed up AIJs to mine the database changes. The LogMiner wrote data as output to an RMS file, the *unload file*. The Loader read this file and sent the data to a target.

The Static LogMiner and Static Loader were batch-like and single threaded. Static mode is asynchronous with whatever is happening in the source database.

For the Original Static Mode, the LogMiner is instructed to write the unload file with binary data so that the Loader can extract null information. Binary data has the added advantage that, if the target is an Rdb database, the Loader can avoid data-type conversion.

Commit records were not available from the LogMiner, with the early releases, and the Loader determined the end of a transaction by recognizing that a new TSN (transaction sequence number from Rdb) had appeared.

The original approach to running Static LogMiner[1] and Static Loader is not recommended for new applications, as there have been significant improvements in the options.

Continuous Mode

Subsequently, Rdb Engineering and JCC worked together to develop the Continuous LogMiner (Rdb) and the Continuous LogMiner Loader (JCC). This combination provides near realtime[2] updates to the target, synchronous with commits on the

1. Static LogMiner is still used as a component of Copy mode use of the Loader.
2. JCC uses the term “near realtime” rather than “realtime” to describe operations because the LogMiner cannot process the information until the source database commit is seen. See also “Performance” on page 18.

source. The continuous LogMiner and Loader are coordinated by the Loader Control process and communicate binary output via an OpenVMS mailbox. Commit records are included in this stream.

The JCC LogMiner Loader Control Process manages ongoing operation and shutdown and restart such that operation can run in the active AIJ or can start in the backup AIJ(s) and continue into the active AIJ until caught up.

The Loader Control Process (CLML) also manages logging and exception responses and multiple assists to the Administrator.

Continuous mode was introduced with version 2.0 of the Loader and is discussed in detail in “Continuous LogMiner and the Loader” on page 69.

Copy Mode

Copy mode was introduced, for testing, with version 2.2.3 of the Loader and improved for full production use with version 3.1.

Copy mode is a hybrid. For Copy mode, the static LogMiner is run on the backed up AIJs to produce an output file, the “unload file.” In Copy Mode, the Loader control process starts a special copy program to write the unload file to the Loader OpenVMS mailbox *as if it were output directly from the Continuous LogMiner.*

With Copy mode, the Loader can be run repetitively on the same LogMiner output. The repeatability, coupled with Loader monitoring and tuning tools makes it a powerful testing tool. Copy mode has also proven valuable in situations that require some asynchronicity between the source and target. Details of these uses are discussed in “Use” on page 82.

Improved Static Mode

Static mode was also improved as part of the modifications to Copy mode and no longer requires the unload file. If the static LogMiner and the Loader are run on the same platform or have an adequate network, the static LogMiner may be run under the Continuous LogMiner Loader Control Process which will cause it to write directly to the Loader’s OpenVMS mailbox. Using the Static mode in this fashion, enables performance tuning options, including the Loader’s parallel thread capabilities.

Original Static Mode is no longer recommended for *new* uses. Support continues and current uses are upwardly compatible. In this chapter, the original style of Static Mode is specifically referred to as “Original Static Mode” while “Static Mode” implies the improvements present in Version 3.1 and later of the Loader.

LogMiner

The LogMiner output is, directly or indirectly, input to the LogMiner Loader. The LogMiner may be run, directly, in static mode or run via the Continuous LogMiner Loader control process.

The diagram shows the operation of the LogMiner without showing the Loader Control Process or the destination of the LogMiner output.

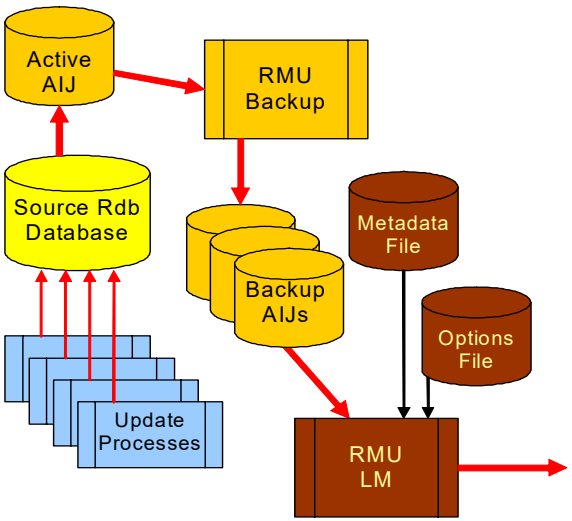


FIGURE 1. Static LogMiner

The diagram shows several blue update processes and one yellow source database. In most environments, journaling to the active journals occurs and RMU backup

backs up the live journals to backup journal files.[1] This activity is shown in gold boxes.

The LogMiner activity is shown in brown in the diagram. The options file determines the activity of the LogMiner and is required for operation of the LogMiner. The metadata file is required when the LogMiner is run on a computer where the source database is not open. It provides metadata definitions of the source database so that the LogMiner can be run without immediate access to the source database.

Static LogMiner Loader

The static LogMiner Loader with improvements continues to be a valid choice for some installations.

Use

In 2001, JCC developed the JCC LogMiner Loader to be able to reorganize a database that was in serious need of attention, but was so central to the business that any disruption in service was critical. Static mode was used for this. Since then, both the LogMiner and the Loader have been improved. The approach that JCC recommends, today, for database reorganization relies on the Continuous LogMiner Loader. A simple set of steps is described in “Example: Reorganizing an Rdb Database” on page 529.

A continuing use for static mode is for help with older versions of Rdb. Continuous LogMiner was not introduced until June, 2002 in Rdb V7.0.6.4. For a version older than that, it is not possible to use Continuous LogMiner which means that it is not possible to use Continuous LogMiner Loader.

Architecture

When the (new) Static LogMiner Loader is used, the Loader Control Process manages both the Static LogMiner and the LogMiner Loader. Information is supplied by the LogMiner to the Loader with an OpenVMS mailbox.

1. Journaling is a requirement for the LogMiner and is highly recommended for any database.

Static Mode Exceptions

Because most use of the Loader relies on Continuous mode, Loader processes and even this documentation are oriented to Continuous mode. Two specific exceptions are cited here.

When the Loader displays the Rdb version being used, for static mode, the Loader is not checking the database for the version, but is checking the version that the process is using.

The Loader supplies the AERCP to the static LogMiner to support restarts. Versions of Rdb prior to 7.2.4.0 do not accept this qualifier as valid for the static LogMiner and fail with a message citing " illegal combination of command elements." To run Release 3.3.0 and later of the Loader with Rdb versions prior to 7.2.0.4, define the logical name JCC_CLML_REMOVE_STATIC_AERCP to 1.

Copy Mode

Copy mode is a hybrid. The Static LogMiner is run to produce the unload file and the Loader makes use of the unload file as if it were a live feed from the LogMiner to the mailbox.

Use

To run in continuous mode — and, therefore, to use parallel Loader threads and other near realtime features — has required a source database that provides transactions from live journal files. This can be inconvenient for testing, both because of potential impact on the source and because workloads are not exactly repeated. In some circumstances, it can be inconvenient for production application.

Copy mode is a hybrid which supports using the static LogMiner (with AIJ backup files) and processes (copies) the LogMiner unload file to use the result as the source of the transactions for the Continuous LogMiner Loader.

Copy mode has been used in two important fashions.

- To provide the LogMiner and the Loader for on-going use, in an environment in which the source and the target cannot reliably be continuously synchronized. The reasons to disassociate the source and target might be a collection of

  - High data volumes coupled with a slow network
  - An unreliable network
  - Inconvenience (or political difficulty) for coordinating systems management, database administration, and/or down times for the source and the target environments.
  - Security concerns
  - A lack of need for absolutely up-to-date data in the target or a positive need for planned and known refresh times.
- To synthesize running Continuous LogMiner and the Loader in order to test or tune an architecture and/or target-based ("down stream") procedures.[1] To benefit from a testing tool that is
  - Repeatable
  - Tunable
  - Does not involve the source in repeated trials
  - Uses actual data and actual data volumes and patterns[2]

Architecture

Copy mode relies on the Static LogMiner to produce an output file, the unload file. The unload file is copied by a special Loader procedure and used to run the Loader *as if* it represents live actions on the source database.

It can be illustrated as shown in the diagram in Figure 2 on page 84. The red arrows show the main flow of information. the black arrows show additional input and information exchange. The target for the Loader operations may be a database - such as Rdb or Oracle - or a transport - such as JDBC, XML, Tuxedo, or Kafka - that writes the changes to an end target.

1. See "Throttling the Loader" on page 480.
2. The Loader replays the write operations and can be set to replay them as they occurred or faster or slower than they occurred or at some artificial time sequence. Copy mode, of course, does not include read operations that might have been occurring and so is not a perfect test of performance.

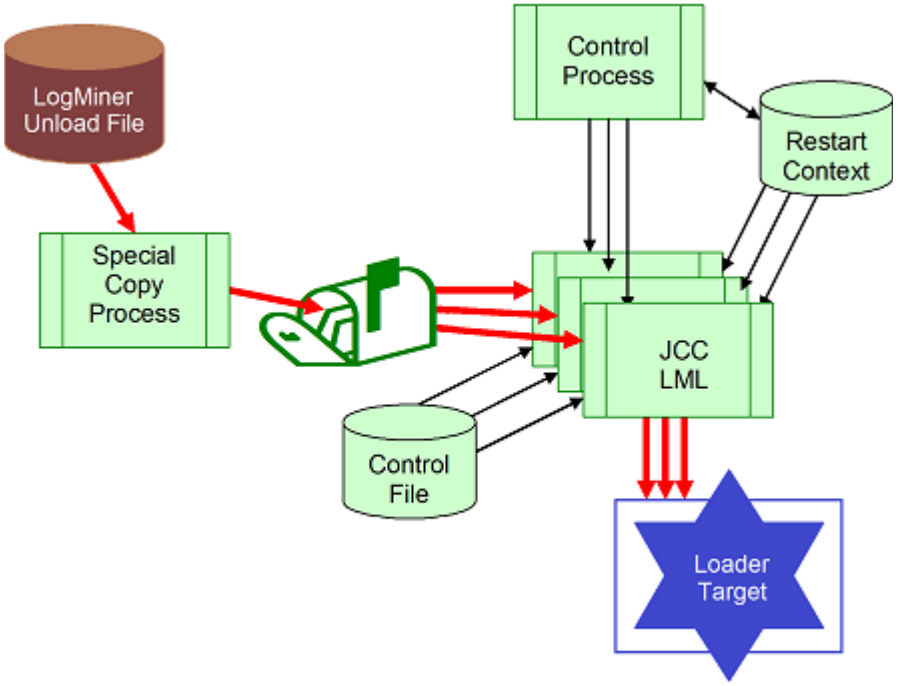


FIGURE 2. Copy Mode LogMiner Loader

Which Mode to Use?

Continuous LogMiner and the Continuous LogMiner Loader are the default. “Use” on page 82 describes conditions which warrant use of Copy mode. Static mode has been used for database reorganization and continues to be useful for older versions of Rdb.

This section is intended to clarify the differences, advantages, and disadvantages of the three.

Distinguishing Static and Continuous Mode

The Static LogMiner is different from the Continuous LogMiner in that

- Only the backed up AIJ files are used.
- The LogMiner does not control the AIJ order, *unless the qualifier '/order' is added to the run command*. Therefore, you will need to add this qualifier when using the static LogMiner.
- The LogMiner cannot signal discontinuities (from AIJ to AIJ) between runs, although it can signal discontinuity within a single run.
- The metadata file is required (if the LogMiner is run without access to the source database). (See "Metadata for the LogMiner" on page 89.)

Loading Performance and Flexibility

All of the advantages of performance tuning and all other features described in this document apply to Continuous mode. Because Copy Mode and Static Mode (with the improvements) imitate Continuous Mode in the loading, they share the performance tuning aspects of Continuous. However, care must be taken when increasing the commit interval. See "Prepare for EOF" on page 91.

Whether Copy Mode or Static Mode provides the best performance depends on the environment.

In situations where the target is not on the same cluster as the source, the unload file has the advantage of being smaller and more compact than the AIJs. Therefore, running the LogMiner to produce the unload file and using it at the remote site that hosts the target helps to reduce network challenges. Then, the Copy mode can be used to apply it to the target.

Where the network is not an issue, the Static Mode has performance advantages. Copy Mode requires that the unload file be written to disk and then read from disk. This makes Static Mode more efficient if the AIJ backup files are available on the node where the LogMiner and Loader are to be run. Also, in Static Mode, the LogMiner output is written directly to the mailbox as each transaction is committed making it available for the Loader to read and process immediately, whereas Copy Mode is delayed until the LogMiner is finished and the Unload File is processed.

Completeness and Timeliness

Of course, the most significant differences are that *only* the Continuous LogMiner and Loader include the active AIJ and keep pace with changes to the source database. Therefore, only Continuous Mode is on-going, although the other modes can be run again to capture additional source database changes.

Control

Copy Mode requires running the LogMiner manually. Static Mode, Copy Mode, and Continuous Mode, all process the data changes under the Control Process of the Continuous LogMiner Loader.

Greater care is required in managing an application which relies on Static or Copy Mode, if processing is not a one time event. Unlike Continuous Mode, Static and Copy were not designed to be on-going. See “Checkpointing and Discontinuities” on page 92.

Distinguishing Static and Copy

Static mode (when writing to a mailbox) is run under the control of the CLML Control Process. Running of the LogMiner is transparent to the user. Copy mode is under the control of the CLML Control Process while loading, but the LogMiner must be run in Static mode to supply the unload file that the Loader will use.

As stated in “Loading Performance and Flexibility” on page 85, which of these modes has the best performance results depends on the locality of the data and the goals to be met. Copy mode has efficiencies in moving data from one site to another. Static Mode, when local to the backup files, avoids disk writes and avoids waiting for the LogMiner to finish all work before the Loader can begin processing transactions.

If testing is the goal, the deciding factor is that Copy Mode can be repeated over and over with the same unload file without repeating the overhead of running the LogMiner on the AIJ files.

Summary

For reference purposes, the features, uses, and issues of the modes are summarized in “Summary of the Characteristics of Modes” on page 87. The Original Static

Mode and Continuous are included for reference, but the focus of this chapter is Static (with improvements) and Copy.

TABLE 1. Summary of the Characteristics of Modes

Characteristic	Original Static	Continuous	Static	Copy
Default?	no	yes	no	no
Backed up AIJs?	yes	yes	yes	yes
Process source db changes from live journal after the commit?	no	yes	no	no
On-going or finite?	finite	on-going	finite for a given run	finite for a given run, repeatable with same Unload File
Metadata file required?	yes	no	no	yes, unless the source db is available
LogMiner Options File required?	yes	yes	yes	yes
Commit rows required?	no	yes	yes	yes
Loader Control File required?	yes	yes	yes	yes
Unload file used?	yes	no	no	yes
Copy procedure used?	no	no	no	yes
Mailbox used?	no	yes	yes	yes
Checkpointing and restart?	no	automatic	okay within same data set	okay within same data set
Restart with additional data changes?	not done	not applicable	discontinuous issue	discontinuous issue
Performance?	little control	full tuning options	full tuning options; fewer disk writes than copy; more immediate than copy.	full tuning options, network traffic can be reduced, & can simulate scalability
Uses?	db reorg, if in use	diverse and numerous	db reorg	both for testing & for when source and target need to be asynchronous

Running the LogMiner Loader

How you run the LogMiner and the Loader depends on the mode. In some cases, you will, specifically run the LogMiner. In others, the Loader Control Process runs the LogMiner such that the LogMiner activity is transparent to you.

Different preparation is required, depending upon mode. Some of the preparation is discussed in the following sections. See also “Post-Installation Preparation” on page 99.

- Original Static Mode: There is no reason to begin a new project with this mode.
- Continuous Mode: See “Continuous LogMiner and the Loader” on page 69.
- Copy Mode: Run the LogMiner to create the unload file. Use the command for running the Continuous LogMiner Loader to use the unload file as input to the Loader. The logical name JCC_LOGMINER_MODE must be defined for the mode. The LogMiner will require the options file. The metadata file will be required if the source database is not available when the LogMiner is run and is optional if it is.The Loader will require the Control File. See also “Continuous LogMiner and the Loader” on page 69 and “Example for Copy Mode” on page 97.
- Static Mode: use the command for running the Continuous LogMiner Loader. The logical name must be defined for the mode. The LogMiner will require the options file and the Loader will require the Control File. See also “Continuous LogMiner and the Loader” on page 69.

When running the LogMiner for copy mode, use the qualifier ‘/order’ to order the AIJ backup files.

Setting the Mode

To specify the mode, define the logical name JCC_LOGMINER_MODE. Options are:

- Continuous: This is the default and is not required when running the Continuous LogMiner and Loader.
- Copy: This mode will use a pre-processed (static) LogMiner unload file. (Note that the logical name JCC_LOGMINER_OUTPUT_FILE must be defined to identify the unload file which is output by the LogMiner and used as input for the Loader.)

- Static: This mode will run the Rdb (static) LogMiner on a list of AIJ backup files specified by the JCC_AIJ_BACKUP_SPEC[1] logical name. (Note that the JCC_LOGMINER_METADATA_FILE logical name must also be defined.) This relies on the improved static mode which includes commit records and uses the mailbox.

If an invalid value is specified for the logical name, then the JCC_RUN_CLM procedure will exit with the following error:

```
%SYSTEM-F-BADPARAM, bad parameter value
```

Metadata for the LogMiner

With static LogMiner, it is possible to perform the LogMiner operation on a computer different from the one where the database is open. To support this likely division of the Loading from the source database, a LogMiner metadata file is required. This is a special (undocumented format) file used by Rdb to describe the metadata in a database. The command to extract a metadata file is:

```
$ rmu  /unload                                          -
       /after_journal                                   -
       /save_metadata=<your db.metadata>                -
       /log                                             -
        <target db name>
```

The metadata must be supplied to the LogMiner during the unload session, unless the source database is available to supply the same information about the metadata. Define the logical name JCC_LOGMINER_METADATA_FILE to identify the file created by the command just described so that the file will be used to specify the metadata that is valid for the specified AIJ backup files.

This is required when the LogMiner output is to a file, as for Copy Mode.

If an invalid value is specified, then the JCC_RUN_CLM procedure will exit with the following error:

```
%SYSTEM-F-BADPARAM, bad parameter value
```

1. Note that definition of this logical name must be sufficient to specify all the necessary backup files. Use wildcards as necessary.

The LogMiner Unload File

To run the Static LogMiner for Copy Mode, you will unload the after image journal files as shown here. There may be several of these files. The example shows unloading by wildcards.

The example includes specifications for statistics intervals for the LogMiner, extend sizes for OpenVMS, and other useful settings. The required portions are highlighted in red, but the other portions or something similar are important, as well.

```
$ rmu/unload/after_journal                      -
     /log                                       -
     /sort_workfiles=4                          -
     /io_buffers=15                             -
     /statistics_interval=60                    -
     /extend_size=65535                         -
     /notrace                                   -
     /restore=<your database>.metadata          -
     <target db name>                           -
     dpa300:[jeff]prod.aij;*                    -
     /options=(file=<your db.opt>)              -
     /include=action=(modify,delete,commit)     -
     /order_AIJ_files
```

FIGURE 3. Example Unload for Static LogMiner

Now, define JCC_LOGMINER_OUTPUT_FILE[1] to specify the unload file, the output file of the Oracle Rdb LogMiner that you just created.

```
$ define jcc_logminer_output_file <file name of unload file>
```

If an invalid value is specified, the JCC_RUN_CLM procedure will exit with:

```
%SYSTEM-F-BADPARAM, bad parameter value
```

See the LogMiner documentation for more discussion of the specifics of LogMiner.

1. In early versions, the logical name, jcc_logminer_loader_input has been used in this same way. It continues to work, but is deprecated with later releases.

LogMiner output files prepared for this use should include commit records.

Prepare for EOF

Since Static and Copy modes use only the backup AIJ files, there is an end to the input stream that they provide to the CLML Control Process. If the commit interval is greater than one, the Loader may receive a single transaction (or any number of transactions fewer than the commit interval) and wait to fill the commit interval.

The keyword Input_failure[1] in the Control File sets the amount of time the Loader should wait for the commit interval to fill before writing the information that is buffered.[2] To avoid having the Loader wait indefinitely for additional input before finishing, when in Static or Copy mode, the Loader behaves as if the keyword is defined with an input timeout of five seconds. This is the equivalent of

```
Input_failure~5
```

Five seconds is chosen to cause a fairly substantial pause in case the LogMiner is working on a large transaction. Should you find a need to make the wait shorter or longer use the keyword definition. See also “Checkpointing and Discontinuities” on page 92.

When the Loader sets the input_failure keyword, it will write to the log

```
COPY/STATIC mode input timeout set to default (5 seconds.)
```

Tuning

If tuning Copy or Static mode for maximum performance, see “Multiple CLM Processes” on page 71 in the Continuous chapter and “Parallelism and Loader Threads” on page 383 in the Performance chapter.

Other

You will also need to

1. See “Keyword: Input_failure” on page 248.
2. Continuous Mode can also face a similar issue in systems that have bursts of activity followed by no activity.

- Define the target. To learn about targets, see the information on the specific target and the section “Keyword: Output” on page 275.
- Create the Control File. See “Control File” on page 217.
- Prepare to monitor the session. See “Monitoring an Ongoing Loader Operation” on page 313.

Restart

Loader operations may be interrupted whichever mode is used. In any mode,

- There may be a difficulty caused by the target, network, or other source that causes an unplanned interruption.
- There may be a decision to interrupt the operation manually.

Restart from these interruptions is addressed in the same way for all of the modes. See “Rdb Issues” on page 412.

In addition, in Static and Copy mode, there will be planned interruptions when all the work that was prepared has been loaded. In some cases, only one run of the LogMiner and Loader is required. In other cases, the architecture will require repeated runs. It is the repeated runs that will require care. See “Checkpointing and Discontinuities” on page 92.

Manual Shutdown

If operations must be manually interrupted, use the command

```
$ JCC_CLML_SHUTDOWN <loadername>
```

All comments made relative to Continuous apply. See “Shutting Down Continuous LogMiner” on page 75.

Checkpointing and Discontinuities

In the normal Loader restart mode, the checkpoint data saved by the Loader requires that it look for a particular transaction for restart. In situations that repeatedly acquire data from the Source in Copy mode, the Loader will attempt the usual

restart precautions. Without special intervention, the attempt will fail because the last transaction processed before the interrupt will not be represented in the new unload file.[1]

In the example to follow, the restart information in the log file shows that it is looking for TSN 1001202358 in AIJ sequence number 50:

```
************************************************************
Restarting from previous execution
Restart information:
Checkpoint Timestamp:      6-APR-2007 15:34:39.61


Loader Sequence Number:  585679
TSN:                     1001202358
Start TAD:                4-FEB-2007 15:37:27.34
Commit TAD:               4-FEB-2007 15:37:31.04

AERCP:                   1-28-50-3-1001202358-1001202358

************************************************************
```

FIGURE 4. Example Checkpointing Discontinuity with Copy Mode

The log, for this example, also shows that the first transaction in the input data is in AIJ sequence number 52 (TSN 1025588256):

```
MQP in AIJ sequence number 52 (MQP 52-3-1025588256)
```

Given this scenario, the Loader reads to the end of the input file without finding the restart point. It will exit with the exception message

```
%DBA-I-INPUT_EOS, End of the input stream has been reached.
```

Obviously, this is a problem that comes with loading *as if the mode were Continuous*, when, in reality, the input is discontinuous. The problem can be addressed in several ways.

1. These comments can apply to Copy or Static Mode. The discontinuity issue is more frequently encountered when using Copy Mode.

Workaround. One workaround is to create a job that uses RMU/UNLOAD to unload the logminer_highwater table and then truncates it. This job should be executed once, each time that the Loader has completely processed the input data.

The unload file represents the safe way to undo the truncation and needs to be retained until you are satisfied.

The difficulty with this approach is that nothing guarantees the inclusion and order of all the backed up journal files.

Possible. The second method is to specify that the Continuous LogMiner Loader (only on the first run of each new batch of Copy data) ignore the checkpoint data. This can be done specifying the override parameter on the JCC_RUN_CLM_LML command line as “CHECKPOINT”.

The difficulty with this approach is that nothing guarantees the inclusion and order of all the backed up journal files.

Recommended. The recommended approach takes advantage of a full understanding of the problem, but does involve no change updates of the *source* database. The steps for this approach assume a periodic (perhaps, daily) and repeated use of the Copy (or Static) mode.

- Backup the AIJ (quiet point to get all the work) This is referred to in following steps as the BIG backup.
- Make a no-change update to a single row in a table that is processed by the LogMiner and the Loader.
- Backup the AIJ (quiet point) This is referred to in the following steps as small backup, as there will only be the one (no-change) change reflected in the AIJ.
- Run the LogMiner on the files
    - Small AIJ backup from previous run
    - BIG AIJ backup from this run
    - Small AIJ backup from this run

These steps provide input that supports the Loader’s finding the checkpoint. The three sets of AIJ files should be specified for a single LogMiner run so that the LogMiner will validate that the transactions follow one another appropriately.

The disadvantage of this approach, of course, is that it violates the goal of not modifying the source database. If that is not an issue for your environment, this is the preferred solution.

Which approach is correct is dependent on your environment. Consult JCC LML support for additional assistance.

Finer Control of the Start Time

When using Copy mode of the Loader, the AIJ files must be processed in Static mode of the LogMiner. The Loader kit includes a procedure to do this, JCC_UNLOAD_AIJS.

In its default mode the procedure provides all of the data in the available AIJ backup files since "midnight yesterday". It is possible to specify other starting points.

JCC_LM_DEFAULT_FROM_TIMESTAMP is a logical name that enables change in the default start time for the static Rdb LogMiner. The logical name can have any of the following date types:

Date Type	**Use**
absolute	uses the value input as the starting point
delta	uses the current time adjusted by the delta time provided
combination	uses a combination of one of the values listed below and a delta time

Additionally, any of the following values can be used:

Value	**Effect**
TODAY	begins at midnight today
TOMORROW	an unlikely value, as there will be no data in the AIJs for tomorrow
YESTERDAY	begins at midnight yesterday
NONE	eliminates the start time stamp such that Rdb LogMiner returns all of the data changes from each AIJ file processed

Examples and the resulting behavior:

```
$ define JCC_LM_DEFAULT_FROM_TIMESTAMP "1-Apr-2011 01:23:45.67"
$ define JCC_LM_DEFAULT_FROM_TIMESTAMP "-4:00"
$ define JCC_LM_DEFAULT_FROM_TIMESTAMP "TODAY +3:00"
$ define JCC_LM_DEFAULT_FROM_TIMESTAMP "YESTERDAY"
$ define JCC_LM_DEFAULT_FROM_TIMESTAMP "NONE"
```

The first example sets a specific time to begin. The next subtracts four (4) hours from the current time. The next begins three (3) hours after midnight today, eg at 3:00 AM today. The next sets the begin time to yesterday at midnight, which is the default. The final example eliminates the timestamp completely from the Rdb LogMiner command such that LogMiner processes all of the data changes for each of the AIJ files processed.

Note that any specification which leads to a time in the future will cause Rdb LogMiner to select no data changes. That is, “TOMORROW” and any combination which sets a time later than the current time basically turns off the procedure. For example, “TODAY +9:00” or “TOMORROW -15:00” will be in the future until 9:00 AM.

If the logical name is not defined, the default (YESTERDAY, meaning yesterday at midnight) is used.

Example for Copy Mode

This example from JCC regression testing has two parts, one for running the LogMiner and one for running the Loader. Copy mode specifics are highlighted in red.

```
$ set verify
$ set noon
$ rmu/backup/after/log     loader_regression_test_db                        -
                           loader_regression_copy_db_dir:backup_aij.aij
$ define options_file      jcc_root:[test]loader_regression_test_lm_unl.opt
$ define rdb_logminer_output_file                                           -
                           loader_regression_copy_db_dir:unload.dat
$ rmu/unload/after_journal loader_regression_test_db                        -
                           loader_regression_copy_db_dir:backup_aij.aij     -
                           /include=action=(modify,delete,commit)           -
                           /options = file=options_file                     -
                           /statistics_interval = 10                        -
                           /log
```

FIGURE 5. Example: Running the LogMiner for Copy Mode

```
$ set verify
$ set process/name="Copy Loader"
$ set proc/priv=all
$ show log dcl$path
$ set output_rate=0:0:05.0
$ set on
$!
$ define jcc_clml_logging_style                "Reuse"
$ define jcc_logminer_loader_lock_threshold    1000
$ define jcc_logminer_loader_stat_interval     300
$ define jcc_logminer_loader_stat_type         DELTA
$ define jcc_logminer_mode                     "COPY"
$!
$ define target_db               loader_regression_copy_db
$ define source_db               loader_regression_copy_db_dir:unload.dat
$ define jcc_logminer_output_file source_db
$!
$! In this section define the files to use with the Loader.
$!
$ define loader_regression_test_options                                     -
                     jcc_tool_examples:loader_regression_test_lm_unl.opt
$!
$ define JCC_LogMiner_Loader_Name              REGTESTCPY
$ define JCC_LogMiner_Loader_HW_Response       CREATE
$!
$ jcc_run_clm_lml                source_db                                  -
                                 loader_regression_test_options             -
                                 loader_regression_test_control
```

FIGURE 6. Example: Running the LogMiner and Loader for Copy Mode

Example for Static Mode

There is an example of running the Static LogMiner in the section “The LogMiner Unload File” on page 90. That example illustrates some of the settings used for tuning OpenVMS in one of the Loader applications. That particular application runs the Static LogMiner on the system on which the source database is open and uses the source database directly to define the metadata.

The example which follows relies on a metadata file to define the source database metadata and is useful for comparing to the Copy mode example in the previous section.

```
$ set verify
$ set process/name="Static CLML"
$ set proc/priv=all
$ show log dcl$path
$ set output_rate=0:0:05.0
$ set on
$!
$ define jcc_clml_logging_style "Reuse"
$ define jcc_logminer_loader_lock_threshold 1000
$ define jcc_logminer_loader_stat_interval 300
$ define jcc_logminer_loader_stat_type DELTA
$ define jcc_logminer_mode "STATIC"
$!
$ define target_db loader_regression_copy_db
$ define source_db -
       regression_test:loader_regression_test_lm.saved_metadata
$ define jcc_logminer_metadata_file source_db
$ define jcc_aij_backup_spec -
loader_regression_copy_db_dir:backup_aij.aij
$!
$! In this section define the files to use with the Loader.
$!
$ define loader_regression_test_options -
jcc_tool_examples:loader_regression_test_lm_unl.opt
$!
$ define JCC_LogMiner_Loader_Name REGTESTSTA
$ define JCC_LogMiner_Loader_HW_Response CREATE
$!
$ jcc_run_clm_lml source_db -
       loader_regression_test_options -
       loader regression test control
```

FIGURE 7. Example: Running the Loader for Static Mode

CHAPTER 7

Post-Installation Preparation

Before the Loader can be used for the first time, a few more preparatory steps are required.

This chapter focuses on steps to prepare the source, provide for LogMiner output, and establish some of the logical names needed. Other input and output are covered elsewhere.

The diagram to follow is simplified to emphasize the inputs and outputs of the Loader.

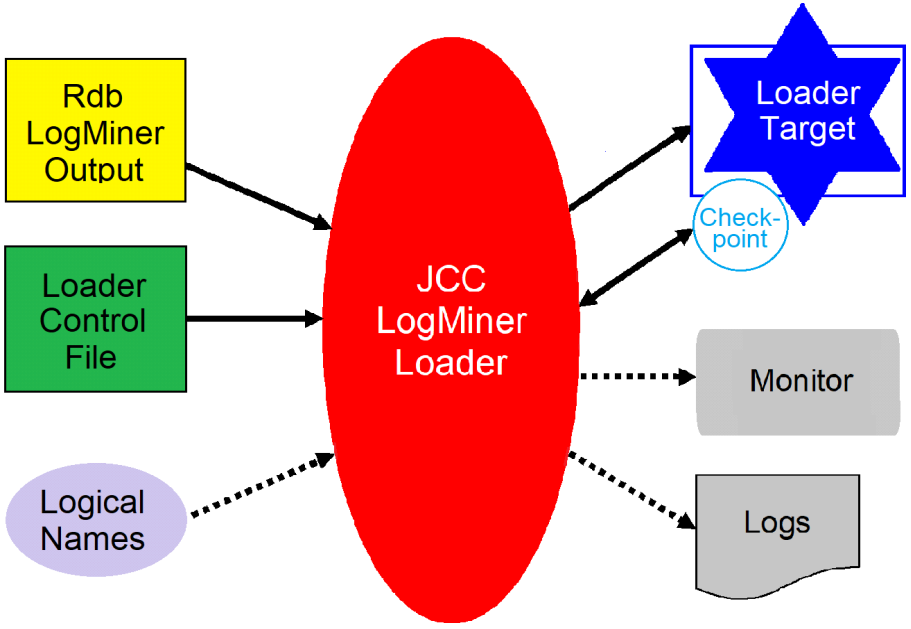


FIGURE 1. JCC LogMiner Loader Input and Output

Inputs

The Loader requires two inputs and may also use logical names.

- Output from the LogMiner
  - For Continuous *LogMiner* use, the LogMiner feeds a stream of records to an OpenVMS mailbox and the Loader reads it from there.[1]
  - For Static *LogMiner* use, the LogMiner outputs to a file, "the unload file." For Copy mode the Loader copies this file to use to feed the OpenVMS mailbox. (See "Modes of Operation" on page 77.)
- The Loader Control File (See "Control File" on page 217.)
- Logical names (See "Loader Process Logical Names" on page 106.)

1. The new Static mode of the Loader can also use the LogMiner to output directly to the OpenVMS mailbox. See "Improved Static Mode" on page 79.

Outputs

The exact nature of each output varies according to choices made in the Control File. Output may include

- Updates to the Loader target
- Checkpoint updates (may be written to the target, if the Loader target is an Rdb database or an Oracle database accessed directly.) The checkpoint will also be read as input should the Loader operation be interrupted and restarted.
- Log files and screen output for Administrator monitoring

Preparing the Source Database

Understanding how to prepare this starting point requires some understanding of how the LogMiner works.

Quiet Points

A quiet point is an arbitrary point in the functioning of the database for which there are no transactions that have been started that have not been either committed or rolled back. Quiet points can occur because the Database Administrator requested a quiet point backup. Journal quiet points also occur naturally. The naturally occurring quiet points are called Micro Quiet Points (MQP).

LogMiner relies on starting at a known point so that transactional consistency can be maintained. Both a database backup and an AIJ quiet point backup are important steps in preparing the source. To request a quiet point backup, issue the following command:

```
$rmu/backup/after/quiet <database name>
```

The AIJ file written in response to the request for a quiet point backup will be labeled as quiet point. It is the *next* AIJ backup file that begins at a quiet point and provides a suitable place to begin the LogMiner.

See also “Restart and Backup” on page 424 to understand the role of quiet points in *re*starting Loader operations and “Tracking AIJ Switches” on page 432 for additional information on interpreting the AIJ sequence number.

Enabling the LogMiner

The LogMiner input to the Loader is created when the source database is enabled for LogMiner or Continuous LogMiner. This is accomplished within Rdb with an RMU command. The optional parameter 'continuous' is used to specify continuous operation.

```
$ rmu/set logminer/enable[/continuous]<database name>
$ rmu/backup/after/quiet <database name> ...
```

Continuous use of the LogMiner and Loader provides a continuous, near realtime feed of source database changes to the target. Continuous is the usual choice. For preparation and use specific to continuous operation see the chapter "Continuous LogMiner and the Loader" on page 69.

Rdb LogMiner Options File

Running LogMiner requires a specification called the Rdb LogMiner options file. The Loader kit includes a script to create a default options file. You may execute the command[1]

```
$ JCC_CREATE_LOG_MINER_OPT_FILE <source database>
```

to create an options file that includes all tables in the database. The file will be created in your current directory and will be named "<database>_LM_UNL.OPT". JCC, generally, recommends that your directory for this work be JCC_TOOL_LOCAL, but that is not a requirement.[2]

For example:

```
$ jcc_create_log_miner_opt_file LOADER_REGRESSION_TEST_DB
Directory USER_ROOT:[REGRESSION_TEST]
LOADER_REGRESSION_TEST_JDBC_LM_UNL.OPT;1
2/515 15-JUN-2007 13:53:15.26
Total of 1 file, 2/515 blocks.
1 !
```

1. This procedure was named 'create_log_miner_opt_file.' Since that naming conflicted with the DCL command 'create', the full path specification was required. To simplify using the procedure, the name was changed, with the introduction of LogMiner Loader Version 2.2.8, to the command shown. The procedure is deprecated.
2. See "Tailoring Procedures" on page 59 for reservations about this approach and additional discussion.

```
No substitutions
USER_ROOT:[REGRESSION_TEST]LOADER_REGRESSION_TEST_JDBC_LM_UNL.OPT;2 19 lines
Directory USER_ROOT:[REGRESSION_TEST]
LOADER_REGRESSION_TEST_JDBC_LM_UNL.OPT;2
2/5 15-JUN-2007 13:53:16.61
LOADER_REGRESSION_TEST_JDBC_LM_UNL.OPT;1
2/515 15-JUN-2007 13:53:15.26
Total of 2 files, 4/520 blocks.
%DELETE-I-FILDEL, USER_ROOT:[REGRESSION_TEST]LOADER_REGRESSION_TEST_JDB-
C_LM_UNL.OPT;1 deleted (515 blocks)
$
```

You may edit the generated options file as necessary. To exclude tables from the Options File see “Excluding Tables from the Options File” on page 103.

To learn more about the options file, consult the Rdb documentation.

Excluding Tables from the Options File

Some architectures do not require use of all of the tables from the source database. The most efficient method to exclude tables from Loader operations is to remove them from the Rdb LogMiner Options File.

The Loader includes the option of creating symbols to exclude unwanted tables so that the Options File does not need to be hand edited. The DCL symbols must be of the form:

```
JCCLML_<table name> = “EXCLUDE”
```

A symbol so declared will cause the JCC_CREATE_LOG_MINER_OPT_FILE procedure to mark the requested table as excluded in a comment within the generated Options File.

Note that the JCC_TOOL_SQL:VMS_FUNCTIONS.SQL is updated in version 3.3 of the Loader. If you have been running an older version of the Loader, the new version of JCC_TOOL_SQL:VMS_FUNCTIONS.SQL must be applied to the source database in order to use the new functionality.

LogMiner and the AIJs

JCC recommends that AIJs written before the LogMiner was enabled be not only backed up at a quiet point, but also removed from the backup directory that will be used with the Loader. See also “Preparing the Source Database” on page 101.

Define Functions for Later Use

Many of the procedures included with the Loader kit rely on the procedure vms_-functions.sql having been applied to the source database. Apply this procedure to the database prior to using any other procedures from the kit.

```
SQL>attach 'filename <source database>';
SQL>@jcc_tool_sql:vms_functions.sql
SQL>commit;
```

Preparing Sources Created with Earlier Versions of Rdb

The Rdb LogMiner requires access to the metadata in the source database. In addition to the metadata, the LogMiner also accesses the Rdb Area Inventory Pages (AIP) in the database. It uses the information in each AIP entry to help resolve the logical area numbers to be assigned to tables in mixed areas.

Prior to Rdb version 7.0.1, AIP entries were not marked with the type of object they mapped. With version 7.0.1, Rdb provided the ability to do RMU/SHOW statistics displays by logical area. This version also provided the capability to retroactively mark these areas. If your source database being referenced during an Rdb LogMiner session was created with a version of Rdb prior to 7.0.1, you should be sure to perform the relevant RMU/REPAIR commands to properly mark AIP entries. There is documentation in the RMU/SHOW Statistics handbook that documents how to do this. Commands similar to the following are appropriate:

```
$ rmu   /repair                                          -
        /initialize=larea_parameters=sys$input           -
        /log                                             -
        /noabm                                           -
        <db name>
account                 /type=table
customer                /type=table
      o
      o
      o
```

User Procedures

Before beginning or using any of the examples, establish appropriate JCC LogMiner Loader context by executing the procedure JCC_LML_USER located in JCC_TOOL_COM:

```
$ JCC_LML_USER <version> [1]
```

<version>. The version is 'S' to invoke the system-wide default version or a version number to invoke the variant version. A version specification is only required if you are running a multi-variant installation. For example

```
$ JCC_LML_USER 3.5
```

can be used to specify the 3.5 version, if you are running it as a variant while also running a different version as the standard version.

Additional User Procedures

Some targets will have additional user setup that is appropriate. JCC_LML_USER applies to all.

Control File

The Control File is where you further define the source, the target, and the mapping between them. The Control File is also where you define performance and monitoring characteristics for your Loader family.

There is an entire chapter to guide you through the available options and how to specify your choices. See "Control File" on page 217. To see how to run the Loader procedure to create the metadata portion of the Control File, see "Building the Metadata Control File" on page 222. To see how to include the metadata control file, see "Referencing Other Control Files" on page 219 and "Keyword: Include_file" on page 246.

1. The first time this is run, it needs to be run as $ @JCC_TOOL_COM:JCC_LML_USER. This configures the JCC LogMiner Loader environment for an OpenVMS process. Among other things, this step is necessary to cause the commands shown in this document to work as shown.

JCC, generally, recommends that all of the control and options files used by the Loader and LogMiner be located in the JCC_TOOL_LOCAL directory, but that is not a requirement. See “Tailoring Procedures” on page 59 for reservations about this approach and additional discussion.

Additional Resources

The Loader kit includes additional resources. Two Loader features are of particular interest to your startup.

Loader Process Logical Names

The continuous LogMiner Loader uses a mechanism to define logical names in the context of individual Loader processes. To do this requires the presence of a special file jcc_runtime_parameters.dat. This file contains the necessary logical names. The procedures supplied with the Loader read appropriate records from this file and define logical names as required.

See “Logical Name Controls for Loader Procedures” on page 463 for more on defining logical names and the appendix for a complete list of logical names used with the Loader.

Database for FilterMap

A small database will also be created to support analysis of SQL statements should the keywords filtermap or mapresult be used. This database is not intended to hold data and requires no management. For more information see “Keyword: FilterMap” on page 242, “Keyword: MapResult” on page 266 and “Controls for the Filter Database” on page 461.

Running the Loader for the First Time

When the Loader is run for the first time, there is no restart context in the target.[1] Whether the target has been defined to be a database or a target that requires a local high-water file, you still have no restart context the first time the Loader is run.

When the Loader or the Control process finds that there is no restart information, an undefined restart situation exists. By default, the Loader (or the control process) emits an operator request to determine whether the lack of highwater data is expected or not. The operator must respond either "QUIT" or "CREATE".[2]

The "CREATE" response is available for starting for the first time. In this case, the Loader will write new "blank" context before it starts. This blank context will cause the Loader to start with the materialized column Loader_Sequence_Number set to 0 and to begin LogMiner and Loader operations at the start of the context provided.

However, if there has been a context and something inadvertent has happened to remove this context, the Loader's behavior might result in incorrect results in the target. For this reason, the Loader will request operator intervention before it writes this initial context.

An example of this interaction is shown here.

```
%%%%%%%%%%  OPCOM  20-FEB-2002 13:53:45.83  %%%%%%%%%%

Request 21, from user JEFF on ATLAS

JCCCTL: 'SCP_CLM_ORACLE' has no existing highwater data.  (CREATE, QUIT)

 $ REPLY/TO=21 CREATE

CREATE
13:54:00.68, request 21 was completed by operator _TNA85:
```

FIGURE 2. Example of Running the Loader for the First Time

1. For restarting the Loader, see "Rdb Issues" on page 412 and other sections.
2. By default, operator requests go to CENTRAL. For information on directing the Operator requests to other destinations, see "Keyword: Operator" on page 274.

For some, the operator interaction is a requirement that prevents operational errors. For others, this is more annoying than helpful. Therefore, a logical name (JCC_LOGMINER_LOADER_HW_RESPONSE) can be defined. The translation values for this logical name should be either CREATE or QUIT.

If the Loader or Control process finds a situation with no highwater data, the process will translate the JCC_LOGMINER_LOADER_HW_RESPONSE logical name and use the translated value as the response for the operator message. If the translated value matches one of the expected responses, the program will continue as if the operator interaction has occurred. If the translated value does not match one of the expected responses or is not defined, the Loader or Control process will generate the OPCOM message as normal.

Preparing for Statistics on the Session

See “Monitoring an Ongoing Loader Operation” on page 313 and “The Log Files” on page 356 and other sections in the Monitoring chapter for additional preparation that will collect the data that you may want for analyzing your Loader session.

Preparing the Target

The JCC LogMiner Loader, working with the Oracle Rdb LogMiner, publishes changes made to your source database into a target. You must provide a target that is populated with the data that you want as it exists at the time at which you initialize the LogMiner.

For notes on preparation of the target, see the relevant chapter or chapters:

- “Rdb Targets” on page 111
- “Oracle Targets” on page 123
- “JDBC Loader Targets” on page 143
- “XML for File or API Targets” on page 199
- “Tuxedo Targets” on page 179
- or the Kafka Option documentation

For a way to populate the target, initially, see “Data Pump” on page 505.

Examples

There are examples provided with the kit. To make use of them define the logical name JCC_TOOL_EXAMPLES to point to the examples that are pertinent for your choice of target.

```
$ Define JCC_TOOL_EXAMPLES JCC_TOOL_ROOT:[EXAMPLES.<your choice>]
```

where "<your choice>" is "API", "JDBC", "ORA", "RDB", or "TUX".

CHAPTER 8

Rdb Targets

The JCC LogMiner Loader publishes changes made in your source Rdb database to a target or targets. There are many possible targets. This chapter discusses the Loader's interactions with Rdb targets.

An Rdb database can be a direct target of the Loader[1] or it can be the end target of declaring JDBC as the Loader's target. This chapter is designed to discuss Rdb as the direct target of the Loader.

Some of the things discussed in this chapter are similar for other targets. However, this chapter focuses on Rdb targets and each other target type is presented in a separate chapter. This arrangement is intended to enhance the benefit for your specific installation.

1. Rdb as the direct target of the Loader is the default. It can also be defined as the direct target of the Loader through defining the keyword Output with the output type "Rdb".

Defaults

Replicating to an Rdb target is the default Loader behavior. This chapter lays out the simple steps for establishing your Rdb target. You will, of course, need to consider the chapters on the Control File, tuning, and others, especially the chapters that lay out the basics.

If you are only using an Rdb target, you will not need to reference the chapters on other targets.

Software Versions

For details on tested configurations, see “Software Versions and Related Products” on page 49 or the blog at http://www.jcc.com/LML_prod_compat

Preparing the Target

Preparation of the Rdb target follows these steps.

1. Define the target database. Alternatives for creating the database are:
   - Define the target database with standard SQL syntax.
   - Define the target database by using RMU extract to generate a script that completely defines the existing database. Edit this script as appropriate to eliminate tables and columns not needed in the target. Edit this script to add tables and columns, as necessary, for your architecture.
   - Define the target database by exporting the source database with no data and then importing the result. Redefine necessary structures during the import.
   - Define and populate the target database by backing up the source (originating) database and using a restored version of the backup as the target. Without additional changes, this is a suitable base for replication. For other uses, modify the target database, as appropriate.
   - Create the target database by unloading data from the source database. Such unload functions should be done to binary files to preserve *nulls*.

2. Remove from the target all triggers and constraints that exist in the source. See "Constraints and Triggers in the Target Database" on page 118.
3. Add the Loader highwater table to the target database. See "Adding the High-Water Table" on page 114.
4. Add the dbkey columns, if any are required. See "Adding Dbkey Columns" on page 115
5. Perform whatever physical redesign is necessary and edit the generated SQL script.[1]
6. If you did not define the database with a method that also populates the tables and columns with data, you must now load an appropriate base set of data into the target database. Alternatives for populating the target database are discussed in the following section.
7. Populate the dbkey columns, if any. (You must ensure that the dbkey columns are populated with the actual dbkey column values in the *source* database, not the dbkeys of the target.) See "Adding Dbkey Columns" on page 115.
8. Configure the target database for performance. Be sure to add indexes on dbkey columns you have added. Do not create more than one unique index per table.

Populating the Target

The JCC LogMiner Loader in conjunction with the Oracle Rdb LogMiner publishes, to your target, *changes* made to the source data. It is necessary to start with the target populated with the data that you will want to update.

If the target is updated by programs other than the Loader, care will be required to avoid overwriting target data in a fashion that is incompatible with your architecture for replication. See "Restoring the Initial Load" on page 114.

1. For some uses of the Loader, database reorganization embodies the whole goal. For others, there will be no intention to revise the physical database.
   For database reorganization as a goal, see the chapter "Example: Reorganizing an Rdb Database" on page 529

Initial Load of the Target

The initial population of the data may be handled in a variety of ways. Here are some that apply.

1. Use RMU/UNLOAD. Be certain to use binary files for the unload/load. This method requires the creation of disk files and is, therefore, the slowest method. The creation of the disk files takes a large fraction of the total time for this process.
2. Backup the source and restore it to create an identical target.
3. Use the Data Pump of the JCC LogMiner Loader. This method is the fastest and has the least overhead. It does, however, require that you pass all rows (that you want in your target) through the AIJs. You will have to provide sufficient AIJ space to support this. The Data Pump is discussed further in “Data Pump” on page 505.

Continued Change While You Work

If your source database remains active, while you are preparing the target, additional changes are accumulating in the AIJ files. You can run the LogMiner and the Loader to transfer these changes to the target and catch up. See also “Quiet Points and AIJs” on page 43.

Restoring the Initial Load

The Data Pump was originally developed to address issues when target databases that had required weeks to populate became, during development of downstream applications, invalid representations of the source data. The Data Pump permits pumping subsets of the data from the source to the target to restore the synchronization of the two. The Data Pump is discussed further in “Data Pump” on page 505

Adding the High-Water Table

The high-water information is used by the Loader to keep track of what has been processed. For Rdb targets, the default use is to maintain the information in a high-water table that can be updated within the same transaction that writes the source transaction updates to the target database.

The highwater information is required for recovery from failures and shutdowns.

Create the high-water table in an Rdb database by executing the command:

```
$ jcc_create_logminer_highwater <target database>
```

This script will add a small storage area to the database and create the necessary high-water table. You may edit this script as necessary to place the new storage area on a preferred disk and directory. You should not change the table names, column names or data types.

This script should be run on the same node as the target database.

Note: The create_logminer_highwater procedure is constructed in such a way that it expects the target database to be closed and marked as "open is automatic" when it is initiated. If your database does not meet these criteria, type out the procedure and perform the operations manually.

Populating the HighWater Table

The highwater table is not created with any data rows. The first time that you run the Loader, this will need to be corrected. See "Running the Loader for the First Time" on page 102.

Recovering from a Failed or Shut Down Session

The Loader maintains its own context with either a high-water table or a checkpoint file. For Rdb targets, the database table is generally the better approach.

By default, the Loader will restart where failure or shutdown occurred.

The chapter "Aids for the Administrator" on page 409 includes a section of Rdb issues that are current as of this writing. See "Rdb Issues" on page 412 for a rare, but serious Rdb issue with restart and backup.

Adding Dbkey Columns

For the Loader to support data updates and deletes, it is necessary that the Loader be able to identify, in the target, the row that has been updated or deleted in the

source. This requires a column or collection of columns that uniquely defines the row — without nulls or changing values in any of those fields.[1]

If there is no collection of columns[2] that uniquely defines the row — without nulls or changing values in any of those columns — you may need to establish a column for the originating dbkey.

There are issues in using dbkeys and JCC recommends against using them, unless there is no other option.

See "Identifying Rows in the Target" on page 39 for a more complete discussion of the importance of keys for identifying rows in the target. That section also discusses the occasional necessity as well as the drawbacks of using the originating_dbkey approach and the alternative of changing the *source* to include identity attributes.

Data Types

When the target database is Rdb, the originating_dbkey column is, by default, an 8-byte string. When using Microsoft Access with your Rdb target, you will want the DBkey columns to be BIGINT.

Aids to Creating Originating_dbkeys Columns

The special dbkey columns can be automatically added to tables in the target database by creating a file containing a list of such special tables, one table name per line. You should edit the generated files as appropriate to modify only those tables requiring originating_dbkey columns and to create and place indexes in proper storage areas, etc.

Execute the following command to generate the necessary scripts for adding the dbkey columns:

```
$ jcc_add_odbkey_set_index <file of table names>
```

1. Note that it is inconsistent (with the rule that all columns in the key must be unchanging and not null) for the key to require all columns in the table and expect to be able to do updates.
2. Note that it is also possible to use the Rdb option for creating an additional column *in the source* that will be unique and not null. See "Identity Attributes" on page 118.

The command will generate the following files:

TABLE 1. Files generated by the dbkey procedure

File	Purpose
<file_root>_source.sql	An SQL script to create views in the source that include the dbkey
<file_root>_odbkey.sql	An SQL script to add the originating_dbkey column to each table in the target database. This generated script will also populate the new column with the current dbkeys of existing rows. (This assumes that the database being so managed reflects the source database exactly.)
<file_root>_set_odbkey.sql	An SQL script to populate the new columns in the target database with the current dbkey of the column in the source database. (The procedure reflects an assumption that the target is an exact replica.)
<file_root>_index.sql	An SQL script to create necessary indexes on the new dbkey columns. These indexes are sorted indexes.
<file_root>_drop.sql	An SQL script to drop the originating_dbkey columns.
<file_root>_ unload_-load.com	A command procedure to unload views from the source database and load into the target.

You should edit the generated files as appropriate to modify only those tables requiring originating_dbkey columns and to create and place indexes in proper storage areas, etc.

Creating DBkey Columns Using Rdb Materialized Values

As an alternative to the Loader process, one can unload a view in which the dbkey is materialized as a column and load that result into another table. The advantage of this alternative is that there is likely to be much less fragmentation in the resulting table. To use the materialized column approach, you will need a patch that is available in Rdb 7.0.6.1 or a later version of Rdb that allows RMU to unload this materialized value.

Identity Attributes

It is possible to avoid DBkey columns with an identity attribute in the source.[1] Since creating an identity attribute, if one does not already exist, requires modifying the source database, this approach may not be acceptable in your environment.

Constraints and Triggers in the Target Database

The order of records in the LogMiner output cannot be predicted. If the source database enforces a referential integrity constraint and a transaction adds both parent and child rows in a single transaction, then it is possible that the Loader will first receive the child row and then the parent. If a referential integrity constraint is present in the target database, this sequence will cause the Loader to fail.

Similarly, if a column is maintained in the source by a trigger and is written to the target by the Loader, then the trigger is unnecessary in the target database. In fact, it is inappropriate. The rows that are maintained by the trigger will appear in the LogMiner output themselves and the Loader will similarly process them. The result is that the triggered actions will occur anyway. Since the order of records in the LogMiner output is unpredictable, it is possible that the firing of a trigger in the target database could result in inaccurate data.

For replicated databases, all database integrity and triggers will be maintained by the source database. In the target database, these actions are completely unnecessary and inappropriate.

In instances other than replication, there may be triggers that are appropriate to the target that were not in the source. In other instances than replication, there may also be constraints that are used. However, constraints should be limited to tables that are *not* maintained by the Loader and do *not* include data that is subject to foreign key constraints or check constraints involving data that is maintained by the Loader.

1. Identity attributes require Rdb version 7.1.0.2 or later.

Targets that Are Different from the Source

The target database will contain some or all tables that are present in the source database. The table names in the target may differ from those in the source although, by default, they are the same.

Similarly, the names of the columns in the target database may be anything you wish. The names used by the Loader are specified in the Control File. The relative order of columns in the target database table is not important.

Note that you may also use multiple tables to receive the data from one source table and may filter which target table receives which data. Alternately (or in addition), you may direct the data from multiple tables (in the same or different databases) to one target table, *if the keys are appropriate.* These mappings of source to target are specified in the Control File.

See “Schema and Data Transforms” on page 489 for additional discussion.

Note that the comments on changes between the source and target apply to other target types, but they are so important to understanding how to get started that they are repeated here.

Backup and Quiet Points

It is appropriate to start with the backup copy of your database. JCC recommends that you establish an arbitrary point (epoch, a point in time) at which you know that the database has no active update transactions. This is otherwise known as a *quiet point.*[1] You should, then, perform both a database and an AIJ quiet point backup at that epoch. You now have a stable starting point.

If your source database remains active, while you are preparing the target, additional changes are accumulating in the AIJ files. You can run the LogMiner and the Loader to transfer these changes to the target and catch up. See also “Quiet Points and AIJs” on page 43.

1. See also “Preparing the Source Database” on page 97.

Note that the comments on backup and quiet points apply to other targets, but they are such an important part of getting started that they are repeated here.

Remote Rdb Targets

If accessing the Rdb target with a remote TCPIP connection, the Validation keyword may be used to provide the username and password. See “Keyword: Validation” on page 290.

Isolation Levels and Rdb Targets

Rdb targets default to read committed for the locking model. This provides a locking model that is less conservative than Rdb’s usual serializable. By using read committed., the Loader reduces the duration that read locks are held. This choice is appropriate for most circumstances.

The success of using read committed is due to the Loader’s applying transactions to the target in the same order as they were committed in the source and assumes a constrained[1] model such that updates are applied to the target in the correct order.

Possible Difficulty

Read committed opens the way, in certain environments, for a rare exception. In some very high update tables with particularly high duplicate cardinality indices, Loader threads may process in an order that causes a Loader thread to receive an SQLCODE_EOS exception. This exception normally indicates that a row the thread is seeking does not exist. Due to the order in which threads are processed, however, it can be reported incorrectly.

1. See “Keyword: Parallel” on page 270,

Retry

The Rdb target code beginning with Version 3.5 handles the EOS exception in the same way that it does a deadlock.[1] It will retry the set number of times. This is likely to address the difficulty.

Changing the Isolation Level

It is possible for the Loader Administrator to change the isolation level by setting the logical name JCC_LML_RDB_ISOLATION_LEVEL. The logical name defaults to "READ COMMITTED". Changing it to serializable (the Rdb default) can eliminate the rare SQLCODE_EOS, but is likely to introduce significantly more locking.

```
$ define JCC_LML_RDB_ISOLATION_LEVEL serializable
```

Possible values for the isolation level are anything that Rdb will accept.

Using the Source as the Target

It is possible to use the Loader to publish back to the source. There are, of course, some difficulties with this. To publish to the same tables and columns would set up an endless loop. To create additional tables and/or columns requires a careful analysis of goals and methods. Tuning can be challenging.

This is not a recommended approach, unless it meets a genuine need.

Next Steps

Your target database is now prepared and you can focus on the Control File to tell the Loader what you want to do. The chapters on other targets may be of no interest to you.

1. See "Keyword: Output_failure" on page 269 for how to set the number of retries used.

CHAPTER 9

Oracle Targets

The JCC LogMiner Loader publishes changes made in your source Rdb database to a target or targets. This chapter discusses the Loader's interactions with Oracle targets that are accessed directly.[1]

Preparation of the target differs little between Rdb and Oracle and some of the things discussed in this chapter are similar for other targets, as well. However, this chapter focuses on Oracle targets.

Focusing on the Oracle targets is intended to enhance the benefit for your specific installation and to cite the many special features that have been added to the Loader to improve performance or ease of use with Oracle targets.

1. Oracle is defined as the direct target of the Loader through defining the keyword Output with the output type "OCI". Oracle can also be the end target of declaring JDBC as the Loader's target See "JDBC Loader Targets" on page 143.

Software Versions

The Loader can write directly to an Oracle target. For specific versions supported, see also “Software Versions and Related Products” on page 49 or the blog at http://www.jcc.com/lml_prod_compat

Constraints on Version Combinations for Oracle Clients and Servers

Several Loader users have encountered difficulty when there is a discrepancy between the Oracle client (running on the OpenVMS source) and the Oracle server (running on the target). The performance enhancements added to the Loader with version 3.1.2 expose this bug. According to Oracle resources, the problem can occur if the server included the fix for bug 3396162[1] and the client is a version which does not include the fix. Server versions that include the fix are 9.2.0.8, 10.1.0.5 and 10.2.0.1 onwards. Clients which do not include the fix are versions less than 9.2.0.7 or 10.1.0.4. The bug results in an access violation. Oracle notes (document IDs) 5933477.8 and 455832.1 should be reviewed to determine exposure to this bug, as well as possible workarounds.

Issue with Oracle Compilation

Oracle introduced an issue with early versions of Oracle 9.2 and repeated the issue in versions 10.1 and 10.2 on alpha and in early versions of 10.2 for Integrity. The issue is fixed in Oracle 9.2.0.4 and, for Integrity, can be addressed by installing a patch on all client systems. The issue with the versions that do not work correctly is that they include local installation timestamps in the Oracle shared images. Because of this, Oracle images linked on one system (JCC’s, for example) would not run on another system (yours, for example). Consequently, the Loader is not linked with Oracle versions that have this difficulty. The exception message looks like

```
dba create_output_stream condition handler
%LIB-E-ACTIMAGE, error activating image DSA204:[ORA10204.LIB32]LIBCLNTSH.SO
-SYSTEM-F-SHRIDMISMAT, ident mismatch with shareable image
%TRACE-E-TRACEBACK, symbolic stack dump follows
  image    module    routine            line     rel PC           abs PC
 LIBRTL                                    0 00000000008DEBC FFFFFFFF80CBDEBC
 JCC_LOGMINER_LOADER_BASE_SHARE DBA_OS_SPECIFIC dba_callg_shareable_routine
```

FIGURE 1. Exception Message for Oracle Versions with Timestamp Issue

1. Contact Oracle for a patch specific to the version of Oracle that you wish to use.

Preparing the Target

There are some preliminary Oracle-specific steps that are described, first. The main steps for creating the Oracle target are very similar to those for an Rdb target.

Preliminary Steps for Oracle Targets

The JCC LogMiner Loader kit is shipped with shareable images that communicate with SQL*net interfaces for Oracle. These interfaces require that INSORACLE, the Oracle startup procedure, has already been executed on your OpenVMS server.

The kit contains a procedure, JCC_LML_ORACLE_USER, that determines which version of Oracle is to be used by the Loader at run time. This procedure requires two parameters:

```
JCC_LML_ORACLE_USER <Oracle interface version>    -
             <fullpath to Orauser procedure>
```

The JCC_LML_Oracle_user procedure must be run *after* any changes are made to lnm$file_dev. Doing so preserves all logical name definitions.[1]

Exceptions and the Oracle Installation

Exceptions in the ORAUSER procedure are indications that the Oracle software is not correctly installed and/or configured on the system. When an exception is encountered, the JCC_LML_ORACLE_USER procedure will generate a message similar to the following.

1. The Loader uses logical name tables, Oracle uses logical name tables, and many installations have their own. To preserve all of these, the procedure JCC_LML_Oracle_user looks for an existing lnm$file_dev in the lnm$process_directory (to see if it has already been modified) and, then, inserts the Oracle table immediately after the process table. If the lnm$file_dev does not exist in the lnm$process_directory, the Loader procedure copies one that is in the lnm$system_directory to the lnm$process_directory, then, inserts the Oracle table in the appropriate slot. The Loader procedure never removes any logical name tables, but the Loader can be confused by failure to run the JCC_LML_Oracle_user procedure after any changes to lnm$file_dev.


```
$ @JCC_LML_ORACLE_USER 9.2 disk$software:[oracle92]orauser.com
Setting JCC LogMiner Loader Oracle version 9.2
************************************************************
**** There was a problem encountered with the orauser procedure.
**** The Oracle software is not correctly installed/configured.
**** Please consult Oracle to resolve this problem.
****
**** Exception returned by orauser procedure:
**** %NONAME-E-NOMSG, Message number 00000002
****
**** Oracle version not set
************************************************************
%NONAME-E-NOMSG, Message number 00000002
```

FIGURE 2. Exception Indicating Improper Installation of Oracle

Preparing the Oracle Target

Preparation of the Oracle target database can begin with your use of common syntax for defining an Oracle database or can use procedures provided with the Loader kit. Using the Loader procedures may help you avoid issues.

Note that many of these steps assume that the vms_functions.sql procedure has been applied to the source database. A couple of the steps assume that the logical name RDMS$DEBUG_FLAGS_OUTPUT is defined to specify the output file. Also, the overall approach is intended for inclusion in your own DCL.

Using the procedures provided with the Loader kit follow these steps.[1]

1. Define the target database.
   - Create a JCC names table (in the source database). The purpose of this table is to resolve any issues of Rdb names that are too long for Oracle. Rdb names can employ 31 characters. Oracle names are limited to 30. When the target is Oracle, the Loader checks names for tables and columns to be used in the target and reports those that are too long for the Oracle target.
   - While attached to a restored backup of the source database, populate the names table with corresponding Oracle names using

     `jcc_generate_oracle_names.sql`

1. If you have familiarity with preparing Rdb targets for the Loader, you will find these steps familiar. Also, there is a detailed example in one of the appendices.

- Edit the names table as desired to modify the names to be used for the Oracle tables and columns.
- Populate the DCL symbol "TABLE_NAME" with the name of the table to be defined for Oracle. (Repeat this and the next step for each table to be propagated.)
- Generate Oracle style SQL to generate the table, using `jcc_generate_oracle_table_sql.sql`
- Edit the resulting scripts for tablespace names and other features, as desired.
- Create definition of indexes for the Oracle database that reflect the indexes currently on the source Rdb database, using `jcc_generate_oracle_indexes.sql`
- Edit the script to add materialized columns as needed.

2. Remove from the target all triggers and constraints that exist in the source. See "Constraints and Triggers in the Target Database" on page 133.
3. Add the Loader high-water table to the target database. See "Adding the High-Water Table" on page 129.
4. Add the dbkey columns, if any are required. (You must take pains to ensure that the dbkey values in these columns are those in the original database.) See "Adding Dbkey Columns" on page 130.
5. For each table, include one and only one unique index on the primary key to improve performance and accuracy.
6. Perform whatever physical redesign is necessary and edit the generated SQL script.
7. Load an appropriate base set of data into the target database.[1] See the next section to understand this step and the alternative approaches.
8. Establish the link back to the Rdb source database. See "Set Up Database Link Between Oracle and Rdb" on page 538 or "Initial Load of the Target" on page 128.

1. By default, the Loader sends character strings to the target Oracle database in the same format used in the source Rdb database. If the source database is defined as char(n), the string will be sent as a fixed length character string of n characters. Modifying character string formats — or other data formats — during the initial load can lead to confusion when comparing data from the initial load with data that is written by the Loader.

9. Configure the target database for performance. Be sure to add indexes on any dbkey columns that you have added. Also, consult the Oracle documentation for assistance with physical database design.
10. Review the remainder of this chapter to ensure that you have understood and addressed issues such as those discussed in "Data Types" on page 134, "Reserved Words" on page 138, and others.

Populating the Target

The JCC LogMiner Loader in conjunction with the Oracle LogMiner publish, to your target, *changes* made to the source. It is necessary to start with the target populated with the data that you will want to update.

If the target is updated by programs other than the Loader, care will be required to avoid overwriting target data in a fashion that is incompatible with your architecture for replication. See "Restoring the Initial Load" on page 129.

Initial Load of the Target

The initial population of the data may be handled in a variety of ways. Here are some that apply to Oracle targets.

1. Move the files by FTP to the target machine and load with SQL*Loader. This method requires the creation of disk files and is, therefore, the slowest method. The creation of the disk files takes a large fraction of the total time for this process.
2. Use OCI services and dblink by enabling OCI services on the source database and creating a dblink in the Oracle target. Then, use the SELECT FROM and INSERT INTO statements to move the data from Rdb tables to Oracle tables. This method requires accessing the source database which will cause some overhead. If overhead on the source is not acceptable, this may not be a good choice.
   Alternately, you can use one of the JCC procedures to generate scripts and edit those scripts for placement and other physical characteristics.
   ```
   jcc_link_data_to_oracle.com
   jcc_move_data_to_oracle.com
   ```
3. Use the Data Pump of the JCC LogMiner Loader. This method is the fastest and has the least overhead. It does, however, require that you pass all rows (that you

want in your target) through the AIJs. You will have to provide sufficient AIJ space to support this. The Data Pump is discussed further in “Data Pump” on page 505.

Continued Change While You Work

If your source database remains active, while you are preparing the target, additional changes are accumulating in the AIJ files. You can run the LogMiner and the Loader to transfer these changes to the target and catch up. See also “Quiet Points and AIJs” on page 43.

Restoring the Initial Load

The Data Pump was originally developed to address issues when target databases that had required weeks to populate became, during development of downstream applications, invalid representations of the source data. The Data Pump permits pumping subsets of the data from the source to the target to restore the synchronization of the two. The Data Pump is discussed further in “Data Pump” on page 505

Adding the High-Water Table

The high-water information is used by the Loader to keep track of what has been processed. For Oracle targets, the default is to maintain the information in a high-water table that can be updated within the same transaction that writes the source transaction updates to the target database.[1]

The highwater information is required for recovery from failures and shutdowns.

An SQL script is provided to define the high-water table for an Oracle database.

```
create_logminer_highwater_oracle.sql
```

1. Targets other than Rdb and Oracle must use a file alternative to the high water table which means that they cannot update the checkpoint within the transaction which writes the source transaction updates to the target.

This script creates one tablespace and the required high water table. This script should be edited to locate the tablespace as appropriate to the database and system. You should not change the table names, column names, or data types.

This script should be run on the same node as the target database.

Populating the HighWater Table

The highwater table is not created with any data rows. The first time that you run the Loader, this will be corrected. See “Running the Loader for the First Time” on page 102.

Recovering from a Failed or Shut Down Session

The Loader maintains its own context with either a high-water table or a checkpoint file. For database targets, the database table is generally the better approach.

By default, the Loader will restart where failure or shutdown occurred.

The chapter “Aids for the Administrator” on page 409 includes a section of Rdb issues that are current as of this writing. Since the source database is Rdb based, these apply even when the Loader target is Oracle. See “Rdb Issues” on page 412 for a rare, but serious Rdb issue with restart and backup.

Adding Dbkey Columns

For the Loader to support data updates and deletes, it is necessary that the Loader be able to identify, in the target, the row that has been updated or deleted in the source. This requires a column or collection of columns that uniquely defines the row — without nulls or changing values in any of those fields.[1]

See “Identifying Rows in the Target” on page 39 for a more complete discussion of the importance of keys for identifying rows in the target. That section also discusses the occasional necessity, as well as the drawbacks of using the originat-

1. Note that it is inconsistent (with the rule that all columns in the key must be unchanging and not null) for the key to require all columns in the table and expect to be able to do updates.

ing_dbkey approach and the alternative of changing the *source* to include identity attributes.

If there is no collection of columns[1] that uniquely defines the row — without nulls or changing values in any of those columns — you may need to establish a column for the originating dbkey.

There are issues in using dbkeys and JCC recommends against using them, unless there is no other option.

Data Types

When the target database is Oracle, the dbkey column, if used, is a NUMBER.

Other data type settings, when using an Oracle target, are discussed in "Data Types" on page 131

Aids to Creating Originating_dbkeys Columns

The special dbkey columns can be automatically added to tables in the target database by creating a file containing a list of such special tables, one table name per line. You should edit the generated files, as appropriate, to modify only those tables requiring originating_dbkey columns and to create and place indexes in proper storage areas, etc.

Then, execute the following command to generate the necessary scripts for adding the dbkey columns:

```
$ jcc_add_odbkey_set_index <file of table names>
```

1. See also "Identity Attribute" on page 40 for an Rdb option for creating an additional column that will be unique and not null.

This script will generate the following files:

TABLE 1. Files generated by the dbkey procedure

File	Purpose
<file_root>_source.sql	An SQL script to create views in the source that include the dbkey
<file_root>_odbkey.sql	An SQL script to add the originating_dbkey column to each table in the target database. This generated script will also populate the new column with the current dbkeys of existing rows. (This assumes that the database being so managed reflects the source database exactly.)
<file_root>_set_odbkey.sql	An SQL script to populate the new columns in the target database with the current dbkey of the column in the source database. (The procedure reflects an assumption that the target is an exact replication.)
<file_root>_index.sql	An SQL script to create necessary indexes on the new dbkey columns. These indexes are sorted indexes.
<file_root>_drop.sql	An SQL script to drop the originating_dbkey columns.
<file_root>_ unload_-load.com	A command procedure to unload views from the source database and load into the target.

You should edit the generated files as appropriate to modify only those tables requiring originating_dbkey columns and to create and place indexes in proper storage areas, etc.

Creating DBkey Columns Using Rdb Materialized Values

As an alternative to the Loader process, one can unload a view in which the dbkey is materialized as a column and load that result into another table. The advantage of this alternative is that there is likely to be much less fragmentation in the resulting table. To use the materialized column approach, you will need a patch that is available in Rdb 7.0.6.1 or a later version of Rdb that allows RMU to unload this materialized value.

Identity Attributes

It is possible to avoid DBkey columns with an identity attribute in the source.[1] Since creating an identity attribute, if one does not already exist, requires modifying the source database, this approach may not be acceptable in your environment.

Constraints and Triggers in the Target Database

The order of records in the LogMiner output cannot be predicted. If the source database enforces a referential integrity constraint and a transaction adds both parent and child rows in a single transaction, then it is possible that the Loader will first receive the child row and then the parent. If a referential integrity constraint is present in the target database, this sequence will cause the Loader to fail.

Similarly, if a column is maintained in the source by a trigger and is written to the target by the Loader, then the trigger is unnecessary in the target database. In fact, it is inappropriate. The rows that are maintained by the trigger will appear in the LogMiner output themselves and the Loader will similarly process them. The result is that the triggered actions will occur anyway. Since the order of records in the LogMiner output is unpredictable, it is possible that the firing of a trigger in the target database could result in inaccurate data.

For replicated databases, all database integrity and triggers will be maintained by the source database. In the target database, these actions are completely unnecessary and inappropriate.

In instances other than replication, there may be triggers that are appropriate to the target that were not in the source. In other instances than replication, there may also be constraints that are used. However, constraints should be limited to tables that are *not* maintained by the Loader and do *not* include data that is subject to foreign key constraints or check constraints involving data that is maintained by the Loader.

1. Identity attributes require Rdb version 7.1.0.2 or later.

Targets that Are Different from the Source

The target database will contain some or all tables that are present in the source database. The table names in the target may differ from those in the source although, by default, they are the same. When the target is an Oracle database, the schema name may also be specified in the maptable definitions in the Control File.

Similarly, the names of the columns in the target database may be anything you wish. The names used by the Loader are specified in the Control File. The relative order of columns in the target database table is not important.

Note that you may also use multiple tables to receive the data from one source table and may filter which target table receives which data. Alternately (or in addition), you may direct the data from multiple tables (in the same or different databases) to one target table, *if the keys are appropriate.* These mappings of source to target are specified in the Control File.

See “Schema and Data Transforms” on page 489 for additional discussion.

Login Credentials

The validation keyword will be used to provide login credentials. See “Keyword: Validation” on page 290.

Data Types

There are a few issues to consider that are best summarized as issues of data types.

The data types of corresponding columns in the target database must be compatible with data from the source database columns. For instance, it would be inappropriate to attempt to convert text columns in the source database to numeric columns in the target database unless you are guaranteed that no data conversion exceptions will be generated or unless you use the MapResult keyword to intentionally transform the data.

Comparing Character Data

Oracle handles variable length strings differently than Rdb does. This has led to some questions, followed by enhancements to the Loader, followed by additional questions. The questions mainly revolve around comparing text data written to the target by the Loader with data written to the target by some other product. The other product, for example, might be SQL*Loader used to populate the target initially or the product might be some downstream process that also modifies the target. A partial solution to handling varchar data in an Oracle target was to provide the trim option in the Loader.[1]

However, if the Rdb data is all spaces, the trim option trims to zero characters and Oracle interprets this as a null.

The Loader provides a way of defining how to interpret null.[2]

A zero length column is consistent with the Rdb definition of zero length and was not, originally, interpreted as null in the Loader definition. Since Oracle processes this as null, the Loader extends the interpretation of null such that trimmed values that Oracle would interpret as null are set to the value specified for ifnull.

If your use has the following characteristics, there is still an issue

- You are using an Oracle target with the trim option
- You have fixed length character columns in the declared key of the table (CHAR(x) datatype)
- The character data in any of those columns contain trailing spaces

Oracle's SQLPlus, when presented with a trimmed VARCHAR2 representation of a column will insert, into fixed length columns, data that is padded to the right. This same trimmed value does not match the stored padded value. Therefore, the column will not match on subsequent lookups of the row. This will cause the Loader to insert the same row a second time, which will likely result in a uniqueness constraint/index failure. This can be addressed by changing the fixed length column to VARCHAR2.

1. See "Keyword: Output" on page 266 and the optional parameter "<output conversion> optional" on page 267 which includes the TRIM conversion.
2. See "Keyword: MapColumn" on page 252 for the optional parameter "<value if null> optional" on page 252.

Timestamps

The Loader supports the Oracle 9i timestamp data type in the following way: When using Oracle 9.0 (or later), the Loader will send date data to the Oracle target as timestamp(7), seven digits of precision is the maximum available on OpenVMS. If the target column in the Oracle target is declared as a timestamp data type, the fractional seconds (available in the seven digits) will be stored in the column. If the target column is a date data type, the fractional seconds will be truncated.

Note that timestamp data is only available beginning with 9i. When using the Oracle 8.1 client, because 8i does not support timestamp data, the Loader will send only date format data. Thus, no precision to the seconds is available.

Datetime in the Key

When Oracle stores a value in a column of a different, but compatible, data type, it does a conversion. Similar conversion is not automatic when Oracle compares date values. For the Loader, this has created a problem with date columns other than TIMESTAMP(7) that participate in the key. Consequently, the generated SQL in the Loader explicitly converts the stored column into the same data type used by the Loader for date columns.

For Oracle 9i and higher

```
to_date(<target column>,'DD-MON-YYYY HH24:MI:SS.FF') = <Loader column>
```

Data conversion permits finding matching rows in the target database to achieve correct updates. However, conversion may reduce performance.

Conversion logic necessitated by timestamp columns as part of the primary key can be avoided by using the originating_dbkey mechanism or by adding an identity column to the source database.[1]

Date Format Overrides

As described in the previous section, the JCC LogMiner Loader, when replicating data to targets that require datetime conversion from the Rdb internal 8-byte binary

1. See “Identifying Rows in the Target” on page 39.

format, translates the value to a target-specific textual representation of the value. The default datetime formats for Oracle targets can be stated as:

Target	Default Date Format
Oracle 8i	date_format~|!Y4-!MN0-!D0|!H04:!M0:!S0|
Oracle 9i	date_format~|!Y4-!MN0-!D0|!H04:!M0:!S0.!C7|

The JCC LogMiner Loader supports overrides to the pre-defined date format for Oracle and JDBC target. To utilize this enhancement, the DATE_FORMAT keyword must be defined in the Control File after the OUTPUT keyword and before the first date column is declared. See “Statement Ordering” on page 215.

When the DATE_FORMAT keyword is correctly used, a warning (shown in red in the example) appears in the Loader log file. The message is informational. It is written to the log, but does not cause the Loader to shutdown.

```
%dba_parse_init_file: A date format has already been specified
file: JCC_LML_TARGET_INI
line: DATE_FORMAT~|!Y4-!MN0-!D0|!H0:!M0:!S0.!C6|
**** WARNING ****    You are overriding the default date format.
**** WARNING ****    An incorrect format will cause the Loader to fail
**** WARNING ****    or write incorrect data.
```

National Language

If you utilize a character set for Japanese, Swedish, or some other options, you may need to consider issues related to character sets. There are multiple places that character set settings can be made, including OpenVMS through logical names, Rdb at the column level, Oracle client and Oracle server.

The LogMiner Loader does not interpret bytes to modify the character set.

If you are using an Oracle target, NLS_LANG, for certain character sets, must be set to achieve the correct interpretation of the data in your source Rdb database. See the Oracle documentation for additional discussion, as it is the best resource for understanding this feature. However, at JCC, we have found NLS_LANG=AMERICAN_AMERICAN.WE8DEC appropriate for Rdb databases using the traditional DEC_MCS character sets. See also “NLS Language Setting Example” on page 564.

Note: If your application reads and writes its own special character set in Rdb, there may or may not be an Oracle NLS setting that will provide the desired translation.

Reserved Words

Column names in the Oracle target database must not be of the form c*N*, where *N* is a whole number between 1 and 2047, inclusive.

To update the target database, the Loader generates variable names in the form c*N*. Then, since in Oracle there is no syntactic difference between a variable name and a column name, for a statement of the form

```
select rowid into t_rowid from TABLE1 where c1 = c1 ;
```

Oracle does not correctly parse the ambiguous reference.[1]

A column name of the form c*N* will generate an exception message of the following form:

```
%dba_parse_init_file: Column name 'C1' is a reserved word for OCI
output in file: T_MAPTABLE.INI
line: MapColumn~T1~C1
%dba_initialize: %DBA-E-INV_INIT_RECORD, Invalid intialization
record encountered.
%DBA-E-INV_INIT_RECORD, Invalid intialization record encountered.
```

FIGURE 3. Exception for Oracle Reserved Words

Performance

The Loader, through these notes and through added features, offers assistance with some performance related issues for Oracle targets.

1. Declared variables in Rdb are prefixed with a colon. Thus, for Rdb targets the generated variable names and column names of this form can be distinguished.

Oracle RAC Load Balancing and the Loader

Oracle Real Application Clusters (RAC) can be configured such that several computers provide access to a single Oracle database (instance). As a performance feature, Oracle allows the configuration of a timed delay of data change propagation among the participating computers. While this delay can reduce the network communication traffic among the Oracle software on the participating computers, it can provide a transactionally inconsistent view of the data to client processes.

A caution is required when using the Loader with Oracle RAC. The inconsistency induced by the RAC load balancing is apparent when a writer on one computer inserts a row into a table using a unique key and a reader on another node attempts to read that row using the same key values after the writer commits, but before the data propagation interval expires. The reader will be informed by the instance to which it is attached that the row does not exist. If the reader, then, attempts to insert a new record with the same key value, the insert will fail with a duplicate key value.[1]

The JCC LogMiner Loader, using parallel[2] processing, creates a number of threads, each of which updates the target data source. Use of the load balancing features in the client software for Oracle RAC distributes the Loader threads to different computers. The pairing of Oracle RAC with propagation delay and the Loader with parallel processing can, therefore, yield transactional inconsistency. When this situation is encountered the Loader session will shutdown.

JCC recommends that, when using the parallel option of the JCC LogMiner Loader, all threads configured to update the same data also be configured to update the same target computer of the Oracle instance.

Performance and the Query Cache

JCC recommends attention to the option of specifying the Oracle parameter 'session_cached_cursors' for output tables defined in the Loader Control File. An Oracle whitepaper titled "Designing Applications for Performance and Scalability" contains the following:

1. See "Inserts and Updates" on page 41 for explanation of Loader insert and update.
2. Using parallel processing and Loader threads for performance enhancement is discussed in "Parallelism and Loader Threads" on page 383.

> "Applications of category 2 are identified by repeated (soft) parse of identical SQL statements. On the Oracle server side, this has the implication that the same values repeatedly are assigned to the server side information about cursors. This behavior can be modified by allowing the server to keep information available for frequently parsed SQL statement, at the expense of the need to lookup such SQL statements. The parameter **session_cached_cursors** can be used to do exactly that: If set to an integer value, the Oracle server engine will attempt to keep that many cursors in each session parsed and ready for execution. Appropriate values for the **session_cached_cursors** parameter depend on the Oracle release. In Oracle9*i* Release 2 and later, values as high as several hundreds can be used, in earlier releases, value above 10 to 20 are not likely to be useful as more CPU usage has a tendency to negate the effect of increased scalability using the parameter."

Loader SQL Choice and Oracle Target Performance

Originally, for Continuous LogMiner Loader using an Oracle target, each SQL statement required a parse (hard or soft) by the Oracle engine to determine if the query existed in the query cache. The necessity was caused by the Loader's use of ANSI SQL which implicitly eliminated the reuse of cursors. Such failure to reuse the cursors resulted in extraneous CPU usage and network interaction between the Loader client and Oracle server.

Beginning with Version 3.1.2 of the Loader, ANSI SQL is replaced with Oracle SQL. The change means that cursors are reused for all significant Loader queries.

The extent to which this change in the Loader improves performance for a given Loader family is dependent on the Control File configuration, profile of source database updates, network latency, and system loading.

Tuning for the Virtual Address Space

Some Loader users have found that, with an Oracle target, attaches to the Oracle database would constantly increase the PGA usage until the virtual address space was exhausted. This is the result of a bug in the Oracle server related to the usage of the UROWID datatype. The Loader, by default, uses the UROWID datatype in the generated SQL for updating the target database.

Although this is due to an Oracle bug, the Loader can provide a workaround.[1]

Beginning with Version 3.1.3, the Loader provides the option of overriding the default behavior. This option is provided as a logical name so as to not change the default behavior of the Loader. If you are encountering the problem, you may avoid it by defining the logical name.

```
$ define JCC_LML_USE_ORACLE_ROWID "1"
```

The value of the logical name is not important so long as it is longer than the empty string. If the logical name is not defined or is defined to the empty string, the Loader will use the default UROWID datatype. If the logical name is defined to something other than the empty string, the Loader will use the ROWID datatype.

Optimization of Inserts

By default, the Loader attempts to update a row and, if the row is not found, does an insert. For environments in which most operations are inserts, performance enhancement can be achieved by reversing this.

Setting the logical name JCC_LML_OPTIMIZE_INSERT to 1 will cause the Loader to reformat the compound SQL statement that it creates to attempt an insert first and, only if a duplicate exception is raised, then update the target row. Setting this logical name for an Oracle target can improve performance significantly, for *certain* workloads.

Note that unique indices must exist for the declared primary keys to ensure that a duplicates exception is generated.

Alternative Control for Exceptions in the Target

If a target encounters an exception, the Loader will try the operation again for the number of tries specified in the Control File and then shutdown. (See “Keyword: Output_failure” on page 269.) This approach is suitable for deadlock or other transient difficulties. If there is too small a memory allocation or other the target exception that will not clear up, the default behavior can prove inconvenient.

1. Reference Oracle bug #6739842: “UGA KEEPS INCREASING WHEN A PL/SQL WHICH HANDLES UROWID EXECUTING”. The bug appears to be in Oracle versions beginning with 9.0.1 and is fixed in Oracle version 11.2. It is also possible to request approved Server Patch Set 11.1.0.7 and approved Server Patch Set 10.2.0.7.

Setting the logical name JCC_LML_ORA_ROLL_DISC to 1 will modify the default behavior. With this setting, if the Loader encounters an exception that requires a rollback of a transaction for an Oracle target, the Loader will disconnect from the target and then re-attach to the database when it attempts to transmit the transaction again.

Backup and Quiet Points

It is appropriate to start with the backup copy of your database. JCC recommends that you establish an arbitrary point (epoch) at which you know that the database has no active update transactions. This is otherwise known as a *quiet point.*[1] You should, then, perform both a database and an AIJ quiet point backup at that epoch. You now have a stable starting point.

If your source database remains active, while you are preparing the target, additional changes are accumulating in the AIJ files. You can run the LogMiner and the Loader to transfer these changes to the target and catch up. See also “Quiet Points and AIJs” on page 43.

Next Steps

Your target database is now prepared and you can focus on the Control File to tell the Loader what you want to do. The chapters on other targets may be of no interest to you.

1. See also “Preparing the Source Database” on page 97.

CHAPTER 10

JDBC Loader Targets

The JCC LogMiner Loader supports transmission of source database changes to an end target via a class 4 or later JDBC driver. Therefore, JCC LogMiner Loader end targets, potentially, include any database or other platform that has a class 4 or later JDBC driver available.

The JCC LogMiner Loader first included JDBC target options with Version 3.0. With Version 3.2, the JDBC target interface was completely rewritten with substantial performance improvements, but with upward compatibility protected.[1]

Not all of the possible JDBC end targets have been explicitly tested with the Loader.

If you are not using JDBC as the Loader target, you can skip this chapter without negative impact on your understanding of the Loader.

1. See "Output Keyword" on page 150 for the single exception.

If you are new to using the Loader and are also new to JDBC and to your end target, you may want to, first, test the Loader features and learn the Loader concepts while using Rdb as a target. Because it is the default and because you will not have to learn the features of JDBC and or your end target at the same time, learning on a sample Rdb target is likely to provide a better understanding.

JDBC Drivers

There is a wide selection of JDBC drivers available. None are included with the Loader kit.

For a product to use JDBC to be the ultimate target of the Loader updates, there must be a class 4 JDBC driver available for that product. At JCC, the first Loader/ JDBC driver combinations tested were those that our clients specifically requested. Since then, JCC testing has embraced a wide range of end targets both in support of clients and in support of our own research and the list grows.

If the JDBC driver for your product is not on the list, you should validate that it runs properly with the supported versions of Java.

If the product that you wish to use does not have a class 4 JDBC driver or that driver is not on the JCC list and does not appear to work, JCC recommends using the XML target for the Loader and using the API example in the kit to create your own API.

Drivers, Versions, and Acknowledgments

Using JDBC with the Loader incorporates software developed by a variety of sources. Acknowledgments are provided in this section.

Regression testing for the Loader is extremely thorough. Additional products or versions *may* be compatible even if they have not been tested.

Tested Products and Versions

See the blog at http://www.jcc.com/lml-jdbc-targets.

Compatibility Requirement

It is necessary that the driver version you use is certified for the JAVA version that you are planning to use.

Transferring JAR Files via FTP

Java JAR or CLASS files that are transferred to OpenVMS using FTP must have the file attributes modified to be accessible to Java on OpenVMS. The following command sets the appropriate file attributes.

```
$ set file/attr=(rfm:stmlf,rat:cr,lrl:0,mr:0)    -
  <JAR file name>
```

Additional References

General information on JDBC may be found at

http://www.oracle.com/technetwork/java/javase/jdbc/index.html

Detailed information about installing the HP Java for OpenVMS systems may be found, if you login, at

https://www.hpe.com/global/java/

User Procedure for JDBC

To utilize the JCC LogMiner Loader JDBC interface, it is necessary to execute the JDBC specific user procedure. After executing the jcc_lml_user procedure, next execute the following.

```
$ jcc_lml_jdbc_user                              -
    <Java version>                               -
    <Java setup procedure>                       -
    <Java setting>
```

The third parameter, Java setting, is passed directly to the procedure named in the second parameter. 'FAST' may enhance performance on alpha systems. However, note that there are memory implications for using the fast Java engine.

Other options for the third parameter for version 1.4.2 are 'CLASSIC' and 'HOTSPOT'. 'HOTSPOT' is not available on alpha. On Integrity, 'HOTSPOT' is the only option recognized. On Integrity, all others, including no third parameter, are interpreted as 'HOTSPOT'.

Please consult your JAVA documentation.

Example

```
$ jcc_lml_jdbc_user 1.4.2                        -
sys$common:[java$142.com]java$142_setup FAST
```

Defaults

Defaults for this procedure can be defined using the logical names

```
$ define JCC_LML_JDBC_DEFAULT_VERSION 1.4.2
$ define JCC_LML_JDBC_DEFAULT_SETUP              -
sys$common:[java$142.com]java$142_setup
```

Java Command Line Options

The logical name JCC_LML_JAVA_COMMAND_LINE specifies command line options to the Java Virtual Machine. The supported OpenVMS Java command line options are documented in the *Software Development Kit (SDK) v 1.4 for the OpenVMS Operating System for the Java Platform User Guide* from HP.

Warnings

1. Some command line options change the way that the Java engine interprets Java code. Avoid using these options.
2. The options should be specified in quotes and should be space delimited. (Java command line options are case sensitive.)
3. The JCC LogMiner Loader supports a maximum of thirty command line options.
4. Should it be desirable to use command line options to increase the Java Virtual Machine memory and stack size, it may also be necessary to increase VMS process quotas.

Applicability

An example of why you may want to use the Java command line options is to control the Java Virtual Memory. An example of when you may need to do this is discussed in “jcc_lml_jdbc_batch_size” on page 152.

Example

```
$ define JCC_LML_JAVA_COMMAND_LINE "-Xmx96m -verbose:gc"
```

Although this is a reasonable example of how to pass two arguments, *JCC does not recommend using the second argument unless debugging JAVA memory issues.*

Systems Tuning Using JDBC as the Loader Target

Both using JAVA on OpenVMS and using the Loader with a JDBC target may introduce systems tuning requirements.

HP's recommendations for the JAVA SDK user guide are quoted in "Java and OpenVMS" on page 148. The user guide section quoted is titled "Setting Process Quotas for Better Performance on OpenVMS." Attention to the vendor's tuning options is strongly recommended.

Java and OpenVMS

"The Java runtime environment was designed to perform optimally on UNIX systems, where each process is given large quotas by default. On OpenVMS, the default behavior gives each process lower quotas so that many processes can coexist on a system.

To get the best Java performance on OpenVMS, HP recommends that you change some setting to what might be expected for a typical UNIX system. In the recommended list Channelcnt is an OpenVMS sysgen parameter, The rest are authorization quotas. HP recommends these as the *minimum settings* (except where noted).

UAF Fillm	4096
Channelcnt	4096
Wsdef	2048
Wsquo	4096
Wsextent and Wsmax	16384
Pgflquo	2097152[a]
bytlm	400000
biolm	150
diolm	150
tqelm	100

a. A good number for `Pgflquo` is (2 x heap-size, for example, 128 MB (2*128*1024*1024)/512=524288. Recall that the recommended minimum `Pgflquo` is 96 MB when using the RTE. When you increase the `Pgflquo` parameter, you should always increase the system's page file size to accommodate the new `Pgflquo` parameter, if needed."

Other Tuning

Additional tuning tips are included in the “Logical Names to Use with JDBC” on page 151 and in “User Procedure for JDBC” on page 146.

Loader Tuning and the JDBC Interface

Because the Loader target of JDBC is used with a wide range of end targets, not all situations can be addressed with tuning of the Loader itself. There are opportunities, however, to address likely uses of the JDBC target. One is handling of batch updates.

When the Loader reads data from the LogMiner, there is an indication that the record is either a delete or a modify. A modify may be either an insert of a new row or an update to an existing row. The Loader tries an update and, if the row is not found, does an insert.

The Loader may be set to include multiple transactions in what is written to the target in a single transaction. (See Keyword: Checkpoint in the full documentation.) When using the JDBC target, it is also possible to set the JDBC batch size.

When a batch includes an insert for a row and, later, includes an update for that row, the JDBC code will first send the batch to the UPDATE statement. Since the row does not yet exist, both the insert and the update rows will have no affect. Both rows will next be sent to the INSERT statement. The first record will succeed, but the second will fail due to a duplicate value.

When this happens, in the previous release, the Loader temporarily disabled batch mode and sent each record, one at a time, to the update and failing that the insert. The data was eventually correct. However, each row in the batch was processed, including rows that were already updated.

That approach meant unnecessary work and obscuring of temporal markers that are important for some applications.

The solution is to keep track of which records in the buffer have been successfully processed so that if the Loader needs to fall back and process rows individually, the Loader will only process ones that have not yet been successful.

JDBC and the Loader Control File

The aspects of the Control File to check for JDBC specifics are

- The output keyword has a jdbc option. See “Keyword: Output” on page 266.
- The validation keyword will be used to provide login credentials. See “Keyword: Validation” on page 290.
- The JDBC keyword will be used to specify behavior. See “Keyword: JDBC” on page 242.
- The checkpoint keyword will be used to specify how to handle the highwater information.

Output Keyword

To enhance performance with large transactions, the specification of the Output keyword is constrained. Beginning with version 3.2, “record” is the default as the message contents parameter of the Output keyword. Further, when using JDBC targets, “transaction” is no longer supported.[1] The following from the regression test shows an example of explicitly specifying record as the message contents parameter:

```
output~jdbc~synch~                                              \
jdbc:jtds:sqlserver://thor:1434;DatabaseName=RegTest~record
```

If the message contents parameter is included as “transaction” instead of “record”, the following exception message will be given

```
%dba_parse_init_file: invalid output message contents in file: <file>
line: <input line>
Valid value for JDBC is RECORD.
```

1. See “Keyword: Output” on page 275.

Of course, '<file>' and '<input line>' will be replaced with the file name of the control file and the specific input line that caused the message.

Logical Names to Use with JDBC

The logical names are available to tailor Loader operations to the JDBC driver that you are using. Due to the great variety of JDBC drivers and the products to which they write, something introduced as a performance improvement for JDBC targets may not be appropriate for the driver that you choose.

JCC_lml_jdbc_default_version

This logical name provides input to the User Procedure for JDBC. See "Defaults" on page 146.

JCC_lml_jdbc_default_setup

This logical name provides input to the User Procedure for JDBC. See "Defaults" on page 146

JCC_java_command_line

JCC_java_command_line provides the opportunity for you to modify what commands are given to the java engine. Discussion of uses occurs in these sections:

- "Java Command Line Options" on page 147
- "Multinational Character Sets in JDBC Targets" on page 164
- "Processing of Timestamp, Date, and Time Columns" on page 171
- "Update Only Operation and the JDBC Interface" on page 169
- "Schema Separators" on page 168

jcc_lml_jdbc_single_statement

Best performance is achieved when multiple, concurrent open statements can be used. Not all JDBC drivers support multiple, concurrent open statements.

For drivers that do not support multiple, concurrent open statements, set the logical name JCC_LML_JDBC_SINGLE_STATEMENT to 1. This causes the Loader to use single statements only.[1]

jcc_lml_jdbc_batch_disable

Best performance is obtained using JDBC batching. Not all JDBC drivers support generic JDBC batching sufficiently for the purposes of the Loader. In fact, not all of the drivers that JCC has tested support multiple, concurrent prepared statements.

If the Loader fails, stating that the update status returned from the end target is "-2", you will not be able to use batching. The "-2" status means that the Loader command succeeded, but the number of rows updated is unknown. The Loader must distinguish between zero or more than one rows updated to determine whether future action is required to replicate the source row.

To disable batching, set JCC_LML_JDBC_BATCH_DISABLE to 1. This causes the Loader to limit the batch size to one row and forces use of executeUpdate rather than executeBatch.[2]

jcc_lml_jdbc_batch_size

Increasing batch size may increase performance. JCC_LML_JDBC_BATCH_SIZE may be defined as an integer. The integer specifies the maximum number of records to include in a batch.

---

1. The SQL Server 2000 JDBC driver tested does not support multiple, concurrent open statements. Therefore, with this driver, the Loader runs as if this logical name is set to 1, regardless of other definitions.
   Newer SQL Server drivers and the jTDS Sourceforge driver did not exhibit the problem.
2. All Oracle drivers tested returned "-2" when using executeBatch. Therefore, the Loader disables batch updates for Oracle JDBC drivers, by default, regardless of the setting of this logical name. For most installations this will not be an issue because the OCI interface will be used for Oracle target databases.
   Note that this issue does not pertain to Rdb, but getting the latest version of Rdb is particularly important if you find reason to use a JDBC Loader target to write to an Rdb end target.

Batches of records are collected per table. When a new table is encountered, the batch of records collected for the previous table is flushed to minimize Java memory usage.

It should be noted that larger batch sizes imply larger amounts of data that must be cached within the Java Virtual Machine. Every column of each row is a separate object within Java. The expected size of each row must be increased by the object overhead required to represent each object in Java. Additionally, both the Loader and the JDBC driver will cache each row and column. Care should be taken to match the batch size to the available memory. Note that JVM memory can be modified using the logical name JCC_LML_JAVA_COMMAND_LINE. (See "Java Command Line Options" on page 147.)[1]

jcc_lml_jdbc_gc_commit

Early versions of Java (eg 1.4.2) provide garbage collection infrequently when the default serial garbage collector is used. For such a version, you can force explicit garbage collection at the end of each commit interval by defining the logical name JCC_LML_JDBC_GC_COMMIT to 1.

Later versions of Java default to a more pro-active garbage collection method which ignores any explicit garbage collection calls. Therefore, using this logical name with newer versions of Java will have no effect.

JDBC Name Delimiters

The Loader's JDBC interface requires that the Loader obtain column metadata information from the target data store. To use the metadata information, the Loader must match the target-supplied names to the user-supplied names. Since the target does not return names with delimiters (even if they are required to reference the column), the Loader provides a way of specifying the delimiters.

Two logical names are used to support the specification of target delimiters:

- JCC_LML_JDBC_NAME_DELIM_START
- JCC_LML_JDBC_NAME_DELIM_END

1. For best results when using this logical name, sort should be by_record, the default.

To use delimiters, both logical names must be defined, even if the same character is used for both starting and ending a delimited name. If only one is defined, it is ignored.

These examples might apply to SQL Server end targets.

```
$ define JCC_LML_JDBC_NAME_DELIM_START "["
$ define JCC_LML_JDBC_NAME_DELIM_END "]"
```

These examples might apply to Rdb end targets.

```
$ define JCC_LML_JDBC_NAME_DELIM_START """"
$ define JCC_LML_JDBC_NAME_DELIM_END """"
```

jcc_lml_jdbc_target_schema

See “Ability to Specify JDBC Target Schema for Metadata Queries” on page 173.

jcc_lml_jdbc_version

See “OpenVMS Java Changes Across Versions” on page 172 and “Java Version” on page 162.

jcc_lml_java_bootclasspath

See “OpenVMS Java Changes Across Versions” on page 172 and “Java 2 (a.k.a. Java 1.5) on Alpha OpenVMS” on page 176.

jcc_lml_case_sensitive_target

See “Mixed Case Names” on page 166 and “Other Mixed Case Challenges” on page 167.

Set Defaults for Logical Names Used with Java

There are logical names that can be set to control how Java is used. These have some convenient defaults that are automatically set beginning with Version 3.6. If any of these are already defined, it is assumed that the Administrator wishes to override the defaults and the default is not set.

TABLE 1. Logical Name Defaults for Java

Logical Name	Default	Effect
JAVA$FORCE_IEEE_FPSR	1	Meets the requirement for Java garbage collector to work on IPF Java 1.6.0-5 and above.
DECC$STDIO_CTX_EOL	1	Removes the extraneous linefeed from Java logging.
DECC$FILENAME_UNIX-_NO_VERSION	1	Makes changes in how C CRTL routines format VMS files as Unix files.

End Targets for JDBC

As far as the Loader is concerned, the JDBC driver specified is the target. However, the end result is to publish changes in the source database to the target of the JDBC driver. To do so, the end target must be defined and populated with the initial set of data. The sections “Preparing the End Target” on page 155 and “Populating the End Target” on page 156 apply to all targets, but are adapted here to be specific to JDBC targets. Special notes for specific end targets are included in “Additional Topics for Specific End Targets” on page 174 and “Further Notes on Companion Products” on page 175.

Preparing the End Target

The end target must be prepared. The steps are cited here. The sections to follow include details.

1. Define the end target. How you define the end target is, of course, dependent on the end target that you chose. You may want to consult Loader support to discuss what has been done for regression testing.

2. If the end target is a database, remove from that target all triggers and constraints that exist in the source. See “Constraints and Triggers in the End Target” on page 157.
3. Add the file that will contain the highwater information. See “Adding the High-Water Information” on page 158.
4. Add the dbkey columns, if any are required. “Adding Dbkey Columns” on page 159.
5. For each table, include one and only one unique index on the primary key to improve performance and accuracy.
6. Perform whatever physical redesign is necessary.
7. If you did not define the end target with a method that also populates the tables and columns with data, you must now load an appropriate base set of data. See the section to follow.
8. Populate the dbkey columns, if any. See “Adding Dbkey Columns” on page 159.
9. Configure the end target for performance.

Populating the End Target

The JCC LogMiner Loader in conjunction with the Oracle Rdb LogMiner publishes, to your target, *changes* made to the source data. It is necessary to start with the target populated with the data that you will want to update.

Initial Load of the Target

The initial population of the data may be handled in a variety of ways. The one that works consistently well is to use the Data Pump which is packaged with the JCC LogMiner Loader. This method applies to all target types. It is also the fastest and has the least overhead.

Using the Data Pump, does, however, require that you pass all rows (that you want in your target) through the AIJs. You will have to provide sufficient AIJ space to support this.

The Data Pump is discussed further in “Data Pump” on page 505.

Continued Change While You Work

If your source database remains active, while you are preparing the target, additional changes are accumulating in the AIJ files. You can run the LogMiner and the Loader to transfer these changes to the target and catch up. See also "Quiet Points and AIJs" on page 43.

Restoring the Initial Load

If the target is updated by programs other than the Loader, care will be required to avoid overwriting target data in a fashion that is incompatible with your architecture for replication. However, corrections for some errors are possible.

The Data Pump was originally developed to address issues when target databases that had required weeks to populate became, during development of downstream applications, invalid representations of the source data. The Data Pump permits pumping subsets of the data from the source to the target to restore the synchronization of the two. The Data Pump is discussed further in "Data Pump" on page 505

Constraints and Triggers in the End Target

If the end targets has the option of constraints and triggers, some caution must be observed.

The order of records in the LogMiner output cannot be predicted. If the source database enforces a referential integrity constraint and a transaction adds both parent and child rows in a single transaction, then it is possible that the Loader will first receive the child row and then the parent. If a referential integrity constraint is present in the target database, this sequence will cause the Loader to fail.

Similarly, if a column is maintained in the source by a trigger and is written to the target by the Loader, then the trigger is unnecessary in the target database. In fact, it is inappropriate. The rows that are maintained by the trigger will appear in the LogMiner output themselves and the Loader will similarly process them. The result is that the triggered actions will occur anyway. Since the order of records in the LogMiner output is unpredictable, it is possible that the firing of a trigger in the target database could result in inaccurate data.

For replicated databases, all database integrity and triggers will be maintained by the source database. In the target database, these actions are completely unnecessary and inappropriate.

In instances other than replication, there may be triggers that are appropriate to the target that were not in the source. In other instances than replication, there may also be constraints that are used. However, constraints should be limited to tables that are *not* maintained by the Loader and do *not* include data that is subject to foreign key constraints or check constraints involving data that is maintained by the Loader.

Adding the High-Water Information

The high-water information is used by the Loader to keep track of what has been processed. For Rdb and Oracle targets, the default use is to maintain the information in a highwater table that can be updated within the same transaction that writes the source transaction updates to the target database. For targets other than Rdb and Oracle, it is necessary to store the high water information in a file.

Since no highwater data is stored in the JDBC target, an LML_INTERNAL checkpoint must be used. LML_INTERNAL checkpoints to the local highwater file. The LML_INTERNAL checkpoint is established with the checkpoint keyword. See the Control File chapter for details.

The highwater information is required for recovery from failures and shutdowns.

Because the LML_INTERNAL checkpoint cannot be written as a part of the transaction that writes the data changes to the target, it is possible that the Loader will have written the data to the target and not yet updated the checkpoint when an interrupt occurs. Since checkpointing is designed to ensure that a source database transaction is processed completely, the source database transaction will be repeated if the checkpoint file does not indicate that it was completed. This may result in re-sending information, but it will ensure that all information is received at least once.

The Loader will re-play the transactions in Loader Sequence Number order. That will be the same as the source database transaction commit sequence.

Populating the HighWater Information

The highwater information must be initialized the first time that you run the Loader. See “Running the Loader for the First Time” on page 102.

Recovering from a Failed or Shut Down Session

The Loader maintains its own context with either a high-water table or a checkpoint file. For Rdb targets, the database table is generally the better approach.

By default, the Loader will restart where failure or shutdown occurred.

The chapter “Aids for the Administrator” on page 409 includes a section of Rdb issues that are current as of this writing. See “Rdb Issues” on page 412 for a rare, but serious Rdb issue with restart and backup.

Adding Dbkey Columns

For the Loader to support data updates and deletes, it is necessary that the Loader be able to identify, in the target, the row that has been updated or deleted in the source. This requires a column or collection of columns that uniquely defines the row — without nulls or changing values in any of those fields.[1]

If there is no collection of columns[2] that uniquely defines the row — without nulls or changing values in any of those columns — you may need to establish a column for the originating dbkey.

There are issues in using dbkeys and JCC recommends against using them, unless there is no other option.

See “Identifying Rows in the Target” on page 39 for a more complete discussion of the importance of keys for identifying rows in the target. That section also dis-

1. Note that it is inconsistent (with the rule that all columns in the key must be unchanging and not null) for the key to require all columns in the table and expect to be able to do updates.
2. Note that it is also possible to use the Rdb option for creating an additional column *in the source* that will be unique and not null.

cusses the occasional necessity as well as the drawbacks of using the originating_dbkey approach and the alternative of changing the *source* to include identity attributes.

Data Types

JDBC uses BIGINT as the data type for the DBkey column.

Aids to Creating Originating_dbkeys Columns

The special dbkey columns can be automatically added to tables in the target database by creating a file containing a list of such special tables, one table name per line. You should edit the generated files as appropriate to modify only those tables requiring originating_dbkey columns and to create and place indexes in proper storage areas, etc.

Execute the following command to generate the necessary scripts for adding the dbkey columns:

```
$ jcc_add_odbkey_set_index <file of table names>
```

The command will generate the following files:

TABLE 2. Files generated by the dbkey procedure

File	Purpose
<file_root>_source.sql	An SQL script to create views in the source that include the dbkey
<file_root>_odbkey.sql	An SQL script to add the originating_dbkey column to each table in the target database. This generated script will also populate the new column with the current dbkeys of existing rows. (This assumes that the database being so managed reflects the source database exactly.)
<file_root>_set_odbkey.sql	An SQL script to populate the new columns in the target database with the current dbkey of the column in the source database. (The procedure reflects an assumption that the target is an exact replica.)
<file_root>_index.sql	An SQL script to create necessary indexes on the new dbkey columns. These indexes are sorted indexes.

TABLE 2. Files generated by the dbkey procedure

File	Purpose
<file_root>_drop.sql	An SQL script to drop the originating_dbkey columns.
<file_root>_ unload_-load.com	A command procedure to unload views from the source database and load into the target.

You should edit the generated files as appropriate to modify only those tables requiring originating_dbkey columns and to create and place indexes in proper storage areas, etc.

Creating DBkey Columns Using Rdb Materialized Values

As an alternative to the Loader process, one can unload a view in which the dbkey is materialized as a column and load that result into another table. The advantage of this alternative is that there is likely to be much less fragmentation in the resulting table. To use the materialized column approach, you will need a patch that is available in Rdb 7.0.6.1 or a later version of Rdb that allows RMU to unload this materialized value.

Identity Attributes

It is possible to avoid DBkey columns with an identity attribute in the source.[1] Since creating an identity attribute, if one does not already exist, requires modifying the source database, this approach may not be acceptable in your environment.

JDBC Targets and the Log File

When using a JDBC target, the JCC LogMiner Loader log will show additional information that is useful in debugging issues.

1. Identity attributes require Rdb version 7.1.0.2 or later.

Java Version

When using a JDBC target, the version of Java used when initializing the JVM will appear in the log file. For example,

```
Java version: 6.0
```

Retries

When a JDBC target is used and an exception occurs that requires retrying a record buffer, a message is logged if buffering is disabled. The message includes the source table, target table and, in parentheses and with labels, the number of records in the buffer and the action (insert, modify, delete).

The message will have the format:

```
Retrying <src table>-><tgt table>(size:<records in
buffer>, act:<action>) buffer with batch support
disabled...
```

Different JDBC Drivers and the Logs

JDBC drivers vary in how target exceptions are reported. Some provide a single exception message that reports the cause of the problem; others provide a list of exceptions which, taken as a whole, report the problem. The Loader reports the entire list, if it is available.

Data Types and Details with JDBC Targets

The Loader interface to JDBC provides access to all data types supported for other targets. However, there are a few issues to consider that are best summarized as issues of data types.

The data types of corresponding columns in the end target must be compatible with data from the source database columns. For instance, it would be inappropriate to attempt to convert text columns in the source database to numeric columns in the target database unless you are guaranteed that no data conversion exceptions will be generated or unless you use MapResult to intentionally transform the data.

Dates

The JDBC interface for the Loader writes all date data type columns (including intervals) as timestamps with seven fractional seconds of precision.

Date Format Overrides

The JCC LogMiner Loader, when replicating data to targets that require datetime conversion from the Rdb internal 8-byte binary format, translates the value to a target-specific textual representation of the value. The default datetime formats for JDBC targets can be stated as:

```
date_format~|!Y4-!MN0-!D0|!H04:!M0:!S0.!C7|
```

The JCC LogMiner Loader supports overrides to the pre-defined data format for non-Rdb[1] targets. To utilize this enhancement, the DATE_FORMAT keyword must be defined in the Control File after the OUTPUT keyword and before the first date column is declared. See “Statement Ordering” on page 215.

When the DATE_FORMAT keyword is correctly used, a warning is emitted in the Loader log file. The message is informational only. It is written to the log file, but does not cause the Loader to shutdown.

The example shows the warning in red.

```
%dba_parse_init_file: A date format has already been specified
file: JCC_LML_TARGET_INI
line: DATE_FORMAT~|!Y4-!MN0-!D0|!H04:!M0:!S0.!C6|
**** WARNING ****      You are overriding the default date format.
**** WARNING ****      An incorrect format will cause the Loader to fail
**** WARNING ****      or write incorrect data.
```

Trim

The Output keyword (“Keyword: Output” on page 266) includes a conversion option that can trim trailing blanks, carriage returns, and line feeds from a column value. Since not all targets will process the result as you might be expecting, it is

1. Date format override can also be used with Rdb targets, but is unnecessary because Rdb targets can use the OpenVMS binary data type.

also wise to read “Comparing Character Data” on page 135 in the Oracle chapter and the discussions in “Keyword: MapResult” on page 257.

Binary Data in Character Columns

The XML foundation for the first JDBC target release did not support binary data in character columns. That limitation was removed with Version 3.2 of the Loader.

Multinational Character Sets in JDBC Targets

One goal in implementing the JCC LogMiner Loader is to ensure that the character data in the source database is faithfully transmitted to the target database. Attempts to use the Loader with a source that uses a character set that is not the default OpenVMS Multinational Character Set and with a JDBC target revealed a need for a more sophisticated approach. To understand the issue requires an understanding of OpenVMS character set encoding and of available Java ISO character sets, as well as JDBC targets.

The default character set on OpenVMS is “DEC-MCS”[1]. This is a character encoding created by Digital Equipment Corporation in the 1980s for use in the VT220 terminal. It is an 8-bit extension of ASCII that added accented characters, currency symbols, and other character glyphs missing from 7-bit ASCII. The 7-bit ASCII characters include the Latin characters sufficient for representing English and many Western European languages. This character set was the basis of ISO-8859-1 and is also often called Latin-1.

The OpenVMS default character set can represent only a fraction of the characters needed to represent all of the languages of the world. There is support for dozens of other languages on OpenVMS, but not by using the default.

1. To be more precise, DEC-MCS is the ancestor of the ISO-8859-1 standard character set, which is more formally known as “ISO/IEC 8859-1:1998, Information technology - 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet No. 1”. For the purposes of the printable characters DEC-MCS is a subset of the ISO standard, but there are some minor variations mostly in the control (non-printable) byte codes. Although it is not pertinent to this discussion, DEC-MCS is also the predecessor of Unicode.

When the Loader encounters character data destined for a JDBC target, it passes that data into Java classes which, in turn, provides the data to the JDBC driver. Unfortunately, the character data passed into the Java classes is interpreted using the OpenVMS default character set rather than the character set defined for the locale configured on the OpenVMS system.

Beginning with Version 3.4.4 of the JCC LogMiner Loader, the character set to use for interpreting character data can be configured by the user. To do so, add the new character set property, jcclml.mcs, to the definition of the logical name JCC_LML_JAVA_COMMAND_LINE

```
$ define JCC_LML_JAVA_COMMAND_LINE –
                          "-Djcclml.mcs=<ISO name>"
```

The following table shows the various OpenVMS encodings and their ISO names.

TABLE 3. Character Encodings

Encoding	Description	ISO Name
Arabic	Arabic (ISO)	ISO-8859-6
Cyrillic	Cyrillic (ISO)	ISO-8859-5
Greek	Greek (ISO)	ISO-8859-7
Hebrew	Hebrew (ISO)	ISO-8859-8
Latin-1	Western European (missing euro sign €) (ISO)	ISO-8859-1
Latin-2	Central European (missing euro sign €) (ISO)	ISO-8859-2
Latin-5	Turkish (ISO)	ISO-8859-9
Latin-6	Nordic (ISO)	ISO-8859-10
Latin-9	Revision of Latin-1 (including euro sign €) (ISO)	ISO-8859-15
Thai	Thai (ISO)	ISO-8859-11

This example demonstrates how to assign Hebrew as the character set interpretation:

```
$ define JCC_LML_JAVA_COMMAND_LINE –
                     "-Djcclml.mcs=ISO-8859-8"
```


Note that this logical name has many uses. The following example demonstrates how to assign Nordic as the character set interpretation and also increase the amount of memory allocated to the Java JVM to 256MB:

```
$ define JCC_LML_JAVA_COMMAND_LINE -
                  "-Djcclml.mcs=ISO-8859-10 -Xmx256m"
```

Source Columns with TINYINT Data Type

The Loader uses a Short object instead of a Byte object to pass an Rdb TINYINT. Doing so avoids passing a value that is signed, but is interpreted as an unsigned value.

In SQL Server, the TINYINT data type is an unsigned byte, rather than the signed byte that is stored in an Rdb TINYINT data type. If, for an SQL Server end target, the target column is specified as a TINYINT data type and a negative value is replicated by the Loader, an overflow exception may be generated by SQL Server. JCC recommends that columns in SQL Server that are intended to hold data from Rdb TINYINT columns be created as SQL Server SMALLINT data type columns to preserve the source database data values.

Mixed Case Names

SQL Server, MySQL, and other databases accessible to the Loader via the JDBC interface support mixed case column names. The Loader handles these correctly, but some care must be taken in specification.

Loader support for case sensitive table names is, by default, turned off to protect backwards compatibility. To enable case-sensitive behavior for table names define the logical name JCC_LML_CASE_SENSITIVE_TARGET to 1.

```
$ defube JCC_LML_CASE_SENSITIVE_TARGET 1
```

To specify mixed case table names, the user must specify for the Loader which delimiters will be used for target database object names. See the section “JDBC Name Delimiters” on page 153 for more information.

Additionally, the user must provide the translation, since the source’s case-insensitive name will not match the target’s mixed case name. For example, if ‘SOURCE_TABLE_1’ is a table in the source database and ‘ROLE_NUMBER’ is

a column in that table, those names would be equivalent, for Rdb, to ‘source_table_1’ and ‘role_number’ or any other capitalization. However, for a target that uses mixed case ‘Target’ is not the same as ‘TARGET’ or ‘target’ or ‘TarGet.’. To map from ‘ROLE_NUMBER’ in the source to ‘RoleNumber’ in the target requires using Loader Map statements, described beginning on page 251. To define the table, in the target, as ‘TargetTable1’ and the column as the primary key, named ‘RoleNumber’, requires statements like these:

```
Table~SOURCE_TABLE_1~1~~NoMAPTABLE
Primary Key~SOURCE_TABLE~ROLE_NUMBER~2~4~0~8~0
o
o
o
MapTable~SOURCE_TABLE_1~SOURCE_TARGET_1,TargetTable1~Replicate
MApColumn~SOURCE_TARGET_1~ROLE_NUMBER,[RoleNumber]
o
o
o
MapKey~SOURCE_TARGET_1~[RoleNumber]
o
o
o
```

If you are trying to get started with JDBC target and receive an exception similar to the following, you may be failing to handle mixed case appropriately.

```
Connect to the database: jdbc:mysql://localhost:2001/jcc_lml user: jeff
Driver Information
  Name: MySQL Connector Java
  Version: mysql-connector-java-5.1.35 ( Revision:
5fb9c5849535c13917c2cf9baaece6ef9693ef27 )
java.lang.IllegalStateException: No metadata for table DETAILS-
>DETAILS (DETAILS)
   at LMLTABLE.<init>(LMLTABLE.java:341)
o
o
o
```

Other Mixed Case Challenges

Whether or not SQL Server databases are case insensitive is dependent on the default collation sequence. This may also apply to other databases that support case sensitivity. If the log contains a warning that a column cannot be found in the target,

the difficulty may relate to case sensitivity issues. Column Abc may not be recognized as column ABC.

Schema Separators

The default schema separator is a period (“.”). The default schema separator may be changed with the logical name JCC_LML_JAVA_COMMAND_LINE. For example, to change to change the schema separator to “&”, use

```
$ define JCC_LML_JAVA_COMMAND_LINE "-Djcclml.schema.delim=&"
```

Timeouts

On occasion, while using the JDBC target, Loader users have had queries that never finished in the target. This suggests that an improved timeout mechanism is useful. Some JDBC drivers support a query timeout mechanism.

Beginning with Version 3.5, a timeout can be set for the Loader statements issued to the target database. If the downstream product does not support the timeout, this feature may or may not generate an exception. If the JDBC driver ignores functionality that is not implemented in the driver, the timeout will have no effect.[1] When the JCC LogMiner Loader, through the JDBC driver, executes a statement, and the statement exceeds the specified value, the Loader will log the exception and retry. If the Loader continues to receive an SQLTimeoutException, after the configured number of retries, the Loader will exit with an SQLTimeoutException.[2]

If the JDBC driver throws an exception other than SQLTimeoutException, the Loader will process the exception using the standard exception handling.

To set the timeout, use

```
-Djcclml.queryTimeout=<seconds>
```

1. If the JDBC driver being used generates an error, the Loader will catch the exception, but continue. If trace logging is enabled, the Loader will format and print the exception, each time it is encountered.
2. See Keyword: Output~failure in the full documentation.

For example, the following sets the query timeout to 3 seconds.

```
$ define JCC_LML_JAVA_COMMAND_LINE "-Djcclml.queryTimeout=3"
```

The number of retries before exiting may be set, or modified, with the keyword Output_failure. See the full documentation for how to use this keyword.

Update Only Operation and the JDBC Interface

For JDBC targets using releases prior to 3.5, configuring a table for update only (noinsert,update,nodelete) caused the Loader to exit when receiving a source row that did not exist in the target. The correct behavior, consistent with operations for other targets, is to omit the row because of the noinsert, but continue.

Beginning with version 3.5, the JDBC interface works as intended.

To protect upward compatibility, there is a way of requesting the prior behavior. Any customer who needs to maintain the original functionality of exiting when no row exists should add

```
-Djcclml.origUpdateNoinsert=true
```

to the logical name JCC_LML_JAVA_COMMAND LINE to obtain the original behavior. For example,

```
$ define JCC_LML_JAVA_COMMAND_LINE "-Djcclml.origUpdateNoinsert=true"
```

Loader Features and JDBC Drivers Diversity

Beginning with Version 3.5 of the Loader, JCC has added new features to the JDBC target to provide more flexibility for supporting as yet unknown JDBC drivers.

Because JCC cannot limit how JDBC drivers will change, we assume that additional attention to tuning will be required of users of the Loader, working with the JCC support desk.

Explicitly Loading JDBC Driver Files

Most of the tested JDBC driver JAR files, when properly specified in the Java class path, will be loaded into the Java Virtual Machine once an object within them is referenced. In rare cases, this does not happen as it should. One such driver file is the UnityDB MongoDB JDBC driver. When the file does not get properly loaded, an exception like the following is generated (though others are possible):

```
java.lang.NoClassDefFoundError: <class name>
```

Beginning with Version 3.5, the Loader can be told to enable or disable the explicit loading of the JDBC driver JAR files. The default for the UnityDB MongoDB JDBC driver is to enable explicit loading, but the default for other drivers is to disable explicit loading and use the native Java implicit loading.

The feature can be enabled by setting the property jcclml.loadjar to any of the following values: enable, true, 1 (case-insensitive). Any value defined other than these to enable the feature will disable it. Note, however, that the property "jcclml.loadjar" *is case-sensitive*, so that the value for the logical name must be enclosed in double quotes.

```
$ define JCC_LML_JAVA_COMMAND_LINE "-Djcclml.loadjar=enable"
```

If using the UnityDB MongoDB JDBC driver which is set to be explicitly loaded and the user decides to disable explicit loading of the JDBC driver JAR files, the following message will be written to the log files, but processing will attempt to continue:

```
***
*** Override of jcclml.loadjar (disabled) conflicts with expected setting.  Continuing...
***
```

JCC highly recommends that you contact support before you try to override the Loader defaults.

Processing of Timestamp, Date, and Time Columns

The JCC LogMiner Loader, when replicating data using JDBC, will request the metadata definition of the target tables to validate column names and data types. The Loader uses this data type information to provide the correct data conversions for the target data store. Most of the tested JDBC drivers, when returning the requested metadata, report the data types as stored in the target.

Unfortunately, the Oracle JDBC drivers return the java.sql.Types.DATE data type for all date, time, and timestamp columns. Columns in the Oracle target that are TIME and TIMESTAMP do not correctly process data when converted to java.sql.Date and assigned to the SQL statements using the setDate methods. However, the Oracle driver does accept date, time, and timestamp data if they are converted to a java.sql.Timestamp and assigned to the SQL statements using the setTimestamp method and so the alternate processing model uses only the timestamp data type and method.

Beginning with Version 3.5, the Loader can be instructed to enable or disable the use of this alternate processing model which passes all date, time, and timestamp data passing only a java.sql.Timestamp data. The default for the Oracle JDBC drivers is to disable use of the native JDBC date/time data types and methods in favor of using java.sql.Timestamp only data and method.

The feature can be enabled by setting the property jcclml.handlesTimestamps to any of the following values: enable, true, 1 (case-insensitive). Any value defined other than these to enable the feature will disable it. Note, however, that the property "jcclml.handlesTimestamps" *is case-sensitive*, so that the value for the logical name must be enclosed in double quotes.

```
$ define JCC_LML_JAVA_COMMAND_LINE "-Djcclml.handlesTimestamps=enable"
```


If using an Oracle JDBC driver and the user decides to enable the use of native JDBC date/time data types and methods, the following message will be written to the log files, but processing will attempt to continue:

```
***
*** Override of jcclml.handlesTimestamps (enabled) conflicts with expected
setting.  Continuing...
***
```

JCC highly recommends that you contact support before you try to override the Loader defaults.

OpenVMS Java Changes Across Versions

The JCC LogMiner Loader supports Java versions 1.4.2, 1.5.0, and 6.0 on OpenVMS AXP and IPF. Each of these versions implements the Java Virtual Machine in slightly different ways, some differences are innocuous and others impact how the Loader uses the JNI interface. One such change introduced in Java 6.0 fixed a bug in the JNI interface that required the Loader to specify the user and JDBC driver classes in the bootstrap class path (command line option -Xbootclasspath) as well as the user class path (command line options -cp or -classpath or -Djava.class.path). While many JDBC drivers functioned properly with the bootstrap class path including the user classes, all did not. Those that did not would throw an entirely ambiguous exception like:

```
java.lang.NoClassDefFoundError: Could not initialize class <class name>
```

Beginning with version 3.5, the Loader tests the Java version by inspecting the logical name JCC_LML_JDBC_VERSION which is set by the JCC_LML_JDBC_USER command. If the Java version is 6.0, the Loader does not specify the bootstrap class path when starting the Java Virtual Machine through JNI. This change allows all drivers tested with Java version 6.0 to behave as expected.

In the event that a user is testing a JDBC driver that is not certified with the JCC LogMiner Loader, there may be a possibility that the new defaults are not compatible with that driver. To override the defaults based on the Java version, use the logical name JCC_LML_JAVA_BOOTCLASSPATH.

> Note: The *java.lang.NoClassDefFoundError* exception can be thrown in a number of circumstances, including incorrect file attributes settings on the JDBC driver JAR files, so receiving this exception does not immediately imply that you have encountered this problem. JCC recommends that you contact support for assistance.

Using the logical name JCC_LML_JAVA_BOOTCLASSPATH, the user can direct the Loader to either use or not use the bootstrap class path when starting the Java Virtual Machine through JNI. When the value of the logical is set to "include" (case-insensitive), the Loader will include the user classes in the bootstrap class path. When the value of the logical is set to "exclude" (case-insensitive), the Loader will not specify the bootstrap class path. For example, to include the bootstrap class path:

```
$ define JCC_LML_JAVA_BOOTCLASSPATH include
```

If the logical name is set to any value other than those listed, the bootstrap class path will be included and the following message will be written to the log file:

```
bootclasspath unchanged by invalid logical name value.
```

JCC highly recommends that you contact support before you try to override the Loader defaults.

Ability to Specify JDBC Target Schema for Metadata Queries

It has been reported that some JDBC target data stores utilize database metadata queries differently. Rdb, Oracle and SQL Server, return data from metadata queries based on the database or username specified as the Loader target. Other data stores, such as Teradata and DB2 operate differently and require the schema name in the target in order to return the desired information.

The logical name is JCC_LML_JDBC_TARGET_SCHEMA enables specificatiton of a schema name to use in metadata queries.

e.g.

```
$ define JCC_LML_JDBC_TARGET_SCHEMA "MYSCHEMA"
```


If your target schema to which you are writing happens to match the username that you provided for the target validation credentials, you can specify the logical name as "#JCCLML$TARGET_USERNAME#" and the specified username will be used for the schema name in the metadata query.

e.g.

```
$ define JCC_LML_JDBC_TARGET_SCHEMA "#JCCLML$TARGET_USERNAME#"
```

Please note: There may be case sensitivity in the interpretation of `#JCCLML$TARGET_USERNAME#.`

Additional Topics for Specific End Targets

Note that the comments on a specific end target may have some applicability to others. The first topic is a likely case in point.

Case Sensitivity and Performance - SQL Server

When case sensitivity can be turned on or off, how it is set can degrade performance.

For SQL Server, the default collation is Latin1_General_CI_AS. That is, for comparisons and sorts, uppercase characters are the same as lower case characters. For character columns, case insensitivity adds overhead on queries that reference the column in a where clause.

Adapting the table definition to include COLLATE Latin1_General_CS_AS, at least for columns being indexed, will differentiate between uppercase and lowercase characters and is likely to improve performance for indexes and queries. This causes the SQL Server query behavior to match the Rdb SQL query behavior. The point, though, is that it saves processing and can be expected to improve performance.

Specifically, to define the table Employees with the primary key of employee_id, you might use:

```
CREATE TABLE EMPLOYEES
    (EMPLOYEE_ID char (5) COLLATE Latin1_General_CS_AS NOT NULL
    ,LAST_NAME   char(14) COLLATE Latin1_General_CA_AS NULL
    ,...
    ,CONSTRAINT EMP_LOAD_PRIMARY_EMPLOYEE_ID
      PRIMARY KEY (EMPLOYEE_ID)
    );
```

JDBC Batches - Oracle End Targets

Prior to Oracle Version 12.1 and Loader Version 3.5, the Loader disables use of JDBC batches for Oracle end targets that use the JDBC Loader target. With Oracle 12.1 Oracle-style statement batching is disabled in favor of standard JDBC batching.

Beginning with version 3.5 of the Loader, standard JDBC batching is available to Oracle driver version 12.1.0.1.0 or later, writing to an Oracle 12c target. Having the standard batching available is expected to offer performance options.

For earlier versions of the Oracle driver, the Loader continues to disable batching.

Note that the enhancement to use JDBC batches also is unrelated to Oracle targets that rely on OCI to write directly to the Oracle target.

Further Notes on Companion Products

The JCC LogMiner Loader is middle-ware. As such, the Loader works with a wide range of “companion” or “third-party” products. The behavior or misbehavior of companion products can, of course, change the functioning of the JCC LogMiner Loader.

Many of these issues are still outstanding as of the publishing date this document.

Java 2 (a.k.a. Java 1.5) on Alpha OpenVMS

The JCC LogMiner Loader depends on the OpenVMS implementation of Java to provide the ability to write to JDBC targets. On OpenVMS V7.3-2, the highest supported version of the Java VM is 1.5.0.

Using the FAST VM in version 1.5.0 results in an exception raised in the Loader Java method "lmlPrepareTable2", without the VM being able to output any details about the exception. In this situation, the Loader attempts to repeat the action iteratively until the configured number of output exceptions is reached and then fails with:

%DBA-E-MAX_OUT_RETRIES, Maximum message output failures received.

As this is an issue with the OpenVMS Java VM, *the only solution is to use the CLASSIC VM.*

Java 6 (a.k.a. Java 1.6) on Integrity OpenVMS

There have been various issues reported with different Java 1.6 kits on Integrity OpenVMS 8.4. This note attempts to detail which kits are currently supported by the JCC LogMiner Loader. Many of these kits are not referenced on the HP website, but are available from their FTP server in the folder ftp://ftp.hp.com/pub/gsy/digital/.

Patch level 1.6.0-1 is certified to work with the JCC LogMiner Loader. Patch level 1.6.0-5 with patch QXCM1001322821 is certified to work with the Loader and is the recommended version.

Patch levels 1.6.0-2, 1.6.0-2p1 and 1.6.0-3p1 have not been tested.

Patch level 1.6.0-3 fails with:

```
%SYSTEM-F-FLTDIV_F, arithmetic fault, floating divide by
zero at PC=00000000010DC751, PS=0000001B
```

Patch level 1.6.0-4 fails with:

```
java.sql.SQLException: I/O Error: invalid buffer length
(errno:65535)
```

Note that, although patch level 1.6.0-5 is the recommended patch, it is only recommended when installed with patch QXCM1001322821. Without that patch, the Loader fails after about 5 minutes of runtime with a floating point exception in the Java garbage collector. The exception is similar to:

```
[GC
```

```
%SYSTEM-F-FLTINV_F, floating invalid fault, PC=00000000018655E1, PS=0000001B
%TRACE-F-TRACEBACK, symbolic stack dump follows
image     module    routine              line      rel PC           abs PC
JAVA$HOTSPOT_SHR  GCADAPTIVEPOLICYCOUNTERS  update_counters_from_policy
                                    187824 0000000000000A61
00000000018655E1
JAVA$HOTSPOT_SHR  PSGCADAPTIVEPOLICYCOUNTERS  update_counters_from_policy
                                    189339 0000000000002C42
0000000002686C42
o
o
o
```

In summary:

TABLE 4. Java 6 (Java 1.6) Patch Levels

Patch Level	Comment
1.6.0-7	Recommended
1.6.0-6	Certified to work
1.6.0-5 with patch QXCM1001322821	Certified to work
1.6.0-1	Certified to work
1.6.0-2	Untested
1.6.0-2p1	Untested
1.6.0-3p1	Untested
1.6.0-3	Fails
1.6.0-4	Fails
1.6.0-5 without patch QXCM1001322821	Fails

Oracle Rdb JDBC Driver Incorrect Support of Batches

The difficulty described here is reported as Oracle Rdb Bug # 18816791.

When the ultimate target is Rdb and JDBC is the declared target of the JCC Log-Miner Loader, the Loader depends on the Rdb JDBC driver to write to Rdb via JDBC. Recently, it was discovered that the Rdb JDBC driver is no longer correctly supporting the use of JDBC batches in versions 7.3.2.x.x - 7.3.3.0.2 (a.k.a. v0703-3V0E5R). Unfortunately, the Rdb JDBC driver, with these versions, does not produce any exceptions and continues to process data, while the results returned to the Loader do not represent the work performed in the Rdb database. This Oracle Rdb behavior prevents detection of the issue by the Loader.

The user has two options for successful operation: upgrade or utilize a workaround. The best resolution is to upgrade to the Rdb JDBC driver version 7.3.3.0.3 or higher.

As a work-around to this issue, the user can disable the Loader's use of batches. The logical name JCC_LML_JDBC_BATCH_DISABLE defined as "1" will do so.

```
$ define jcc_lml_jdbc_batch_disable "1"
```

Work-around for Oracle Rdb JDBC Bug in Versions 7.3.3.0.0 - 7.3.3.0.3

The difficulty reported here is reported as Oracle Rdb bug # 18921457.

Versions 7.3.3.0.0 - 7.3.3.0.3 have a difficulty that leads to incorrect timestamps written to the target database, when using the Rdb JDBC thin driver. The problem involves an extraneous division by 10000 after converting to Java-internal time-stamp format when using the JDBC setObject methods to set data values on JDBC statements. The extraneous division has the affect of reducing the offset of the Java-internal timestamp epoch (1970-01-01) from 30+ years to 30+ hours. For example a date of 2014-05-21 is translated to 1970-01-02.

As a work-around to this Oracle Rdb bug, the Loader, beginning with Version 3.4.4, includes a mechanism that utilizes native datatype set methods for these versions of the Oracle Rdb JDBC driver.

Next Steps

Your target is now prepared and you can focus on the Control File to tell the Loader what you want to do. The chapters on other targets may be of no interest.

CHAPTER 1

Tuxedo Targets

When Tuxedo is the Loader target[1], changes in the source database are bundled into Tuxedo-defined FML32 buffers and sent a buffer at a time to the Tuxedo application. The choice of Tuxedo as the target is specified in the Control File with the keyword "Output."

If you are not using Tuxedo, you can skip this chapter without negative impact on your understanding of the Loader.

If you are new to using the Loader and are also new to Tuxedo, you may want to, first, test the Loader features and learn the concepts while using Rdb as a target. Because it is the default and because you will not have to learn the features of Tuxedo at the same time, learning on a sample Rdb target is likely to provide a better understanding.

1. Tuxedo targets are supported in several versions of the JCC LogMiner Loader product. If you are considering using a Tuxedo target, contact JCC LogMiner Loader support for additional information.

Introduction

The Loader interface to Tuxedo is either as a Tuxedo workstation client or as a member of the Tuxedo domain. To be a domain member, requires that the domain is running on the same OpenVMS server as the Loader. The choice is specified in the Loader Control File and in the tuxconfig (UBB) file.

The Loader may be configured to either enqueue FML32 buffers for later processing or perform direct TP calls to Tuxedo servers. The target queue space is specified via the "Output" keyword in the Loader Control File. JCC has found that direct TP calls offer significantly improved throughput.

The Loader groups all rows for a commit interval into a series of FML32 buffers, each FML32 buffer contains the rows for a single table. If the same sort order is used, rows are added to these FML32 buffers in the same sequence as they would be transmitted to an Rdb or Oracle target.

If sorting is specified in the Control File, then the rows are assembled in the specified sort order. JCC recommends using BY_RECORD sort with Tuxedo. If any other sort option is selected, the Loader will send more and smaller buffers.

The Loader begins a Tuxedo transaction. Upon successful transmission of all FML32 buffers for the commit interval, the Loader commits the transaction and then checkpoints its context to a local checkpoint file.

The Loader can be configured to cause the Tuxedo server processes to participate in the Loader's transaction. See "Tuxedo Application" on page 188.

The Loader is also configurable in how it handles transmission exceptions. See "Exception Handling" on page 186. See also "Keyword: Operator" on page 265.

The target TP service for an FML32 buffer is determined from the table name in the source database.[1] If the Loader control metadata file renames the table, the Loader will send those FML32 buffers to the renamed target. It is thus possible to configure applications with application servers which receive a single table and other application servers which receive several tables.

1. If the Tuxedo target is configured with "headers," then each FML32 buffer will include a field indicating the source table name for the buffer.

The Loader supports Tuxedo's asynchronous call interface. Specification of asynchronous and the impact of doing so are discussed in "Asynchronous Calls" on page 189.

Requirements for a Tuxedo Target

Configuring the Loader to send data to Tuxedo is straightforward and similar to configuring the Loader to send data to any other target. However, some special Tuxedo-specific information is required:

1. The Tuxedo target name and, potentially, credentials to log into the application
2. The field definitions to be used in the FML32 buffers
3. The target Tuxedo domain name or workstation target address
4. A Tuxedo application configured to accept the FML32 buffers

See also the Tuxedo specific keywords for the Control File, beginning with "Keyword: Tuxedo" on page 284.

Creating the Field Definitions

Columns in the Rdb database are reflected as "fields" in FML buffers. Tuxedo describes these fields through an internal numbering scheme which must be agreed upon by the "sender" and the "receiver" of data. The standard way of defining these fields is through something called FML32 field names.

These are defined to Tuxedo applications by C header files. The Loader reads these header files and converts database columns to the appropriate field number. How to create these is described in "FML32 Buffer Field Names and Required Header Files" on page 184.

The procedure supplied with the Loader generates a Tuxedo field header file. It is designed to use the fewest possible number of unique fields to represent all columns in a database.

Unique mappings can be a problem if there are two columns within a single table which map to the same name, as sometimes happens with long column names (say twenty-eight or more characters). In that case, the only option is hand editing of the

output file to create two different names for the two different columns (and null indicators.) The file must maintain the same format, so the comment for the new column which contains the mapping must be moved to just after the new field (and indicator.)

Syntax

The full syntax for the command procedure is

```
jcc_create_log_miner_tux_field_def                    -
<database name>                                       -
[fld def base]                                        -
[null indicators]
```

Defaults are

```
fld def base = 300
null indicators = 'N'
```

FML32 Buffer Contents

An FML32 buffer contains two kinds of data, header information and repeating groups of rows.

All fields in the FML32 buffer are represented as ASCII strings. Trailing spaces are removed from character strings and leading zeros are removed from numerics. Decimal points, if any, are included in numeric fields. Floating point fields receive the standard "E" notation to designate mantissa and exponent.[1] The format of date-time fields is that specified with the usual FAO arguments to the date_format keyword in the Loader Control File. [2]

1. The F float minimum is 0.293873558e-38 and maximum is 1.7014117e38. The G float minimum is 0.556268464628004e-308 and maximum is 0.898846567431157854 07e308.
2. See "Keyword: Date_format" on page 229. The default is date_format~|!Y4!MN0!D0!H04!M0!S0!C5|.

The number of rows in an FML32 buffer is determined by the target buffer size specified in the Control File together with the amount of data sent per row. This value is adjusted dynamically by the Loader in an attempt to achieve the specified size. FML32 buffers for different tables may, thus, contain differing numbers of rows.

In addition to the rows, each buffer may contain a pseudo-header set of fields which occur only one time. These header fields document generic characteristics of the buffer such as the source database table for the rows.

Header

If the Control File specifies a header (as part of the Output keyword), each buffer will contain one occurrence of the header (field occurrence 0). The fields of the header specify generic characteristics of the buffer such as the source database table for the rows.

TABLE 1. FML32 Header Fields

Name	Description
LOADERNAME	The name of the Loader in the Control File or as specified by the logical name JCC_LogMiner_Loader_Name
TABLENAME	The name of the table in the source database for the rows in this FML32 buffer.
JCCLML_ROW_COUNT	The number of rows in the FML32 buffer
TRANSMISSION_DATE_TIME	Timestamp from the system just prior to the transmission of the FML32 buffer.
JCCLML_RESEND_COUNT	The number of times that the buffer has been sent. See “Exception Handling” on page 186.

Loader Representation of NULL Values

If NULL values are encountered, the Loader includes a column for the field with a value of the empty string, regardless of data type.

Two mechanisms are provided to support the handling of NULL values from the source database. The first uses null indicator variables and the second provides a comma separated list of the names of columns containing NULL values. The spe-

cific type of NULL column support is selected through the Control File keyword "tuxedo~NullValues". [1]

It is, alternately, possible to set a value for any nulls in a specific column. For more information on this support, see "Keyword: MapColumn" on page 261.

Null indicator column. If the Control File includes a "tuxedo~nullvalues" definition and NULL values are encountered, then appropriate null indicator fields are materialized and set to "Y" in the FML32 buffer. The actual data field is set to a null string.

If no instance of a column in the source database is NULL, there will be no materialized null indicator columns. If the i^{th} instance of a column is null all entries with an index value less than *i* will have a null indicator column materialized for them and this column will be initialized to the null string. This variability does not represent a programming problem since the Fvals32 function to read fields from a buffer will return a null string if the field is absent from the buffer.

Comma-Separated List of Values. The comma-separated list of columns containing NULL values is appended, as an additional column, to each row and is named "NULLCOLUMNLIST".

FML32 Buffer Field Names and Required Header Files

Tuxedo deals with fields within FML32 buffers by assigning specific numeric values to them which are specified symbolically in a C header file. The JCC LogMiner Loader kit includes a procedure to assist the Administrator in the generation of these header files. To execute this procedure completely, your database should contain the JCC-supplied function GET_SYMBOL which is provided in the SQL procedure JCC_TOOL_SQL:VMS_FUNCTIONS.SQL.

This procedure generates a "field table" file with the name <database name>.TBL. This file is the "field table" file required by Tuxedo utilities to generate the ultimate header files. Syntax is:

```
JCC_CREATE_LOG_MINER_TUX_FIELD_DEF                -
          <database name>                         -
```

1. See "Keyword: Tuxedo~NullValue" on page 295.

```
[fld def base]                          -
[Null indicators?]                      -
```

Database name. Specifies the full path to the source database

Fld def base. Represents the base value selected for the Loader FML32 fields within Tuxedo for this application. The range of values assigned by this process must not intersect the range specified for other applications. The default is 300.

If the GET_SYMBOL function is not present, the default base value of 300 will be generated and you may edit the resulting.TBL file later and substitute the value for the application.

Null indicators. May be set to 'Y' or 'N' and indicates whether null indicator fields are to be generated. The default is 'N'.

The names generated in the field definition table are formed by taking the column name and appending a suffix. The suffix appended will depend on the datatype of the column. In the following chart, note the optional "s" that is appended for scaled columns.

TABLE 2. Generated FML32 Field Names

Suffix	Datatype
_t	Timestamp, transmitted as a text string in the format specified by the Loader keyword "date_format"
_s	String
_q[s]	Quadword (8-byte integer) [scaled]
_l[s]	Longword (4-byte integer) [scaled]
_w[s]	Word (2-byte interger)[scaled]
_b[s]	Byte (tinyint) (one-byte integer) [scaled]
_f	Floating point number
_d	Double precision number
_u	Unknown datatype
_ni	Null indicator

Once the field table file has been generated, it must be translated into a C header file. This is done with the Tuxedo command “mkfldhdr32”. The purpose of this command is to translate the field table file into a C header file. This command requires one argument, the name of the field table file. See the Tuxedo documentation for more information.

In addition to the database fields, the Loader may also materialize additional fields. In fact, for Tuxedo, the Loader always materializes a few fields within the header of the FML32 buffers. It may also be directed to materialize other fields. For this reason, a field table file containing definitions for all potential materialized columns is included with the kit.

The name of this file is jcc_tool_source:loader_virtual_columns.tbl. You should make a copy of this file and edit the copy to have the correct base for the Loader columns. After setting the base properly, you should then run the mkfldhdr32 to generate the correct Tuxedo header file.

These two header files — one describing the source database and one describing the materialized columns that the Loader can use — are required by the Loader at run-time and by the Tuxedo application at compile time. Additional header files are possible. You may use as many as is convenient.

The Loader reads these files and uses the appropriate field designators when forming FML32 buffers. It is typical of Loader Control Files that they would include references to at least two header files. If one or both of these header files is not specified in the Control File, the Loader will generate a run-time exception and exit.

It is very clear, therefore, that these two files must be managed very carefully. If the Loader version and the application copy differ at all, erroneous results will be generated by the application.

Exception Handling

Exception thresholds are defined by the parameters established in the keyword output_failure. Its parameters are timeout seconds and message retry attempts. The first parameter is only applicable to the API target. The second defines how many failures to tolerate prior to determining that conditions are not reconcilable and a failure should be reported.

If the reply from a service call results in a 'NACK', the Loader will immediately attempt to resend that buffer. The Loader will attempt to resend a specific buffer the number of retry attempts specified by the output_failure keyword. Both TPESVC-FAIL and application specified failures are treated as 'NACK'.[1]

If the Loader attempts to send a buffer a number of times equal to the specified retry count, the Loader will rollback the transaction, disconnect from Tuxedo, connect to the next available WSNADDR, and resend all buffers in the commit interval.

Application Load Balancing

Tuxedo workstation clients send their work to a target system whose TCP/IP socket (addresses + port) is specified by the logical name WSNADDR. If the Tuxedo application servicing the Loader is distributed on several systems, you may achieve a degree of load balancing across those systems by configuring the Loader Control File to distribute Loader workstation clients across this set of machines.

The method is to include the appropriate WSNADDR definitions in the Control File instead of defining a logical name as specified in the Tuxedo documentation. The Loader threads, as they begin, will cycle through the different WSNADDR values so specified. The Loader achieves this cycling by defining the WSNADDR logical name in process context before calling the Tuxedo images.

The Loader uses each address specified for each Tuxedo attach in round robin fashion. Thus if one Tuxedo server fails the Loader will, in response to the failure use an alternative server. This mechanism provides a modest amount of load balancing and fault tolerance for the Loader.

See also "Keyword: Tuxedo~Transaction" on page 288.

1. The JCCLML_resend_count for each buffer in a checkpoint is initially zero. If an individual buffer receives a failure, then that buffer is resent with its JCCLML_resend_count incremented. If sufficient failures occur such that the entire checkpoint must be resent, then each buffer in the checkpoint will be sent with the JCCLML_resend_count incremented by one.

Tuxedo Application

The Loader can be configured to either send FML32 buffers to Tuxedo queues or to directly call TP servers. A commit interval spanned by the Loader is encapsulated into a single Tuxedo transaction.

The Tuxedo application must provide services or queues with the names of the target tables. The application may accomplish this with one service which interrogates the table_name field in the FML32 buffer header or several specialized services one per table which do not need to process the embedded table name. Note that, if target table renaming is utilized, the names of these services must match the *renamed* names of the tables.

By default, the Loader uses the TPNOTRAN flag during the execution of the "tpcall" call. What this means is that any Tuxedo transactions that are begun by servers do not participate in the Loader's own transaction.[1]

It is possible due both to the handling of exceptions[2] and due to the handling of checkpointing[3] that a single buffer would be sent multiple times. The Tuxedo application should be constructed to be aware of this. There are two aids to the programmer in this circumstance. First, if the resend is due to exception processing, the field JCCLML_RESEND_COUNT incremented. Second, the materialized column Loader Sequence Number will always indicate the relative age of two changes. The Tuxedo application may use this value in determining whether to process any given record.

The design of the Tuxedo application should also recognize the asynchronous nature of a good deal of the processing. While the Loader session itself may also be configured to run constrained or unconstrained, that only refers to the order in which calls to Tuxedo are made. It does not define the order in which they are actually executed within the Tuxedo application. Again, the field Loader Sequence Number can be a great assist in determining the relative age of two records.

1. The default behavior produces a higher throughput in some systems, but requires special programming in the servers. It is possible to specify that the Loader will participate in the Tuxedo transaction. See "Tuxedo Call Transaction Support" on page 189.
2. See "Exception Handling" on page 186.
3. See "Checkpointing with Tuxedo Targets" on page 191.

Tuxedo Call Transaction Support

The Loader's Control File includes a means of specifying that the Loader is to control the distributed XA transaction with the Tuxedo servers. With Tuxedo~transaction set, the Loader starts and commits the transaction and the Tuxedo servers join the transaction.

This option is only available for Tuxedo targets and only if the call interface is used. Syntax is

```
Tuxedo~transaction
```

If this syntax is used without the Tuxedo call interface, it will have no effect. If this syntax is not used, the Loader will follow the default behavior.

By default the TPNOTRAN flag is passed to the Tuxedo tpcall routine. The TPNOTRAN disassociates the Tuxedo service that the Loader calls from the Tuxedo transaction that the Loader starts.[1]

The default behavior produces a higher throughput in some systems, but requires special programming in the servers.

Asynchronous Calls

The Loader can use the Tuxedo asynchronous call interface (tpacall/tpgetrply). Use of this interface enables the Loader to call servers with FML32 buffers and not wait for the server to respond. The Loader can, then, continue formatting additional FML32 buffers for subsequent service calls while previous calls are processing.

The Loader tests for service replies under any of the following conditions.

- At checkpoint intervals, the Loader will wait until all replies have been received.

1. The default behavior is to use the TPNOTRAN flag. The default excludes the Loader from the global XA transaction that includes the called services. Using Tuxedo~transaction disables that flag with the result that the Loader controls the global XA transaction to the target such that the transaction includes the entire commit interval.

- If the Loader receives the TPELIMIT exception from Tuxedo, the Loader will test for replies.
- For asynch Tuxedo calls, the Loader will not send any packet for a table with a higher TableOrder until all previous (lower TableOrder) packets have been acknowledged.[1]

Syntax and Syntax Errors. To enable the Tuxedo asynchronous call interface, requires that Tuxedo is declared as the target and that the call interface is used. The Loader will abort with an error if either Tuxedo is not declared as the target or the Tuxedo queue interface is selected. The following segment of a Control File illustrates a correct definition.

```
Output~Tuxedo~asynch~...
o
o
o
Tuxedo~call
o
o
o
```

Tuxedo Limits and Loader Retry Delay

The goal of introducing asynchronous call support (version 2.2) is to send as much data as possible with the least delay. Version 2.2.3 introduced additional features to avoid a limit in Tuxedo.

Tuxedo has a hard-coded limit of synchronous calls of fifty. To avoid application issues, the Loader limits the number of asynchronous buffers outstanding to 49. When the Loader detects that the limit has been reached, it attempts to retrieve replies for any outstanding calls. If no replies are available, behavior is controlled according to the setting of certain logical names.

The logical name JCC_LOGMINER_LOADER_ASYNC_RETRY_DELAY sets the number of seconds to wait after an attempt to retrieve replies for outstanding asynchronous calls for which no replies are received. Valid values are between 0.0 and 100000.0, inclusive. The default is zero which disables that wait.

If the Loader waits for the number of seconds defined by the logical name JCC_LOGMINER_LOADER_ASYNCH_RETRY_DELAY and retries to retrieve replies from outstanding asynchronous buffers, the number of times it will retry is

1. For a discussion of the keyword TableOrder, see the Control File chapter.

controlled by the keyword OUTPUT_FAILURE. If the number of retries (without success) reaches the number defined by OUTPUT_FAILURE, the Loader will abort the open transaction, disconnect from the Tuxedo application, reconnect and attempt to send all buffers in the transaction again.

Messages in the Log

If the Loader detects the maximum asynchronous messages limit, a message of the following format is written:

```
<timestamp>: Stalled waiting for available call descriptors(max=49)...
```

If the Loader attempts to retrieve asynchronous buffers and no replies are available, a message of the following format is written:

```
<timestamp>: Using all available call descriptors(49); waiting for
<asynch retry seconds> seconds before retry...
```

Once the Loader receives a reply for at least one asynchronous buffer that had been outstanding, a message of the following format is written:

```
<timestamp>: Continuing with available call descriptors(<current
outstanding buffers><49)...
```

Checkpointing with Tuxedo Targets

No highwater data is stored in Tuxedo. An LML_INTERNAL checkpoint[1] must be used. LML_INTERNAL checkpoints to a local file.

It is possible that the Loader will resend data during a restart. As with the API target, it is possible to send data and have an interruption before the Loader can update the checkpoint that the data was sent and received. In addition, a transaction from the source database may be decomposed in the Tuxedo application into several Tuxedo transactions. Some of the Tuxedo transactions may have already done their work when an interruption occurs before the source database transaction is fully processed. Since checkpointing is designed to ensure that a source database trans-

1. See “Keyword: Checkpoint” on page 222. For more on using the checkpoint in restarts, see “Rdb Issues” on page 412.

action is processed completely, the source database transaction will be repeated if the checkpoint file does not indicate that it was completed.

The Loader will re-play the transactions in Loader Sequence Number order. That will be the same as the source database transaction commit sequence. JCC recommends that the column LOADER_SEQUENCE_NUMBER be materialized in Tuxedo applications so that the target Tuxedo application can be aware of the relative age of the rows. (See “Keyword: VirtualColumn” on page 291 for a discussion of how to materialize this column.)

Authorization Model

The authorization model that you choose for your Tuxedo application causes the Loader to supply information to the tpchkauth call. Create the authorization model with the following:

TABLE 3. Loader Information for the Tuxedo Authorization Model

Tuxedo Authorization Mode	Parameter	Value
NOAUTH	init.usrname	"jcc_logminer_loader"
SYSAUTH	init.usrname	"jcc_logminer_loader"
	init.passwd	<LoaderName>
APPAUTH	init.cltname	"jcc_logminer_loader"
	init.passwd	<LoaderName>
	init.usrname	<validation username>
	init.data	<validation password>

Log Messages

The Loader writes to the Tuxedo ULOG.

ULOG Messages and Multiple Loader Families

Originally, when a Loader thread would write to the ULOG, it used the text "JCCLML" to identify the message. On systems with many Loader families, it could be cumbersome to relate a particular ULOG message back to a specific Loader thread.

Beginning with release 2.2.3, messages from Loader threads include both the process name and the process identification number (PID). In the example, the process names are shown in blue and the PIDs in red.

```
114902.ATLAS.JCC.COM!?proc.566363348: JCCLML[||1 REGTESTTUX:21C204D4]:
%dba_get_asynch_reply: PEOPLE(7): TPESVCFAIL - application level
service failure

114915.ATLAS.JCC.COM!?proc.566385938: JCCLML[||2 REGTESTTUX:21C25D12]:
%dba_get_asynch_reply: PEOPLE(3): TPESVCFAIL - application level
service failure
```

Messages for Exceeding Tuxedo's Asynchronous Messages Limit

See "Messages in the Log" on page 191.

Tips for the Administrator

This section is a collection of tips and information that may improve the results of anyone managing a system that includes a Tuxedo target for the Loader.

Location of the .TBL Files

In order to format the FML32 packets into text, Tuxedo requires access to the .TBL files associated with the field header file(s) specified in the Tuxedo~FieldHeader specification. These files are generated as part of preparing the Tuxedo application.

Two logical names are used in this specification. FIELDTBL32 specifies the names of the TBL files in a comma separated list. FIELDTBLDIR32 specifies a (comma separated) search list of directories, identifying where to find the files.

See also the comment on logging~output~dynamic data in the chart in "Logging and Verbosity" on page 247.

Determining Slowness in the Tuxedo Interface

By user request, a method to determine slowness in the Tuxedo interface is included in the Loader. With this method any call to tpcall, tpacall, or tpenqueue that takes more than a threshold number of seconds will cause an addition to the log. The threshold is controlled with a logical name. The logical name is translated each time the Loader [re]connects to the Tuxedo application. The default is 3600 seconds (one hour).

The logical name first introduced for this threshold was specific to Tuxedo, JCC_LML_TUXEDO_LOG_THRESHOLD. However, the more generic logical name, JCC_LML_TARGET_LOG_THRESHOLD has since been introduced and can be used instead. The only difference is that the Tuxedo specific logical name is translated each time the Loader connects to Tuxedo. See also “Target Latency Reflected in the Log” on page 365.

The default is 3600 seconds (one hour). The default is used if the logical name is not defined or is defined as zero or a non-numeric. If the value specified has more than seven digits of precision, the value is truncated to seven digits. If the value specified has more than two, but less than seven digits of precision, all of the digits are used, but no more than two digits are displayed.

For example, the following logical name definition for the regression test produces the following output

```
$ define JCC_LML_TUXEDO_LOG_THRESHOLD 1.111


15-AUG-2008 19:20:55.54 2240B10C||0 REGTESTTUX
JCC_LML_TUXEDO_LOG_THRESHOLD: set to 1.111 (using 0 00:00:01.11)
o
o
```

Statistics Reporting and Tuxedo Targets

Reporting latency and attributing latency to specific stages can be an important analysis tool. Beginning with Version 3.1, the output latency has been broken down into finer components. Target (trgt) and Conversion (cnvt) latencies are of note for the added control available with Tuxedo targets and the JCC LogMiner Loader, Version 3.4.4 and beyond.

In general trgt latency is defined as “the latency attributed to the target data store.” Cnvt latency is defined as “the latency attributed to the Loader during the output phase as it converts the input data into the data format and style that the Control File specifies for the target.”

In early versions of the JCC LogMiner Loader when using the Loader statistics monitor (JCC_LML_STATISTICS) the target (Trgt) latency times recorded in the Detail and T4 output formats represented the elapsed time for each call into the Tuxedo API. Since the Tuxedo API calls include the calls to initialize the application and allocate and process FML32 buffers, the time reported included more than the calls to transmit the data to the target application.

A logical name was added with Version 3.4.4 of the JCC LogMiner Loader to enable the user to choose a finer distinction on the latency. The logical name is JCC_LML_TUXEDO_CONVERT_LATENCY and the value should be a comma separated list of the Tuxedo functions (see chart below.) The logical value should contain no characters other than the listed functions and a comma. There should be no white space in the logical value.

When a function is listed in the logical value, the Loader statistics monitor latency amounts attributable to the named Tuxedo functions are included in the convert (Cnvt) latency, instead of in the target (Trgt) latency. By adjusting this logical name, the user can get a better idea of which Tuxedo functions are most responsible for the target latency.

Valid values are:

TABLE 4. Controls for Statistics Output with Tuxedo Targets

tpalloc	create a message buffer
finit32	initialize fielded buffer
fprint32	print buffer to standard output
fchg32	change field occurrence value
fneeded32	compute size needed for buffer
fsizeof32	return size of fielded buffer
fused32	return number of bytes in fielded buffer
tpenqueue	enqueue a message to a message queue
tpcall	initiate a synchronous request/response to a service
tpacall	initiate an asynchronous request
tpbegin	begin a transaction
tpcommit	commit the current transaction
tpfree	free a message buffer
tpcancel	cancel an asynchronous request
tpabort	rollback the current transaction

TABLE 4. Controls for Statistics Output with Tuxedo Targets

tpterm	leave an application
tpgetrply	receive an asynchronous response

Note: Not all of these Tuxedo functions are used for all Loader configurations. For example, tpenqueue is not called if the Loader is configured to make synch server calls (tpcall).

This excerpt of the jcc_lml_statistics detail display highlights the Cnvt and Trgt latency details affected by this logical name:

```
- Latency(sec) ------ LML detail ------
CLM   12.1m |  Inpt   0.1%  Cnvt   0.2%
------------   Sort   0.0%  Trgt  99.7%
LML    8.83    Sync   0.0%  Ckpt   0.0%
```

This example will remove all FML (listed functions) work from trgt latency and report it as cnvt latency:

```
$ define jcc_lml_tuxedo_convert_latency -
      "finit32,fprint32,fchg32,fneeded32,fsizeof32,fused32"
```

This excerpt of the jcc_lml_statistics detail display highlights the Cnvt and Trgt latency details and shows the effect the above logical value might display (depending on the source and target environments):

```
- Latency(sec) ------ LML detail ------
CLM   12.1m |  Inpt   0.1%  Cnvt   0.4%
------------   Sort   0.0%  Trgt  99.5%
LML    8.83    Sync   0.0%  Ckpt   0.0%
```

ASYNCH Limit Configuration for Tuxedo Target

One reason for the Tuxedo target to be slow can be the Loader sending more data than the target servers can handle. When configured to perform ASYNCH Tuxedo calls, the Loader can fill and send a maximum of 49 buffers concurrently.

The logical name JCC_LML_TUXEDO_ASYNCH_LIMIT enables the user to reduce the maximum number of ASYNCH buffers that can be sent concurrently.

Reducing the maximum number of concurrent ASYNCH FML32 buffers may increase throughput on a resource constrained target.

```
$ define JCC_LML_TUXEDO_ASYNCH_LIMIT "42"
```

This logical name enables the user to limit the number of outstanding FML buffers in ASYNCH mode. Valid values are integers in the range of 1 – 49, inclusive. If the logical name value is outside that range or not a number, it is ignored and the value 49 is used.

End Target of the Tuxedo Application

The end target of the Tuxedo application may include additional processing. However, it is important to avoid repeating triggers which have already fired on the source database and care should be taken with constraints that have already been met in the source. Other possible considerations for the end target of the API will be similar to the end target concerns discussed in the JDBC chapter. The following may be of particular interest:

- “Preparing the End Target” on page 155
- “Populating the End Target” on page 156
- “Constraints and Triggers in the End Target” on page 157
- “Adding the High-Water Information” on page 158
- “Adding Dbkey Columns” on page 159

CHAPTER 2

XML for File or API Targets

The Loader can be directed to emit its results as XML[1] documents. Each document *may* be modified with the addition of a LogMiner Loader header. This header provides additional material to the API routines. The Loader may also be directed to generate documents without headers.

Using XML or JDBC as the Loader target greatly expands the range of ultimate targets available. Since XML uses your own API, it provides more options for the end target than any of the other choices.

If you are not using XML to write to a target, you can skip this chapter without negative impact on your understanding of the Loader.

The Loader can transmit XML to either a file or a customer-supplied API. The API target provides capability for publishing database changes to your own subscribers. The file target is generally used for testing, but has also been used for production purposes.

1. Some XML products use slightly different formatting.

Set Up

Establishing the XML target is relatively straightforward.[1] Some aspects are the same as those required for any other target. Some additional things to consider are included in this section.

Control File

You will need to define the Output Keyword with output type of 'API' or 'FILE' and output conversion of 'XML'. See "Keyword: Output" on page 275.

You will need to define the API Keyword to identify your three API routines. See "Keyword: API" on page 230.

You can use optional XML keywords[2] to provide additional control or to specify an alternate Document Type Definition (DTD).[3] The XML keywords are described in "Keyword: XML" on page 307.

Ultimate Target

Your API will publish the Loader messages to some ultimate target. Since the Loader messages represent *changes* to the source database, there is an assumption that the ultimate target begins with a copy or subset of the data that is available in the source at the initiation of using the LogMiner and the Loader. There is also an opportunity to add materialized data,[4] but that does not change the need to have a base from which to begin. One option for establishing the base data is the Data Pump. See "Data Pump" on page 505.

In populating your ultimate target initially, establishing a known state for the source is recommended. See "Quiet Points and AIJs" on page 43 for a discussion that relates to all targets.

1. Creation of the API is discussed in "API Routines" on page 210.
2. These must occur in your Control File after output conversion is set to XML.
3. JCC recommends using the JCC default DTD. See the example and pointers to files included in the kit at "XML DTD Definition" on page 209.
4. Materialized data options are discussed in "Keyword: VirtualColumn" on page 299.

Keys

As with any target, it is necessary to determine how each row is to be identified. See “Identifying Rows in the Target” on page 39 for a discussion of the importance of keys for identifying rows in the target and for a discussion of the occasional necessity, as well as the drawbacks of using the originating_dbkey approach.

Data Types

The data types of corresponding columns in the target must be compatible with data from the source database columns. For instance, it would be inappropriate to attempt to convert text columns in the source database to numeric columns in the target database unless you are guaranteed that no data conversion exceptions will be generated or unless you specifically use the MapResult keyword to transform the data.

Manipulation of data and data types is possible with the Loader. See “Schema and Data Transforms” on page 489 and others for a discussion of transforms. Your API may provide additional data conversions.

The output conversion parameter enables, among other things, specification that the trailing blanks should be trimmed from source data. If you use trim, a decision may be required as to how your API is to treat text of zero length. Text that was all blanks in the source, if TRIM is specified, will be trimmed to text of zero length. In some systems, this is not an issue; in others, it is. Neither Rdb nor the Loader regard text of zero length as synonymous with null. Therefore, the specification of value if null cannot be used to resolve which value to use.

Tuning

Using an XML target requires sufficient resources for processing multiple copies of the messages generated. See “Message” on page 211 for the limit on message size.

Loader Output

This section describes the XML message which will be provided to your API.

Message Header

The message header (when used) will contain information necessary to decode the rest of the message and to mark the high-water context for messages transmitted. The message header will contain the following ASCII fields:

Checksum of the remainder of message. 8 hexadecimal digits for a checksum. See the figure "Checksum Algorithm" on page 208 for how this is calculated.

Loader sequence number. For compaction purposes, the Loader sequence number (a Loader generated number) will be displayed in hexadecimal radix with 16 hexadecimal digits.

Message length. This will be the length of the message, including the header, in bytes. Its purpose is to provide further validity checking of the integrity of the message and to facilitate programming of message decomposition. It will be represented as 8 hexadecimal digits.

Message type. This is a constant string 'XACT'.

Send type. This is the value 'S' (Startup), 'N' (Normal) or 'R' (Resend). The Loader will generate two 'S'tartup messages on start or restart, then will only send 'N'ormal messages unless NACKed (in which case the Loader will 'R'esend the message.

Publishing database name / Loader name. Note that the database name included here is that specified in the JCC LogMiner Loader Control File or by the logical name JCC_LogMiner_Loader_Name. The length of that name in the Control File will affect the length of the header.

Message transmission date / time. This is the system time obtained just prior to calling the interface routine to transmit the message. The format of this timestamp is specified in the Control File. (See "Keyword: Date_format" on page 238.)

Compression algorithm. This will be a 4-byte ASCII string containing a sequence number indicating the compression algorithm used for the remainder of the message. The value "0000" will not be legal. A value of "0001" will indicate an uncompressed message. No other compression format has been defined, at this time.

Version number of Loader. Note that message format might vary across major and minor versions but will not vary across patch numbers. Version numbers will

be obtained by having the Loader read the process header and by linking with appropriate version definitions in a linker options file.

The format is MMNNPP, where:

MM is the major version

NN is the minor version

PP is the patch number

For example, 020006 is version 2.0 patch 6.

Loader link date and time. The format is the standard time format specified in the Control File for the keyword DATE_FORMAT. The format of this timestamp is specified in the Control File. (See "Keyword: Date_format" on page 229.) [1]

Capability flags. Capability flags (16 bytes, one flag per byte) are reserved for later use. Initially, these flags will all be set to text zeros (not binary).

Message Body

The message body will consist of an XML presentation of the data rows that have been changed during a transaction. The XML will consist of tags that are to be nested as described below. The message body should be considered to be a string containing no extra characters other than those required to transmit the message. In the descriptions below, the underlined characters in the title will be used as the tag and attribute names.

Packet (pkt). Each XML packet will contain both a beginning <pkt> and a trailing </pkt> tag.

Transaction (trxn). Each transaction included within a packet will start with a <trxn> tag and be terminated with a </trxn> tag. Transaction attributes are:

- **TSN** Each transaction within the database is assigned a 64-bit transaction sequence number. Although these sequence numbers may be

1. Early versions of the Loader, limited date formats in XML to nineteen characters. That is no longer the case. Also, the default date format is now |!Y4!MNO!DO!HO4!MO!SO!C5|. Use the date keyword in the Control File to establish a different format.

assigned serially in the source database, transactions are reported by the LogMiner in the order of commit. TSNs are roughly but not precisely sequential in nature. This attribute will be reported in hexadecimal radix with leading zeros stripped from it.

- Start Time (**strt**) Transaction start time, in standard time format, is specified in the Control File for the keyword DATE_FORMAT. (See "Keyword: Date_format" on page 238.)
- **End** Time Transaction end time, in standard time format, is specified in the Control File for the keyword DATE_FORMAT. (See "Keyword: Date_format" on page 238.)

Row. Each row updated during a transaction will be started with a <row> tag and terminated with a </row > tag. Several attributes will be associated with each <row…> tag. Row attributes are:

- **Name** The table name for each row will be designated through the Name attribute. Table names will be enclosed in quotes (') and be transmitted in capitalized ASCII text. The maximum length possible for an Rdb table name is 31 characters.
- Action (**actn**) The operation on the row will be designated with the Action attribute. The action value will be enclosed in quotes ('). Legal values for action are 'D' to signal a row deletion and 'M' to signal the fact that the row has been modified.

Column. Each column within a row will be described completely within a single tag <column…/>. The data value and other attributes will be signaled within that tag. Attributes are:

- **Name** The name attribute will be set as the name of the column being transmitted. This name will be in upper-case ASCII characters and will be enclosed in quotes ('). This name will be specified within the Control File for the Loader. Note that the order of column names for a particular row may vary from database instance to database instance. Rdb column names may be up to 31 characters.
- **Null** ***optional*** The fact that a column is null will be signaled through this attribute. Possible values are {'Y' | 'N'}. 'Y' indicates that the column is NULL and 'N' indicates that the column has a value. This is an optional parameter which, if omitted implies that the column is not NULL. Actually, there is never a case where the Loader will write null='N'. It writes either null='Y' or val='<value>'. You'll never see the

null and val tags in any given <col.../>. See “Data Types” on page 201 for a discussion of zero length strings and their distinction from null strings. See “<value if null> optional” on page 262 in the Control File chapter for additional comments on working with Nulls. See also “Keyword: MapResult” on page 266 for another approach.

- **Datatype** The data type of the item will be signaled by this attribute. Possible values are {‘string’|’number’|’date’}. Date items will be used for all source data that are recorded by Rdb as OpenVMS date data-type.
- **Value** The value of a column is transmitted by this attribute. The value will always be enclosed within quotes (‘). This attribute will not be present if the NULL attribute has been set to ‘Y’. See “Representation of Column Values” on page 205 for additional comments on how the column value is represented.
- **Length** The length attribute will be set only for character strings. This value will indicate the maximum size that the string could be. Its values will be delimited by quote marks (‘).

Representation of Column Values

If there is no fractional portion of the numeric value the decimal point will be omitted. If there is a decimal fraction, trailing zeros will be omitted. The Loader will also omit leading zeros.

Character strings have trailing spaces removed.

XML provides capability for including XML control characters in strings. These characters are escaped as follows:

TABLE 1. XML and “Special Characters”

Special Character	Escaped Sequence
<	<
>	>
&	&
"	"
'	'

Otherwise, all control characters except for TAB, CR and LF will be encoded with standard escape sequences. Escape sequences for these characters are of the form "&#XX;" where XX represents the hexadecimal representation of the character value as an integer. DEL (&#7F;) will be encoded as well.

Date items will be formatted in the standard ANSI date format. This format is specified in the Control File for the keyword DATE_FORMAT. (See "Keyword: Date_format" on page 238.) Note that OpenVMS does not formally guarantee time to greater than 0.01-second precision, although the date-time data type actually stores values that are integer counts of 100 nanosecond intervals. Clocks on the Alpha systems actually tick at 1/1024th of a second.[1] It should also be noted that it is a frequent practice to store unknown dates & times as the value "18581117000000000000" which translates to 17-Nov-1858 at midnight.

Sample Message

The following is a sample of a message that will be passed by the Loader. It assumes that there are two rows updated by the transaction and that each row has columns with the values as described below:

```
EMPLOYEES: ID =  1
           LAST_NAME = 'Jalbert                    ',
           FIRST_NAME = 'Jeffrey                   ',
           TERMINATION_DATE = <NULL>
ADDRESS:   ID = 1
           TYPE = 'BUS '
           STREET = '600 New Rd  "SE"              '
           CITY = 'Granville           '
           STATE = 'OH'
```

These changes would be represented by the text below. The first block of characters is the message header. The second is the XML of the message body.

1. The OpenVMS system generally "acquires" some specified number of hardware ticks before a software tick is generated -- the software clock ticks over every 10ms, while the hardware tick occurs at an architectural minimum rate of 1000 times per second. Various Alpha systems have a hardware tick rate of 1024 interrupts per second. As mentioned, the OpenVMS Alpha software clock rate is 10ms, meaning that a (large) number of hardware ticks transpire for each software tick.

This example is formatted for readability. In practice, the white space would be removed, except for that between quote marks (‘). Also for readability, the different header fields are represented, here, in different colors. Further, this is a synthesized example only. The value of the length field is not correct for the other data shown.

```
CHECKSUM00000000000000000000002E0XACTNSUBRDB0520010814124351
12345000101000020010822153176234590000000000000000
<?xml version=’1.0’?><!DOCTYPE order SYSTEM ‘packet-commented.dtd’>
<pkt>
 <trxn tsn=’5FCDE8’ strt=’2001081412435098265’
end=’20010814124349865 32’>
  <row name=‘EMPLOYEES’ actn=‘M’>
   <col name=‘ID’ type=‘num’ val=’1’/>
   <col name=’LAST_NAME’ type=’str’ val=’Jalbert’ len=’30’/>
   <col name=’FIRST_NAME’ type=’str’ val=’Jeffrey’ len=’30’/>
   <col name=’TERMINATION_DATE’ type=’dt’ null= ‘Y’/>
  </row>
  <row name=’ADDRESS’ actn=’M’>
   <col name=’ID’ type=’num’ val=’1’/>
   <col name=’TYPE’ type=’str’ val=’BUS’ len=’4’/>
   <col name=’STREET’ type=’str’ val=’600 New Rd &quot;SE&quot;’len=‘30’/>
   <col name=’CITY’ type=’str’ val=’Granville’ len=’30’/>
   <col name=’STATE’ type=’str’ val=’OH’ len=’2’/>
  </row>
 </trxn>
</pkt>
```

FIGURE 1. Sample XML Message

Checksum Algorithm

The checksum of the remainder of the message is the first element in the message header. This is used, in some environments, as a check on the accuracy of the transmission. So that you have the option of fully understanding the algorithm, it is shown in its entirety in the following.


```
/
*******************************************************************************
**
**  Copyright (c) 2000-2001, JCC Consulting, Inc.
**                           Granville, OH  USA
**  All Rights Reserved.
**
**  This software is provided on an as-is basis under license, and may be used and
**  copied only as provided under the license terms and with the inclusion of this
**  copyright notice.  This software contains material which is confidential and
**  proprietary to JCC Consulting, Inc., and may not be disclosed to any other
**  person.  No title to or ownership of the software is transferred.
**
**  The information in this software is subject to change without notice and should
**  not be construed as a commitment by JCC Consulting, Inc.
**
**
*******************************************************************************/
#include stdio
#include string

char *comc_checksum(input_data, input_size)
void *input_data;
int input_size;
{
int checksum_value, *int_ptr, left_over, left_over_length, *input_end;
static char return_checksum[9];

/*
** Create a checksum...
**  XOR each 4-bytes of the data
*/
    checksum_value = 0;
    input_end = (int *)((char *)input_data + input_size);
    for (int_ptr = (int *)input_data; int_ptr+1 <= input_end; int_ptr++)
        checksum_value = checksum_value ^ *int_ptr;
/*
**  Anything leftover is copied to an initialized 4-byte structure and XOR'd */
    if ((left_over_length = input_size % sizeof *int_ptr) != 0)
    {
        left_over = 0;
        memcpy (&left_over,((char *)input_end)-
left_over_length,left_over_length);
        checksum_value = checksum_value ^ left_over;
    }
/*
**  Convert to 8-byte hex and return
*/
    sprintf(return_checksum, "%8.8X", checksum_value);
    return (return_checksum);
}
```

FIGURE 2. Checksum Algorithm

The checksum algorithm will be used to generate the checksum for the message header (“Message Header” on page 202). Note that it returns 8 hexadecimal characters.

XML DTD Definition

The following XML DTD (Document Type Definition) provides an example XML format used in the messages:

```
<!--DTD for Messages Received from JCC Loader -->
<!ELEMENT pkt(trxn+)>
<!ELEMENT trxn(row+)>
<!ATTLIST trxn
    tsn CDATA #REQUIRED
    strt CDATA #REQUIRED
    end CDATA #REQUIRED
>
<!ELEMENT row(col+)>
<!ATTLIST row
    name CDATA #REQUIRED
    actn (D|M) #REQUIRED
>
<!ELEMENT col(#PCDATA)>
<!ATTLIST col
    name CDATA #REQUIRED
    null CDATA #IMPLIED
    type (str|num|dt) #REQUIRED
    val CDATA #IMPLIED
    len CDATA #IMPLIED
>
```

FIGURE 3. XML DTD Definition

The JCC LogMiner Loader kit, beginning with version 3.5, also includes two files which may be used as the DTD:

- packet-commented.dtd (transaction-based)
- packet-record-commented.dtd (record-based)

API Routines

The Loader will call a customer-supplied shareable image. It is assumed that the interface is written in "C" and calling protocols are therefore those used by that language. This interface consists of three routines — CONNECT, SEND, DISCONNECT. These are discussed further in the following.

To assist the API programmer, the kit includes a C language header file that describes the interface. This header file can be found in jcc_tool_api:jcc_lml_api.h.

The kit also includes the source code for the API routines used in the Loader regression test. These form useful examples. Hints on debugging techniques are also embedded in the examples.

Connect

The Connect routine is used once per Loader session to establish a communication path to the customer-supplied queues. All three parameters are text strings and their interpretation is left to the C routine. By convention, the last of these is case sensitive. These parameters are defined with the API keyword in the Control File.The Connect routine accepts the following parameters:

Handle. An address of a variable provided by the Loader in which the interface routine places sufficient context so it can uniquely identify this connection. This parameter is defined with the routine name parameter of the API keyword in the Control File. (See "Keyword: API" on page 230.)

Client Name. A null-terminated ASCII byte string containing the name of the publishing database. This parameter is established with the Output keyword in the Control File.

Timeout. An integer (signed integer of 32 bits) passed by reference containing the number of seconds after which the message transmission is assumed to have failed or timed out. This value is passed directly to the API without interpretation. This parameter is established with the Output keyword in the Control File.

Send

The Send routine is called once for each message being sent by the Loader. This routine requires the following parameters:

Handle. The address of the handle specified during the connect call. This parameter is defined with the routine name parameter of the API keyword in the Control File.

Message. This parameter is a pointer to a buffer containing the message, terminated by a binary zero byte. The maximum number of bytes supported is 2.1 GB.

Size. This is the address of an *un*signed 32-bit integer specifying the size of the buffer. Note that this size includes the message plus any padding added to the buffer to round it up to an 8-byte boundary.

Disconnect

This routine is used to terminate a session. It accepts one argument.

Handle. The address of the handle specified during the connect call. This parameter is defined with the routine name parameter of the API keyword in the Control File.

Return Codes

Each of these interface routines will be called as functions. Return statuses are defined in the header file supplied as an example.

The Loader interprets a failure from a Send as a NACK of the message. Failure returned from a Connect call will result in the Loader exiting. Failure from a Disconnect will result in an exception being reported as the Loader exits.

API Header File

The header file supplied with the kit defines the API interface. The header file is

```
jcc_tool_source:JCC_LML_API.H
```


Checkpointing with XML Targets

XML targets supply no database to store the highwater data. An LML_INTERNAL checkpoint[1] must be used. LML_INTERNAL checkpoints to a local file.

It is possible that the Loader will resend data during a restart. It is possible to send data and have an interruption before the Loader can update the checkpoint that the data was sent and received. Since checkpointing is designed to ensure that a source database transaction is processed completely, the source database transaction will be repeated if the checkpoint file does not indicate that it was completed.

The Loader will re-play the transactions in Loader Sequence Number order. That will be the same as the source database transaction commit sequence. JCC recommends that the column LOADER_SEQUENCE_NUMBER be materialized when using the XML interface so that the API can process based on the relative age of the rows. (See “Keyword: VirtualColumn” on page 299 for a discussion of how to materialize this column.)

Writing to a File

It is possible to use an OpenVMS file as a target for the Loader. This option was provided primarily to serve as a way of testing XML output.

At least one Loader installation has used a file target for more substantive purposes, prompting attention to better controls and improved performance.

File may be specified as the output type with the output keyword. See “Keyword: Output” on page 275.

Note that you can also send checkpoint information to a checkpoint file (LML_INTERNAL) to provide restart capability. By default, checkpoint file settings control the operation of both the checkpoint and output file.

If the output stream type is a FILE, then, by default, the file is closed and reopened every however many hours are specified by the commit interval. There are, how-

1. See “Keyword: Checkpoint” on page 231. For more on using the checkpoint in restarts, see “Rdb Issues” on page 412.

ever, options that provide more control of the file opening and closing. You can change the checkpoint interval, the units for the interval, and the nature of how the file is closed.

Change the Units for Checkpoint Intervals

This feature provides a logical name that can set the units to use in specifying checkpoint intervals for output to a file.

```
$ define JCC_LML_FILE_CHECKPOINT_UNIT <value>
```

The value options are seconds or minutes or hours or days. The default is hours. The unit specified by this logical name is applied to the <checkpoint_interval> as specified in the CHECKPOINT keyword. This logical name has no effect when the output target type is other than file.

Change the File Flush Interval

The Loader Administrator can also control how frequently the file is written so that it is available to read. This is also done with a logical name. The minimum value for this is 3.0 seconds. The default is 600 seconds (10 minutes).

```
$ define JCC_LML_FILE_CHECKPOINT_SECS <seconds>
```

Control How the File Closes

In addition to finer control for reopening the Loader output files, performance and other enhancements have been made to reduce the overhead associated with writing the output files. These changes are:

1. truncate the unused portion of the file extension of the output file on closing
2. defer output file writes to minimize I/O
3. eliminate an extraneous linefeed at the end of each XML document.

As a result, there is significant reduction in overhead when writing to a file target. Unfortunately, the reduction is not necessarily obvious on the statistics screens.

File Format Change

In order to improve performance in writing to FILE targets, the RMS organization of target files has been changed. As a result, programs that open those files may require editing. Alternatively, before such files are processed, they may be converted to the previous RMS organization through DCL similar to:

```
$ convert/fdl=version_34_attributes.fdl    'source_file   'converted_filename
```

And the FDL file contains something similar to:

```
IDENT   FDL_VERSION 02 " 3-MAR-2017 08:38:18   OpenVMS ANALYZE/RMS_FILE Utility"

SYSTEM
        SOURCE                  OpenVMS

FILE
        ALLOCATION              100
        BEST_TRY_CONTIGUOUS     no
        CLUSTER_SIZE            16
        CONTIGUOUS              no
        EXTENSION               0
        FILE_MONITORING         no
        ORGANIZATION            sequential
        PROTECTION              (system:RWED, owner:RWED, group:RE, world:)
        GLOBAL_BUFFER_COUNT     0
        GLBUFF_CNT_V83          0
        GLBUFF_FLAGS_V83        none

RECORD
        BLOCK_SPAN              yes
        CARRIAGE_CONTROL        carriage_return
        FORMAT                  stream_lf
        SIZE                    0
```

Reading the Files

A user-requested feature beginning with Version 3.2.4 of the Loader enables reading output files prior to closing them.

Recommendations for the End Target

Topics that may enhance your understanding of what is required in the end target of the API will be similar to the end target concerns discussed in the JDBC chapter. The following may be of particular interest:

CHAPTER 13

Control File

The primary means by which you specify choices for the Loader operations is through the Control File.

This chapter describes how you build the Control File to accomplish what you want. There are a great many options that you will want to understand, but each option is easy to work with. You will discover that there are command procedures in the kit that will help you get started. There are also default values for many of the options.

Building the Control File

The Control File specifies the detailed actions that the LogMiner Loader takes during a session. It specifies each source table, each target table and the mapping between them. The Control File specifies data to materialize, data to transform, what to log, and how to use parallel processing and other performance options.

The Control File is extremely detailed. In addition to everything else, it contains a complete physical metadata definition of each table to be maintained. It is important to note that this definition not only includes the definition of the columns in the

tables together with their data types, it also includes the actual order of those columns within the tables as well as the internal Rdb version number of the table.

It would be a tedious task to prepare a complete metadata description from scratch. Accordingly, the Loader kit includes automated procedures to prepare a complete definition that can be used as produced or edited to meet specific needs.

The default metadata Control File is designed to cause JCC LogMiner Loader to maintain the target database as a complete replica of the source database. You may want to customize this Control File to achieve other purposes.

Should a change in the database metadata be required, you must completely process every AIJ file based on "old" metadata. Then, you must customize the Control File to include the metadata changes for the tables that are being managed by the Loader. After that, you can begin running the Loader with new metadata.[1]

Control File in the Architecture

The Control File directs the Loader in how to handle the LogMiner output.

Logical names also provide some measure of control. See "Logical Names for Control" on page 32, "Logical Names" on page 585, and other discussions.

The following figure from the Basics chapter highlights the input and output.

1. See "Metadata Versions" on page 224.

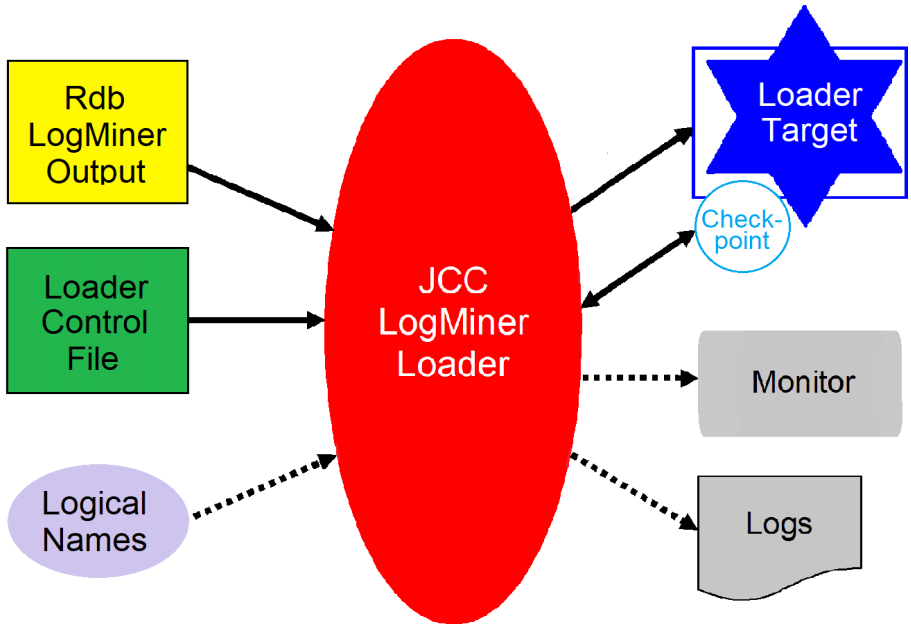


FIGURE 1. JCC LogMiner Loader Input and Output

Referencing Other Control Files

Large files often can obscure important issues. The Control File language implemented by the Loader will allow you to build a Control File by segregating important categories of information in separate files and then referencing those additional files in a main file. See “Keyword: Include_file” on page 246 and “Statement Ordering” on page 224.

JCC recommends the following general architecture for your Control Files

1. A Master Control File containing general information such as the Loader family name[1], the commit interval and such. This Control File will call the rest as a result of Include_file statements.

2. A target-specific Control File to specify the target data store with the output keyword.
3. A Metadata Control File (for the source database) that is generated by the automatic build tool described in "Building the Metadata Control File" on page 222.
4. A file specifying materialized (virtual) columns, if any. See "Keyword: VirtualColumn" on page 299.
5. An exclude file that contains specifications of source tables and columns to be excluded, if any. See "Keyword: Exclude" on page 239.
6. A filter Control File expressing any (source-based) filters that are needed, if any. See "Keyword: Filter" on page 240.
7. A maptable Control File for specifying target specific metadata and the mappings from source to target. See "Keyword: Map..." on page 260.
8. A file containing all FilterMaps, if any. See "Keyword: FilterMap" on page 242.
9. A file that includes target specific excludes, if any. See "Keyword: MapExclude" on page 264.
10. A file that includes MapResult statements, if any. See "Keyword: MapResult" on page 266

Note that you can split up the Control File even more if that suits your environment better. The limit of files supported for the Include_file keyword is 1024. This number is expected to be significantly more than is needed.

When the Loader encounters an include_file in processing, the name of the included file is echoed in the log file.

Example of a Control File Portion

This example contains two of the include files that are recommended.

```
! Control File for DEMO.
LOADERNAME~DEMO
LOGGING~INITIALIZATION
LOGGING~STATISTICS~RUNTIME,TIMER
o
```

1. See "Keyword: Loadername" on page 254.

```
o
o
Include_file~source_metadata.ini
o
o
o
include_file~target_mapping.ini
o
o
o
```

FIGURE 2. Sample Portion of a Loader Control File

The first of the included files would define the source tables.

```
! Define the source metadata and the mapping to the
! target.
Table~EMPLOYEES~1~~NoMapTable
Column~EMPLOYEES~EMPLOYEE_ID~1~5~0~14~0
Column~EMPLOYEES~LAST_NAME~2~14~0~14~0
o
```

FIGURE 3. Example Control File for Defining Source Tables

```
o
o
o
MapTable~EMPLOYEES~EMPLOYEES~Replicate
MapColumn~EMPLOYEES~EMPLOYEE_ID
MapColumn~EMPLOYEES~FAMILY_NAME
o
o
```



```
o
MapKey~EMPLOYEES~EMPLOYEE_ID
o
o
o
```

FIGURE 4. Example Control File for Mapping to the Target

In the examples, you see a portion of a main Control File and two files to be included. The main Control File sets the LoaderName and defines some logging[1] and has the include_file keywords for each of the other segments. The first segment defines one source database table with two columns shown and other columns not shown. The second segment shows the mapping of the EMPLOYEE table to the target and defines the primary key. In the examples, each of the columns is mapped to a similar column in the target, although the second column has a different name in the target than in the source.

Note that comments are permitted and must be prefixed by an exclamation mark (!). None of these lines use the backslash continuation character, but that too is supported.

The example does not illustrate mapping a column to more than one target table, filtering, materialized columns, or of a host of other control options. There are additional examples throughout the chapter.

Building the Metadata Control File

The Loader kit includes a procedure to help you build the portion of the Control File that describes the metadata in the source and to supply the map statements for the target, as well.

The syntax is

```
jcc_lml_create_control_file <database name> [option1 [option2]]
```

1. See “Keyword: Logging” on page 256.

Note that database name is required. This is the name of the source database. If it is missing, the following message results.

```
%DCL-W-ARGREQ, missing argument - supply all required arguments
```

This procedure supports two results and will create one or both of them, depending on the options specified. The options are TABLE, MAPTABLE, or ALL. TABLE is the default, as that protects upward compatibility with pre-Version 3.0 releases of the Loader. *ALL is the recommended option for new projects, particularly if they will have detailed mapping to the target.*

- TABLE: A Control File is generated containing the source TABLE and COLUMN descriptions for each table with the NoMapTable option specified for the Table keyword. The MapTable will not be created.
- MAPTABLE: A Control File is generated containing the MAPTABLE, MAPCOLUMN, and MAPKEY definitions for each source table.
- ALL: Both styles of Control File are generated. All must be listed as option1.

Once the metadata portion of the Control File is generated, the files can be edited to tailor them further to your needs.

Example

```
jcc_lml_create_control_file my_source_db ALL
```

Running the procedure this way will generate a Control File for defining all the source TABLE and COLUMN descriptions *and* will generate a Control File with all the MAPTABLE, etc definitions. Editing the Map ... definitions can tailor what happens.

Handling DBKey as the Primary Key

One of the options that can be specified is originating_dbkey. With this the syntax is

```
jcc_lml_create_control_file <database name> originating_dbkey
```

With this addition, the procedure also creates a file that will have the necessary SQL table statements for tables that have no discernible primary key. The file is named <db file>_ODBKEY (where <db file> is the name of the database root file.

Metadata Versions

The LogMiner will not support metadata changes in a journal file without an interruption in processing.[1] The JCC LogMiner Loader is correlated strongly to a particular set of physical metadata in the source database. This is because the Loader must actually translate the binary data that is presented in the LogMiner output and interpret these binary fields in terms of the underlying database data types.

The Loader was originally designed so that the Loader [2] can be run on a machine where the source database is not present. Accordingly, the entire table definition for each managed table must be specified in the Control File. This means that you must specify each column and its name in the source database, the relative position of that column in the unload record (which corresponds to the Rdb column ordering specified in the table RDB$RELATION_FIELDS), the field data type, scale and sub-type. You will also want to specify, if the column is to be included in the target, where the column is to be mapped in the target.

If the Loader detects an input record for a version of a table that is not consistent with that specified in the Control File, it will report an exception and exit.[3]

Statement Ordering

With a few exceptions, statements in the Control File may appear in any order. The exceptions and some recommendations are included in the following. Special requirements for JDBC, API and Tuxedo targets are also included.

Requirements

The few ordering requirements are

1. See “Metadata Changes in the Source Database” on page 451 for the detailed steps to support metadata changes.
2. See “Modes of Operation” on page 29.
3. Please see “Metadata Changes and Mapping the Source to the Target” on page 452 if you suspect that you may have an older version of a row included in an AIJ that must still be processed.

- LoaderName may be omitted, but it may not occur after any other keyword has been processed. If omitted, the default for the LoaderName is "JCCLML". If the logical name JCC_LogMiner_Loader_Name is defined, that value will override the default.
- The output keyword must be defined *before* any metadata is defined. If it is not, an exception message will be generated and the Loader will exit. The exception message will be of the format

```
%dba_parse_init_file: source/target metadata declared before target
type in file: LOADER_TARGET
line: output~OCI~synch~lmldev~record~trim
```

- Tables (source tables) and columns must be defined *before* excludes, filters, or virtual columns are defined for those tables and columns.
    - Tables must be defined before columns in the table.
    - Virtual columns must be defined after all other columns in a table, but before any excludes or filters.
- The source table (table, column, primary key, virtualcolumn) must be completely defined prior to creating any maptables for the source table.
    - MapTable must be declared before being used in MapColumn, MapKey, MapExclude, or FilterMap declaration.
    - A MapColumn must be declared for a target table before it can be used in a MapKey, MapExclude, or MapFilter declaration.
    - All source components and all target components used in MapResult must be declared before MapResult is included.
- Before the input_failure keyword, the input type must be defined to IPC (mailbox) and checkpointing must be enabled. If either of these features is not enabled at the time the INPUT_FAILURE is detected in the Control File, the Loader will exit.
- If the DATE_FORMAT keyword is used, it must be placed after the OUTPUT keyword and before the first date column is declared.

Recommendations

JCC has a few recommendations, as well.

- The Loader can be set to echo the Control File in the log. This can be a major benefit in problem reporting and resolution. JCC strongly recommends placing this line either first or second in the Control File. (Loadername must be first, if used.)


```
logging~initialization
```

The result is similar to set verify in DCL.

- For readability, JCC recommends that the specifications for each table be grouped together starting with the primary key columns and that columns be specified in the order that they appear in the table.
- It should be noted that include files are read and parsed after all statements in the main file are processed. This means, for example, that you will not get what you may be expecting if you have an exclude statement (to exclude a table) in the main file when the table is not defined except in an include file. The parser will fail when it reaches the exclude statement for the undefined table.

JDBC, API (XML), and Tuxedo Ordering Requirements

There are also some JDBC, Tuxedo, and API related ordering requirements that apply only if you are using one of these targets:

- The OUTPUT keyword (setting the output type to api) must be before any API keywords.
- The OUTPUT keyword (setting the output type to API or FILE and setting the output conversion to XML) must be before any XML keywords.
- The OUTPUT keyword (setting the output type to JDBC) must be before any JDBC keywords.
- The OUTPUT command (setting the output type to Tuxedo) must be before any TUXEDO commands.
- The TUXEDO~fieldheader~<filename> keyword must come before table definitions, if using a Tuxedo target.

Controlling the Operation of the Loader

The Control File provides a number of ways for you to control the operation of the Loader. Some examples are:

- The LogMiner Loader generates statistics about its activities during a run. You can use the Control File to specify which statistics are displayed. See “Keyword: Logging” on page 256.

- The Loader commits transactions under direction of the Control File. By default, the Loader commits its own transaction after each transaction in the LogMiner output. However, in some cases, performance is enhanced by having the Loader commit its transaction after, say, 100 transactions from the LogMiner's output. See "Keyword: Checkpoint" on page 231.
- It is possible to control the Loader response to exception conditions. See "Keyword: Input_failure" on page 248 and "Keyword: Output_failure" on page 278.
- The Loader can be directed to use parallel threads to speed the processing. See "Keyword: Parallel" on page 279 and "Keyword: TableOrder" on page 290.

Other sections provide additional examples.

Keyword Statements

The Control File is composed of keyword statements followed by one or more parameters. This section discusses common characteristics of the keywords. The following sections discuss each keyword in detail.

Common Characteristics of Keywords

The use of keywords is straightforward.

- The order of keywords is not critical, except as noted in "Statement Ordering" on page 224.
- The keyword statements are *not* case sensitive, except
    - API's p4
    - the value used with FILTER
    - items in quotes for FILTERMAP, MAPRESULT, or MAPCOLUMN
    - portions of the JDBC keyword parameters
- Embedded spaces are not allowed in most keywords. They are possible in the PRIMARY KEY keyword, in the SQL predicate for FilterMap, the SQL for MapResult, or the value for null (specified in quotes in the MapColumn keyword).
- The backslash ('\') is used as a continuation character such that a single Control File statement may span several contiguous lines. When a continuation charac-

ter is detected, the input line trailing spaces are trimmed and, on subsequent lines, both leading and trailing spaces are trimmed prior to interpretation.

- The tilde character (~) is the delimiter between parameters associated with a keyword.
- The number of parameters for each keyword varies with the keyword and the desired functions.
- Some parameters for some functions are optional. (These are shown in square brackets in this document.)
- Components of a field, if there are any, are comma separated lists.
- There are 64KB available for each keyword and its parameters.
- The keyword include_file allows inclusion of other scripts and, therefore, allows structured management of Control Files.

List of Keywords

Each keyword is described more completely in one of the following sections. Keywords are discussed in alphabetical order. “Summary” on page 311 gives a brief list showing the syntax for each.

The keywords are:

API
CHECKPOINT
COLUMN
DATE_FORMAT
EXCLUDE
FILTER
FILTERMAP
INCLUDE_FILE
INPUT
INPUT_FAILURE
JDBC
LOADERNAME
LOGGING
MAPCOLUMN
MAPEXCLUDE
MAPKEY
MAPRESULT
MAPTABLE
OPERATOR
OUTPUT
OUTPUT_FAILURE
PARALLEL
PRIMARY KEY
SORT
TABLE
TABLEORDER
THREAD
TUXEDO
VALIDATION
VIRTUALCOLUMN
VIRTUALTABLE
XML

Keyword: API

The API keyword is used only for those Loader sessions that are designed to have the Loader send data to a customer-supplied API. This action is specified by the OUTPUT keyword. The API keyword is used to define the customer-supplied API to the Loader. It must be specified three times, once for each of the three routine types. The routine type may be CONNECT, SEND, or DISCONNECT. These are API specific options and are discussed in "XML for File or API Targets" on page 199.

Syntax

The general syntax is

API~<routine type>~<routine name> \
[<additional parameters or not depending on routine type>]

Specifically, syntax is:

API~CONNECT~<routine name>[~p2 text string>[~<p3 text string> \
[~<p4 text string>]]]

API~SEND~<routine name>

API~DISCONNECT~<routine name>

Parameters

<routine type>. When the Loader calls a customer-supplied routine, which routine is called depends on the routine type. Routine type, as shown in the syntax section, is either CONNECT, SEND, DISCONNECT. The routines are of the format

int msgConnect(&handle_ptr, ip, port, topic_name, loadername_dbname, timeout)
returns Success, Failure status

int msgSend(&handle_ptr, size, msg)
returns Success, Failure status

int msgDisconnect(&handle_ptr)
returns Success, Failure status

For further discussion of these routines see “API Routines” on page 210.

<routine name>. Routine name is the entry point name in the shareable image specified in the OUTPUT keyword.

<P2>. This parameter is valid only for <routine type> = CONNECT and provides the opportunity to pass a string to the connect routine. The interpretation of the string passed to the connect routine is determined by the connect routine itself.

<P3>. This parameter is valid only for <routine type> = CONNECT and provides the opportunity to pass a string to the connect routine. The interpretation of the string passed to the connect routine is determined by the connect routine itself.

<P4>. P4 is like P2 and P3, except that it is case sensitive.

Note: The P4 parameter is case sensitive. That is, the exact case in which the value is entered into the Control File is the case that is supplied to the API.

Examples

api~connect~msgConnect~192.84.218.1~26051~test

api~send~msgSend

api~disconnect~msgDisconnect

Keyword: Checkpoint

In most cases, the checkpoint keyword defines “the commit interval.” However, the Loader interprets the checkpoint keyword differently depending on the target type.

If the target is either an Oracle or Rdb database or JDBC, the checkpoint keyword is used to specify how many incoming source database transactions are to be grouped into an output database transaction. This is referred to as a *commit interval.*

For Tuxedo targets, the commit interval has similar meaning. The number of source transactions indicated is the number that is bundled in the FML packet(s).

However, if the target is a disk file in modified XML format, by default, the checkpoint keyword represents the number of hours to wait before closing the output file and reopening a new one.[1]

There will always be only one transaction per XML document.

In all cases, the default is one.

See also “Interpretation of Lock Conflicts” on page 403 and “I/O Management” on page 393.

Syntax

CHECKPOINT~<commit interval>[~<checkpoint stream type> \
[~<synchronous>[~<checkpoint target>]]]

Parameters

<commit interval>. If the target is anything but a file, the commit interval is the number of source database transactions to include in a single target transaction. If the target is an output file, by default, it represents the number of hours to process before closing and reopening the output file.[2] If the target is XML, the commit interval must be one and only one source transaction will be included per target transaction. In all cases, the default is one. [3]

<checkpoint stream type> ***optional.*** By default, if the Loader is sending data to an Oracle or Rdb database, the Loader will checkpoint to the high-water table in the database. The default value of checkpoint stream type is OCI for Oracle databases and RDB for Rdb database targets.

1. See “Writing to a File” on page 212 for a discussion of file targets for testing or other purposes.
2. See “Change the Units for Checkpoint Intervals” on page 213 and “Change the File Flush Interval” on page 213 for further discussion of checkpoint interval influence on closing a file and on overrides that are available.
3. If you are writing to a file for testing purposes, see “Writing to a File” on page 212 for finer control.

If the Loader is sending data to any other target, the checkpoint stream type is LML_INTERNAL.

If this parameter is set to LML_INTERNAL for an Rdb or Oracle database target, then the Loader will checkpoint to a local file, overriding the checkpoint table in the database.

<synchronous> ***optional*****.** This option affects only the LML_INTERNAL checkpoint. There are two options for this, SYNCH and ASYNCH.

This parameter is ignored if the checkpoint target is a database. In that case, the checkpoint is always part of the commit operation and is, therefore, synchronous.

Asynchronous writes are faster for the Loader. However, if the Loader has shut down in a disorderly fashion, asynchronous writes can cause the Loader to re-send a transaction after a restart. In this case, the API must be able to handle the re-send.

<checkpoint target> ***optional*****.** The checkpoint target is the name of the target file to checkpoint into. The default value is

jcc_tool_data:$<loadername>$.JCCLML_checkpoint

Checkpoint Target

The Loader may checkpoint to a special high-water table in the Rdb or Oracle database or to a local checkpoint file. A local checkpoint file is required when the target is not an Oracle or Rdb database.

The checkpoint file/table contains two records for every potential Loader thread configured for the Loader family. For each thread, the Loader writes to these records alternately.

Each record stores:

- the Rdb high-water context of the last transaction sent to the target and
- the Loader sequence number of the transaction.[1] The Loader sequence number may be materialized in some output messages. (See also “Keyword: VirtualColumn” on page 299.)

The Loader checkpoint file/table contains 100% of the restart context for the Loader.

Checkpoint States

Checkpoints (represented as the value of JCCLML$COMPLETION_FLAG) can be in any of the following states.

TABLE 1. Checkpoint States

State	Full Name	Meaning
I	Initialized	Checkpoint being initialized
N	iNcomplete	At checkpoint time, saving data to output target
Y	complete	Loader is not running. In the case of continuous mode (CLML), the Loader has been shutdown using jcc_clml_shutdown. In the case of the original static mode, the Loader has reached the end of the input stream.
S	Stale	Loader has identified that the checkpoint information may be no longer current.
R	Read-run	At checkpoint time, data was read from CLM mailbox and not yet committed to target.

See also “Examining the Checkpoint Rows” on page 470 in the in the chapter “Aids for the Administrator”.

Timing Considerations

There are several tuning options for the commit interval intended to address specific scenarios. See also “Keyword: Input_failure” on page 248.

Examples

These examples show checkpointing to a file and checkpointing to a database with a commit interval of 10.

1. The Loader Sequence Number (LSN) is incremented by one for each transaction that contains data that is “of interest”, given the Control File’s definitions. Other transactions are called “no work” transactions. “No work” transactions do not cause the Loader to increment the LSN. See “No Work Transactions and Checkpoint Intervals” on page 389.

```
checkpoint~1~lml_internal
checkpoint~10
```

Keywords: Column and Primary Key

These keywords were important before MapColumn and MapKey were available in the LogMiner Loader. They may still be used. However, MapTable, MapColumn, and other keywords provide advanced control. See “Keyword: Map...” on page 260.

These keywords both specify columns in the source table. Each keyword uses the same parameter list. Each column in the source must be defined for any table that is included.

The discussions “Keyword: Table” on page 286 and “Keyword: MapTable” on page 271 further describe source and target specification. See “Keyword: Primary Key” on page 281 and “Identifying Rows in the Target” on page 39 for additional discussion of primary keys in the source and target.

There are tools in the Loader kit to help automate Control File generation. See “Building the Metadata Control File” on page 222.

Syntax

Both column and primary key keywords accept parameters. The syntax is one of two types

```
COLUMN|PRIMARY KEY~<table name>~<column name>                    \
[,<target column rename>] ~<position>~<length>~<scale> ~<type>~<sub type>
```

OR

```
COLUMN~<table name>~<column name>                                \
[,<target column rename>] ~<position>                            \
~<BLOB IGNORE>|<BLOB>|<IGNORE>
```

The first option is the more common syntax. The second relates to segmented string columns. Note that segmented string columns cannot be a part of the primary key.

Note also that the LogMiner does *not* handle segmented strings. You may use this syntax on tables containing segmented strings to specify that the Loader should ignore the segmented strings.

See “Example of a Control File Portion” on page 220.

Parameters

<table name>. This specifies the name of the table in the source database.

<column name>. This specifies the name of the column in the source database.

<target column rename> ***optional.*** When used, this optional parameter is preceded by a comma. It specifies the name of the column in the target database. If omitted it will default to the same name as in the source database.

The output column name may be required with an Oracle database because Oracle names are limited to 30 characters and the name in the Rdb source may be 31.

Note that the keyword mapcolumn is an alternate way to define target column names that differ from the names used in the source table. The map... keywords support definition of multiple target tables for one source table. The map... keywords are also the newer and clearer specification. See also “Logging and Performance” on page 258, “Keyword: MapColumn” on page 261, “Keyword: MapKey” on page 265, and others referenced by those.

<position>. This parameter represents the ordinal position of the column in the table. Although relational concepts do not support the concept of physical record location, in fact the Loader must understand this in order to decompose a binary structure received from the LogMiner.

It should be noted that it is entirely possible that two distinct databases have the same columns in a particular table and that the ordinal position of those columns differs between the two databases. This is caused by metadata changes being applied to the two databases in different orders. Such databases would require different Control Files.

<length>. This is the length of the column.

<scale>. For scaled integers, this represents the scale of the column. The default is zero.

<type>. This represents the datatype of the item. All Rdb datatypes are supported.

<sub type>. This represents the subtype of the item. All Rdb subtypes are supported. The default is zero.

<blob ignore> or <ignore>. The column will be omitted from the output table.

The Rdb LogMiner does not extract segmented string data. Instead the dbkey of the first element of the segmented string is presented. This value has no possible use in the target database and such a column should be ignored.

Instead of the position parameter, the segmented string column should include one of the alternatives shown.

<blob log> is reserved and, at the moment, is a NOOP.

Exception

When a table is replicated, the Loader generates dynamic SQL. The size of a dynamic SQL string is inherently limited by Rdb SQL and OpenVMS to 64K bytes. If the length of the generated string exceeds this value the Loader will fault.

The number of columns supported by the Loader will vary depending on the table and column name lengths. The number is large, but the limit is inexact.[1] If this is a limitation to your application, JCC suggests that, in the Loader Control File, you rename the columns in the table definition.[2] This technique will allow the Loader to support a few more columns.

1. Initial projections suggested that the Loader could handle tables with at least 350 columns. At one site, the Loader is successfully processing 518 columns. Current estimates suggest that tables with in excess of 700 columns can be processed. A (possibly inaccurate) way of estimating the length of the generated SQL is (56 + (2 * average length of column name)) * number of columns.
2. There is no requirement that output column names in the Control File match those in the source database. Target column names can be modified with the optional parameter to the keyword Column or with the target specific keyword MapColumn. See “Keyword: MapColumn” on page 261.

Additional exceptions apply to the primary key keyword. See “Keyword: Primary Key” on page 281.

Keyword: Date_format

For targets other than Rdb and Oracle databases, use the date format keyword to determine the format in which date-time columns are presented in the output.

Note that this keyword, if specified, must be in the Control File after the output keyword and before the first source date-time column is defined.

Also see “Tuning Considerations” on page 465 and “Addressing Data Issues” on page 471.

Syntax

DATE_FORMAT~<date format>

Argument

Possible values are anything that can be passed as an argument to the OpenVMS date format routines. For standard OpenVMS date, this is documented[1] as

```
"|!DB-!MAAU-!Y4|!H04:!M0:!S0.!C2|”
```

The default is shown here as an example.

Example

```
date_format~|!Y4!MN0!D0!H04!M0!S0!C5|
```

1. See the OpenVMS Programming Concepts Manual section 27.6.2.7 Specifying Output Formats at Compile Time

Keyword: Exclude

The Exclude keyword is maintained primarily for reasons of compatibility with prior versions of the Loader. See “Keyword: MapExclude” on page 264 and “Exclude and MapExclude” on page 240.

The exclude keyword is used to suppress output for selected source tables or selected columns in source tables.

It is particularly useful to extract a complete metadata definition of a database into a single metadata file, as described in the section “Building the Metadata Control File” on page 222. This definition will include all tables and all columns of those tables. [1] The exclude keyword can then be used to limit the output of the Loader.[2]

See also “Keyword: MapExclude” on page 264.

Syntax

EXCLUDE~<table>[~<column>]

Parameters

<table>. Specify the name of the source table to be excluded. If no columns are specified, the entire table is excluded from the output.

<column> ***optional.*** This parameter specifies the name of the column to be excluded. If several columns are to be excluded from a source table (but some of the columns will not be excluded), there must be an exclusion of each column separately.

1. See also “Building the Metadata Control File” on page 222.
2. If you are not going to use a table at all, it is more efficient to exclude it in the LogMiner options file so that less work is expended on it. See “Excluding Tables from the Options File” on page 99

Examples

If you don't want to ship the bill detail table to the target because it is more data than is needed and you don't want to ship the credit_code column from the payment table because it hasn't been maintained with reasonable data, the keyword statements would look like this.

Exclude~Bill_detail

Exclude~Payment~Credit_code

Exclude and MapExclude

Use of the exclude keyword provides a way to exclude a column from output while still having it available to use in filtering rows. See "Keyword: FilterMap" on page 242.

In the examples, the table bill_detail and the column payment.credit_code would not be sent to the target.

See "Automatic Generation of MapTable" on page 290.

See also "Keyword: MapExclude" on page 264.

Exclude and the Primary Key

If the target table is set to NoUpdate, NoDelete, the primary key column or columns can be excluded, although this is unusual.

Keyword: Filter

The Filter keyword added filter capabilities in an early version of the LogMiner Loader. The keyword FilterMap, introduced in a later release, provides much better control of filtering. See "Keyword: FilterMap" on page 242.

The keyword Filter is upwardly compatible. (That is, it is still supported.) However, the keyword FilterMap, which operates on target tables defined with the MapTable keyword, provides much greater flexibility in filtering.

The Filter keyword makes it possible to filter *source* records based on the value of a column. When Filter is used to exclude source rows, they are excluded from all mapped target tables.

The relevant Table must be defined before including a Filter.

Syntax

FILTER~INCLUDE | EXCLUDE~<table name>~<column name>=<value>

Parameters

include | exclude. Set whether the column and value specified will cause the row to be included or excluded. There can only be one include filter per column per table; but there can be many exclude filters. (The filters function as a set of ANDed conditions.) See “Examples” on page 241.

<table name>. Indicate the source table to use.

<column name>. Indicate the source column to use.

<value>. Indicate the value to compare to the value in the source column. If the value is the same (notice the “=”) as the value in the column, include (or exclude) the source row from all target tables in the target database.

If the specified column has a null value, the filter will not apply.

Examples

1. For example, to include rows from the source database table “Details” only if the column “code” has a value of ‘SUM’, use the following:

```
!
!       Include only the summary records
!
filter~include~details~code='SUM'
```

   This example specifically includes all rows from the detail table where code = ‘SUM’ and excludes all other rows.

2. This (unrealistic) example shows how to move to the target all rows except those excluded.

```
!
!   Include only the payment records for partial
!   and late payments by excluding the ones paid
!   on time and completely.
!   Also, exclude the payment type CASH.
!
filter~exclude~payment~credit_code='PAID'
filter~exclude~payment~payment_type='CASH'
```

3. The following is a valid example because the include clauses use different columns in the table.

```
filter~include~payment~credit_code='PAID'
filter~include~payment~payment_type='CASH'
```

4. The following is ***not*** a valid example because the include clauses use the same column in the same table. ANDing the two clauses would yield nothing to include.

```
filter~include~payment~credit_code='PAID'
filter~include~payment~credit_code='LATE'
```

Exception

There are materialized (virtual) columns that cannot be used with filters. See “Virtual Columns and Filters” on page 303.

Keyword: FilterMap

FilterMap differs from Filter in two significant ways.

1. FilterMap brings the power of interactive SQL to filtering. FilterMap supports any SQL restriction that only operates on a single row. Filter is more limited.
2. Filter works with the source table and, therefore, influences all target tables. FilterMap works with a maptable specification. If a filter should be the same for all target tables (and it can be specified in the more limited syntax of the keyword Filter) use Filter and the analysis is done once. If the filter result should be different for different maptables, use FilterMap.

Because SQL predicates are more flexible than the specifications available with the keyword Filter, the include|exclude choice that is part of the keyword Filter is not necessary with FilterMap. For FilterMap, rows that satisfy the "where clause" are *included*.

The relevant MapTable must be defined before including a FilterMap.

FilterMap Database

FilterMap requires its own database[1], since it utilizes the Rdb SQL predicate language and engine to determine whether a given row passes the test for inclusion. Rather than attaching to the source database for this purpose and using valuable user limit slots, a separate database is used. The name of this small database is

```
JCC_TOOL_DATA:<LoaderName>.rdb
```

To change the directory name, use the logical name JCC_LOGMINER_LOADER_FILTER_DIR. The name specified must be a valid device and directory. The Loader will append "<Loadername>.rdb" to create the full file specification.

To change the database name, use the logical name JCC_LOGMINER_LOADER_FILTER_NAME. (See also "Controls for the Filter Database" on page 461.)

All Loader threads for a given LoaderName will attach to the same database. Each will start a single read-only transaction and never commit.

If the database is not found or the database found has the wrong version of Rdb, the Loader will create a database with the following format.

```
CREATE DATABASE FILENAME <name>
    OPEN IS AUTOMATIC
    NUMBER OF USERS 33
    NUMBER OF CLUSTER NODES 1
    PAGE SIZE IS 4 BLOCKS
    BUFFER SIZE IS 4 BLOCKS
    GLOBAL BUFFERS ARE ENABLED (NUMBER IS 330,
                                USER LIMIT IS 10,
                                PAGE TRANSFER VIA DISK)
```

1. See "Keyword: MapResult" on page 266 for additional uses made of this database.


```
DBKEY SCOPE IS TRANSACTION
PRESTARTED TRANSACTIONS ARE OFF
DICTIONARY IS NOT REQUIRED
SHARED MEMORY IS SYSTEM;
```

If your version of Rdb is 7.3.1 or greater and your version of the Loader is less than 3.4.3, you will need to hand create the filter map database, due to a change in Rdb's defaults in version 7.3.1. You can do this with the Create database command shown in this section.

Syntax

FilterMap~<map table name>~[where]<sql restriction>

Parameters

<map table name>. map table name

<sql restriction>. any valid SQL where clause that operates on a single row. Note that the actual word "where" is an optional part of the syntax.[1]

Examples

1. The first example for FilterMap shows how to include the rows from a source table named 'details' only if a column named 'code' has a value of 'SUM'. If you have two target tables and have defined them with maptable names of 'sum_of_details' and 'transactions' and you want to write the rows with code = 'SUM' to the target table defined by sum_of_details and all other rows from the source to the target table defined by transactions, you can use the following

```
!
!       Include only the summary records
!
filtermap~sum_of_details~code='SUM'
!
```

1. For any VirtualColumn used in the SQL restriction, the column name to use is the output column name, if any, and the actual virtual column name, if it has not been renamed. See "<output column name> optional" on page 300 as part of the discussion of the Virtual-Column keyword.

```
!       Include the transactions
!
filtermap~transactions~code<>'SUM'
```

2. FilterMap can also support much more complex statements. For example

```
!
!       Example of more complex SQL
!       including only specific sorts of comments
!
!       Note that the where is optional
!
FilterMap~comment~where (comment_cd in (55,67,89))\
or (comment_type = 'MISC' and comment_cd = 100)   \
or (comment_type = 'ACCT' and comment_cd in (99, 53))
```

Exceptions

As for the keyword Filter, there are materialized (virtual) columns that cannot be used with mapfilters. See “Virtual Columns and Filters” on page 303.

Only one FilterMap can be declared for a given MapTable. If a second is declared, the log will include the message

```
%dba_parse_init_file: invalid record - FilterMap has
already been defined for this table in file: <file-
name> line: <input line>
```

Monitoring

Both the online monitor displays and the logs reflect filtering. See “Statistics for Filtered Rows” on page 325 and “Filtering Reflected in the Log” on page 361.

Keyword: Include_file

This keyword allows the inclusion, by reference, of additional Control Files. The parser first reads all keywords in the main Control File and then processes each included file one at a time. The file names can be any legitimate OpenVMS file specification. If you specify an invalid or non-existent file the Loader will fail.

Note that there is an implied order to this process.[1] For instance, a table cannot be excluded before it is defined. It is appropriate, therefore to group all exclude statements into a single Control File and include this file, by reference, as the last element of the main Control File. The order of Control Files that JCC recommends is:

1. Main Control File, including the Loadername.
2. Target specific choice and related components
3. Source database metadata (JCC procedures in the kit can help with this.)
4. Materialized information (virtual columns)
5. Filters (source specific filters, if any)
6. Exclude source elements (tables and columns, if used)
7. Target specific metadata (MapTable, MapColumn, etc.)
8. FilterMaps (if any)
9. MapExcludes (if any)
10. MapResults (if any)

Syntax

INCLUDE_FILE~<filename>

Parameter

<filename>. This parameter specifies the name of the file to be included. This may be a logical name or a full OpenVMS file specification.

1. See “Statement Ordering” on page 224.

Keyword: Input

The Input keyword in the Control File supports declaration of the input name, type and whether or not the data should be read asynchronously.[1]

The Loader input filename must be specified completely in order for the file to be properly located. When no other directory specification is supplied, the current directory is searched. If the file cannot be found, the program exits.

Syntax

The syntax is:

INPUT~<input type>[~<synchronous>[~<input source>]]

Parameters

<input type>. FILE | IPC (e.g. file or mailbox)

<synchronous> ***optional.*** SYNCH | ASYNCH

Use of ASYNCH is currently limited to the IPC input type. ASYNCH may NOT be used in PARALLEL mode, but input timeouts are allowed with ASYNCH. See "Keyword: Input_failure" on page 248 for more information regarding input time-outs.

SYNCH is the default.

<input source> ***optional.*** filename | mailbox name

Defaults and Uses

Defaults vary with the mode of the Loader. (See also "Modes of Operation" on page 77.)

1. The Input keyword is not required. Alternately the input name and type may be included on the command line or defined with the logical name jcc_logminer_loader_input. These alternate designations are limited to synchronous reads.

Parallel Continuous. The defaults are IPC and SYNC. The Loader will fail if the input keyword specifies any other options. The file name parameter is ignored and the generated name is used for the mailbox.

Non-parallel Continuous. The defaults are IPC and SYNC. The Loader will fail if the input keyword specifies any other option than IPC, but will use either SYNC or ASYNC. The file name parameter is used, if specified.

Original Static. The defaults are FILE and SYNCH, but other options can be specified. The file name parameter is used, if specified.

Keyword: Input_failure

The Loader permits you to set a commit interval greater than one.[1] With a commit interval greater than one, the Loader collects several transactions before passing them on. This can be a performance enhancing feature. However, if there is a point at which the updates to the application become infrequent, a stall can occur because there are no transactions to fill the commit interval. Therefore, in systems with an uneven load, running continuously and using a longer commit interval can precipitate a significant and artificial delay for some transactions.

You can specify an input timeout.[2] The input timeout should be set to the amount of time that it is acceptable for the Loader to wait for further input before it checkpoints information that is already buffered.[3]

The input_failure keyword can also be used to control "no work" transactions. See "No Work Transactions and Checkpoint Intervals" on page 389.

1. See "Keyword: Checkpoint" on page 231 for a discussion of checkpoint intervals and tuning options.
2. If the timeout is not set with the input_failure keyword *and* the logical name JCC_LOGMINER_LOADER_STALE_INTERVAL is defined, the logical name definition will be used for the timeout.
3. Note that the Loader always commits or writes to the checkpoint file on transaction boundaries. The Loader, even with the input read timeout, will never commit less than a full transaction. The LogMiner does not provide information to the Loader until the commit and the Loader does not segment transactions.

For Static or Copy mode, the input_failure keyword has a default of 5 seconds.[1]

Syntax

Syntax for the keyword is:

INPUT_FAILURE~<timeout seconds>

Parameter

<timeout seconds>. Timeout seconds is the number of seconds to wait on input before attempting to commit the currently buffered records.

Default

The default value for input timeout is 0.00 seconds which disables the input failure feature. The maximum supported value is 863,913,599.99 seconds (or just under ten thousand days, 9999 23:59:59.99, or roughly 27.4 years.) Note that timeout seconds may be specified as a floating point number.

Disabling

A value of zero will disable this feature.

Requirements

To use the INPUT_FAILURE keyword, requires

- Input type of IPC (mailbox)
- Checkpointing enabled [2]

1. See “Prepare for EOF” on page 87.
2. A checkpoint interval greater than zero will enable checkpointing; a checkpoint interval of zero disables it. Without checkpoint enabled, the commit interval is one and there is no need for input_failure to be defined.

If either of these features is not enabled at the time the INPUT_FAILURE is detected in the Control File, the Loader will exit with a failure. [1]

Recording Checkpoint Data

Even when input_failure is disabled, it is important to keep the checkpoint data accurately. Therefore, the Loader does some operations that are generally of limited interest to the user.

Timeout is disabled if all of the following are true:

- input_failure is not defined or is explicitly set to zero
- no stale interval is set
- checkpoint interval is set to one

In order to properly update the checkpoint information, the Loader sets a one second timer when the timeout is disabled. The following logic is applied when the timer expires:

- If a record is read from the CLM mailbox and the record is a record to be transmitted to the target, the timer is canceled and not reissued.
- If a record is read from the CLM mailbox is a commit record received as part of a no-work transaction, the checkpoint data is updated, the statistics NoWork counter is incremented, and the timer is reset.
- If no records are read from the CLM mailbox before the one second timer expires, the Loader writes the stale checkpoint. This also increments the statistics Timeout value under the Input header.

For reading the statistics information, it is helpful to know that the Checkpoints Timeout counter under the Output header is incremented if all of the following are true:

- the commit interval is greater than one
- a timeout value is set
- a timeout is received after reading a commit record for a transaction that had some data and before the commit interval is reached

1. See “Statement Ordering” on page 224.

A side-effect of this processing is that with checkpoint = 1, the Output Checkpoints Timeout counter will always be zero.

Keyword: JDBC

The JDBC keyword specifies behavior when using a JDBC target. The output keyword must be specified with the JDBC parameter, before the JDBC keyword is used.

Syntax

JDBC~<element>~<attribute>

The JDBC keyword may be used multiple times.

Parameters

<element>. The element options are

- driver
- connect
- classpath

Each of these has an attribute.

<attribute> for driver. The attribute for the driver element provides the JAVA class name of the individual vendor-specific JDBC driver. This value is case sensitive.

Detailed syntax is

```
jdbc~driver~<vendor-specific driver name>
```

Examples of the name for vendor-specific drivers, for MS SQLServer and for Oracle Rdb, are

```
com.microsoft.jdbc.sqlserver.SQLserverDriver
oracle.rdb.jdbc.rdbThin.Driver
```


<attribute> for connect. The attribute for the connect string is the driver-specific connect string (that is, the URL) for the target database. Format for the connect string will vary by driver. The value is case sensitive.

As this string can also be provided in the output keyword, JDBC~driver~<connect string> is not strictly necessary. If it is used, it will override the connect string specified in the output keyword, should they be different.

Detailed syntax is

```
jdbc~connect~<connect string>
```

The generic form of the connect string is

```
jdbc:<driver name>:<driver-specific reference to database>
```

Examples, for MS SQLServer and for Oracle Rdb for testing at JCC, are

```
jdbc:sqlserver://kong:1433;DatabaseName=JCCLoader
jdbc:rdbThin://atlas:1701/training_db
```

In these examples, kong and atlas for server names; 1433 and 1701 are port numbers.

<attribute> for classpath. The attribute for the classpath is an entry to be added to the Java Class Path. The value specified must be the Unix version of the OpenVMS path to the specified JAR file. For example

TABLE 2. OpenVMS and Unix paths

OpenVMS	Unix
disk:[dir1.dir2.dir3]file.ext	/disk/dir1/dir2/dir3/file.ext
logical:file.ext	/logical/file.ext

Detailed syntax is

```
jdbc~classpath~<JDBC driver-specific required Java class>
```

Repeat the syntax as many times as required to specify each Java Class Path. Note that the HP JNI Java interface that the Loader utilizes ignores the value of the CLASSPATH and JAVA$CLASSPATH logical names. All Java Archives required by a given driver at runtime must be specified using the JDBC keyword.[1]

Examples of classpaths are

```
/jcc_tool_java_lib/sqljdbc.jar
/jcc_tool_java_lib/jtds-1_2_2.jar
/jcc_tool_java_lib/mongo-java-driver-2-11-2.jar
/rdb$jdbc_home/rdbthin.jar
```

Example for SQLserver

A control file portion is given here as an example. (The backslash is a continuation character and the exclamation mark is a comment character.)

```
!
output~jdbc~synch~jdbc:sqlserver://                    \
        kong:1433;DatabaseName=JCCLoader
validation~LoaderTest~TestLoader
checkpoint~1~lml_internal
!
jdbc~driver~com.microsoft.sqlserver.jdbc.SQLServerDriver
!
jdbc~connect~jdbc:sqlserver://                          \
        kong:1433;DatabaseName=JCCLoader
!
jdbc~classpath~/jcc_tool_java_lib/sqljdbc.jar
```

Example for Rdb

A control file portion is given here as an example. (The backslash is a continuation character and the exclamation mark is a comment character.)

```
checkpoint~1~lml_internal~asynch~loader_regression_test_chkpt
!
output~jdbc~synch~jdbc.rdbThin://atlas:1701/loader_regression_test_jdbc~record
!
validation~test~example
!
jdbc~driver~oracle.rdb.jdbc.rdbThin.Driver
!
!  Specification of connect string not required if specified in the output keyword
```

1. In the examples, “jcc_tool_java_lib” is a logical name that identifies a directory of the same name. The directory is used to support the JDBC target. It provides a location for the JAR files and for the Loader JAVA code.


```
!  jdbc~connect~
!
!  The thin driver is a class 4 driver
jdbc~classpath~/rdb$jdbc_home/rdbthin.jar
```

Keyword: Loadername

The loadername keyword provides a name for each session of the Loader. There can be many Loader sessions running simultaneously against the source database or even a variety of source databases.

When running in continuous mode, the Loader modifies the process name of each member of the Loader family to be the Loader name plus four characters. The four characters distinguish which family member each process represents. Since the OpenVMS process name is limited to fifteen characters, the Loader name should be restricted to eleven characters when running in continuous mode.

The loadername must also be unique (across a cluster) for Loader families sending data to the same target.

The loadername is used in several important ways.

- Distinguish the different high-water rows
- Name the global section where the Loader keeps track of progress in realtime.
- Distinguish the processes that make up the different Loader families.

For example,

- In order to improve throughput, you may run one Loader session to update a single table that has a high rate of change and run another Loader session to update all the other tables. These two (or more, if you divide the problem further) families can run concurrently and will utilize different high-water rows. You can, alternately, create a Loader family with multiple parallel threads.
- You may run multiple Loader sessions to create multiple copies of the source database or of *portions* of the source database.
- You may use multiple sessions of the LogMiner Loader concurrently to consolidate changes from multiple transaction databases into a single database. Each Loader family member would utilize a different high-water row.

Syntax

LOADERNAME~<text string>

LOADERNAME, if specified, must be the first entry in the Control File.

Parameter

<text string>. Supply a string naming this session of the Loader. The Loadername text is not case sensitive. The text string must be unique (in the first 11 characters) between all Loader sessions maintaining the target database. Note that long names will affect the length of XML messages and, therefore, can interfere with performance for customer-supplied APIs. The default is JCC_Loader.

Example

Loadername~loader_01

Exception Handling

On traditional OpenVMS file systems, many characters are invalid for use in filenames. Use of any of these invalid characters in the LoaderName has resulted in an exception while attempting to open the Loader or Continuous LogMiner log files. To avoid an OpenVMS exception message that is generic and can be confusing, the Loader validates that the characters specified for the LoaderName keyword are only alphanumeric or a hyphen ("-") or an underscore ("_"). If they are not, an exception message is provided and the Loader shuts down.

The exception message will be of the form:

```
%dba_parse_init_file: invalid record in file: jcc_logminer_load-
er_init
line: LoaderName~S UB*RDB%05
LoaderName includes invalid characters
'S UB*RDB%05'
  ^  ^   ^

%DBA-E-INV_INIT_RECORD, Invalid initialization record encountered.
```


Using a Logical Name to Define the LoaderName

Instead of specifying the Loadername in the Control File, you may specify it with a logical name, JCC_LogMiner_Loader_Name.

The translation value of this logical will be used by the JCC LogMiner Loader as the default LoaderName and will be used if the LoaderName keyword is not specified in the Loader Control File.

Keyword: Logging

The logging keyword is used to control information that is reported by LogMiner or the Loader and written into log files. Multiple instances of the logging keyword can be used to assemble the desired results.

See also “The Log Files” on page 356 for a more complete treatment of logging.

Syntax

There are two groups of logging types: those that take an additional parameter and those that do not.

LOGGING~<OUTPUT | INPUT | STATISTICS>~<logging options>
LOGGING~<TRACE|LOCKING|INITIALIZATION>

Parameters

<type>. Type is specified as the first parameter to the logging keyword. The implications of each are shown in the table to follow.

<options>. Logging options vary depending on type and only some types, as indicated, have options. Logging options are a comma-separated list.

Logging and Verbosity

With the logging keyword, you have it in your power to produce a log that is too voluminous to ever fully review. In the following list, the logging options are color coded to suggest use.

- The green for logging~initialization is a strong suggestion that you include it in every Control File so that the Control File is included in the log.
- The yellow is a warning that the results of this logging option will be too voluminous to suit some users.
- The red is a warning that the logging option may be important for tracing an issue, but is very verbose.

TABLE 3. Logging Keyword Usages and Meanings

Usage	Meaning
Logging~output~trace	**Output a trail of the output phase of execution**
Logging~output~ dynamic_data	**Use SQL trace to output data from within SQL (including NULLs). The appropriate Rdb debug flags must be set for this to be useful. (Enabling this option can limit the number of columns that can be supported per table.) Write the data from within the LML code before sending it to OCI, XML, or Tuxedo.**
Logging~output~record	**Dump the output records, before other processing**
Logging~output~synchronization	Display the dbkeys that are causing stalls in parallel mode.
Logging~input~trace	**Output a trail of the input phase of execution**
Logging~input~record	**Dump the input records, as they are read, before any other processing.**
Logging~input~brief	**Dump a brief description of each record read.**
Logging~input~ [NO]log_restart	Output (or not) a line for each input record ignored on highwater restart. Default is nolog_restart.
Logging~input~ [NO]ignore_unknown_tables	Discard tables not defined in the Control File (or fail on records for undefined tables). Default is noignore_unknown_tables, (fail on records for undefined tables).
Logging~input~filter	Output a line for each record that is *excluded* due to a filter, including what column and value caused the discard. See “Keyword: Input_failure” on page 248.
Logging~input~nowork	Output a line (or so) each time a “nowork” transaction is encountered. (See “Keyword: Checkpoint” on page 231 and “No Work Transactions and Checkpoint Intervals” on page 389.)

TABLE 3. Logging Keyword Usages and Meanings

Usage	Meaning
Logging~input~synchroniz-tion	Provide statistics on the interaction of threads waiting for the read lock on the mailbox.
Logging~output~filter	Output a line for each record that is *included* due to a filter. Note that, if both this and logging~input~filter are included, the log will show one entry for every record that is tested by a filter. The log will, however, indicate whether the record is included or not.
Logging~statistics~runtime	Output lib$show_stat information at the beginning and end of the run.
Logging~statistics~commit	Output lib$show_stat information at each commit interval.
Logging~statistics~timer	Output the collected timer results, periodically.
Logging~trace	**Add to the log a trail of the operation of the Loader. Logging~trace can generate very large logs!** **Note that trace can also be used as an option with logging types input or output to limit the amount written to the log to either the input work of the Loader or the output work of the Loader. Logging~trace includes the output of each of these, plus some additional statistics from both initialization and runtime.**
Logging~locking	**Display the locking in the log.**
Logging~NoSort	**Display the benefits of sort~NONE. See "Sort Avoidance Optimization" on page 397.**
Logging~heartbeart	**Logs an entry every time a heart beat event occurs. Verbosity will depend on the heartbeat interval. See "Loader Heartbeat and AIJ Backup" on page 475.**
Logging~initialization	**Echo the Control File in the log. JCC recommends adding Logging~initialization to the Control File immediately after the Loadername keyword, if any. This can be a major benefit in problem reporting and resolution. The result is similar to set verify in DCL.**

Logging and Performance

The priority of the Control (CTL) process is set higheer than the subprocesses to enable the CTL logging facility to process logging reports more efficiently.

Example

These logging settings are commented out because they are only used for debugging.

```
!logging~output~trace,sql,record,dynamic_data
!logging~output~trace,sql,record
!logging~output~sql
!logging~output~synchronization
```


Keyword: Map...

The “map” keywords (maptable, mapcolumn, mapkey, mapexclude, filtermap and mapresult) are available in the Loader to support sophisticated mapping of source database columns to target database columns.

The keywords table, column, primary key, exclude, and filter apply to the source database. In early versions of the Loader, they were also used to define the target. The Loader is upwardly compatible and the source table keywords can still be used to define a single target table for a given source table. However, the target specific keywords support greater flexibility and are better designed for future features.

The target specific keywords can define a mapping of one source row to more than one target table. They also support more flexible filtering and data transforms.

For more on the target specific keywords see

- “Keyword: MapTable” on page 271
- “Keyword: MapColumn” on page 261
- “Keyword: MapKey” on page 265
- “Keyword: MapExclude” on page 264
- “Keyword: FilterMap” on page 242
- “Keyword: MapResult” on page 266

For more on the source specific keywords see

- “Keyword: Table” on page 286
- “Keywords: Column and Primary Key” on page 235
- “Keyword: Primary Key” on page 281
- “Keyword: Exclude” on page 239
- “Keyword: Filter” on page 240

Also see “Sort Order and MapTables” on page 284.

Keyword: MapColumn

MapColumn is the target specific version of the keyword Column. MapColumn is used with MapTable to support writing a single input row to multiple target rows.

MapColumn is not required in the Control File, unless there is to be more than one target table for a source table. MapColumn is also not required in the Control File for a column that is not to be included in the target table.

See “Logging and Performance” on page 258, as well as the other sections referenced there. See also “Keywords: Column and Primary Key” on page 235.

Syntax

MapColumn~<map table name>|*~<source column name> \
[,<target column rename>][~<value if null>]

or, for VirtualColumns only,

MapColumn~<map table name>|*~ \
<source (virtual) column name|output column name> \
[,<target column rename>][~<value if null>]

Parameters

<target table name>|*. target table name or asterisk. Including a specific target table name performs the work for only that table; including the astrisk, instead, performs the function for all tables.[1]

<source column name>. name of column in the source table (which for VirtualColumns is the virtual column name) or the output column name defined in a VirtualColumn keyword.

<target column rename> *optional*. name of the column in the target table The name of the column in the target is, by default, the same as the name of the column in the source.

1. Wildcarding, as using the asterisk is called, is more likely to be used with VirtualColumns than with individual columns from the source tables. See “Wildcarding” on page 303 for examples.

<value if null> ***optional*****.** If a column in a source database is Null, it can be converted to a discrete value in the target by specifying the optional <value if Null> parameter of the MapColumn keyword. The value can be numeric, date/time, or character, but must match the data type of the column for which it is declared. Character values must be enclosed in single quotes, numeric and date/time may not.

If a <value if Null> value is specified for a column and that column is Null for a row, the specified <value if Null> value will be used in all cases where the column is referenced by *target specific* keywords. This includes references in MapFilter restrictions, as well as in what is output to the target. In all cases, the Loader will treat the column (for purposes related to the target) as if the source database contains the <value if Null> value.

However, operations on the source will interpret the column as Null. Specifically, the Filter keyword will see the column as Null.

See also “Comparing Character Data” on page 135 for a discussion of the interaction of the trim operation and the interpretation of zero length strings.[1]

See also “Keyword: MapResult” on page 266 for more flexible data transforms.

Examples

If a source table named customer contains information on both individual and commercial customers with columns customer_id, first_name, last_name, company_name, contact_name, and others, you might wish to have a target table for individual customers and one for commercial customers. If these are named customer and commercial, your Control File might include

```
MapColumn~customer~customer_id
MapColumn~customer~first_name,given_name
MapColumn~customer~last_name,surname
...
MapColumn~commercial~customer_id
MapColumn~commercial~company_name
MapColumn~commercial~contact_name
...
```

1. The section, as written, is specific to Oracle. However, it has wide applicability.

One set of mapcolumn definitions for the customer table will be generated automatically by the source table definitions. (This automatic generation is disabled with the NOMAPTABLE option. See “Automatic Generation of MapTable” on page 290.)

By default, the automatically generated MapColumns will use the same column names in the target as in the source. The example shows the ‘first_name’ (as the source column) renamed in the target to ‘given_name.’

Additional examples show the value if null option:

```
MapColumn~people~people_id~-1
MapColumn~people~init,middle_initial~' '
MapColumn~people~birthday~17-Nov-1858 00:00
```

MapColumn and Virtual Columns

If a virtual column is required as a column for a MapTable, it must first be declared for the source table. If the virtual column is not desired in the target table, it can be excluded. See “Keyword: VirtualColumn” on page 299 and “Keyword: Exclude” on page 239.

An example of using MapColumn and VirtualColumn with an output column name for the VirtualColumn is:

```
VIRTUALCOLUMN~STANDARD_BENEFIT~JCCLML_CONSTANT,PRODUCT_TYPE_SH~'SH'
VIRTUALCOLUMN~STANDARD_BENEFIT~JCCLML_CONSTANT,PRODUCT_TYPE_CO~'CO'
O
O
O
MAPCOLUMN~OUTTABLEA~PRODUCT_TYPE_SH ~OUTPUTA
MAPCOLUMN~OUTTABLEB~PRODUCT_TYPE_CO ~OUTPUTB
```

MapColumn and the Tuxedo Field ID

The field_id is defined in the Tuxedo FML32 buffer for the source column, and that same id will be used for all of the MapColumns that are based on a single source column, regardless of what name they are given in the MapColumn.

Keyword: MapExclude

MapExclude suppresses output from a specific source column to a specific target table and column. MapExclude supports using a column as a filter, but suppressing it in the output. Note that, if the column is not desired in the output and not used in a filter, it is not necessary to define the column with mapcolumn and, if the column is not defined with mapcolumn, it is not necessary to exclude it with mapexclude.[1]

If a source table is to be excluded for all targets, the keyword Exclude may be used instead or the target table can be avoided by not defining it with MapTable.

Note that tables or columns cannot be excluded before being defined.

See “Logging and Performance” on page 258, as well as the other sections referenced there. See also “Keyword: Exclude” on page 239.

Syntax

MapExclude~<map table name>~<target column name>

Parameters

<map table name>. name of the maptable definition

<target column name>. name of the column to exclude. Note that this is a required parameter.[2]

Example

If, in the example given for mapcolumn, customer_type is another column in the source table and is used to filter which rows go to each of the target tables, it might not be desired in the target table as a data column. The syntax for excluding it from the commercial table is

MapExclude~commercial~customer_type

1. The one caveat is that, by default, Table automatically defines one target table structure. Attention is required to make certain that you are specifying what you intended.
2. Since MapExclude does not have source column name as a parameter, it does not have the wrinkle with VirtualColumns and whether they are re-named that MapColumn has.

Keyword: MapKey

Each target table must have a unique primary key (unless it is an insert only table). No column in the key can be null. For a target table that is to support update, the combination of columns must be able to uniquely identify a specific row in the target table. If no set of columns is sufficient, the originating_dbkey will be required. If all columns in the table are required to uniquely identify the row, an alternate mechanism is required to do anything besides inserts and/or deletes. Note also that none of the columns used in the key can be null.

In early versions of the Loader, the keyword Primary Key was used to define the primary key. The keyword Primary Key is source table specific. The keyword MapKey is target specific and, therefore, is required to support writing a single input row to multiple target rows.[1]

Before a column can be used in a MapKey statement, it must be defined in a MapColumn statement. MapKey can be specified many times, but the final columns that are used as the primary key are a superset of all specifications.

See “Logging and Performance” on page 258, as well as the other sections referenced there. See also “Keywords: Column and Primary Key” on page 235 and “Keyword: Primary Key” on page 281.

Syntax

MapKey~<map table name>~<target column name> \
[,<target column name>[,<target column name>[,<target column name[...]]]]

Parameters

<map table name>. map table name. The target table key is defined in terms of the map table statement. If there is more than one map table statement, care is required to avoid unintended discrepancies in the key definitions. It is not the intent that the Loader support mapping different source rows to the same target using different keys.

1. If you are using none of the features that require target specific keywords, the keyword Primary Key will continue to work.

<target column name>. name of a column in the target that is part of the key. This parameter can be repeated as many times as is relevant.[1]

Examples

```
MapKey~commercial~customer_id
MapKey~po_line~po_number,line_number
```

Note that the columns used in the key must also be defined as MapColumns. Note also that the second example could be written as

```
MapKey~po_line~po_number
MapKey~po_line~line_number
```

Keyword: MapResult

The MapResult keyword supports data transforms.

See also the chapter on "Schema and Data Transforms" on page 489, "Keyword: FilterMap" on page 242, and "Controls for the Filter Database" on page 461. See "Examples of Data Transforms with MapResult" on page 550 for examples of using MapResult.

Syntax

MapResult~<MapTable name>~<column name>~<sql expression>

Parameters

<MapTable Name>. The table name in the target database.

<Column Name>. The column name in the target database.

1. Since MapKey does not have source column name as a parameter, it does not have the wrinkle with VirtualColumns and re-naming that MapColumn has.

<SQL Expression>. The expression to determine the value given to the target column. The result of the expression determines what value to give the column in the target database. The columns included in the expression refer to the source database.[1]

The SQL expression can be any standard SQL that returns a single value of a single data type. The SQL expression can be any standard SQL, an Rdb built-in function, or a user-defined function. Evaluation of the SQL expression is performed in the FilterMap database and user-defined functions must be defined in that database.[2]

A function defined in the FilterMap database can reference any object that is also stored in the FilterMap database (given that the Loader process has sufficient privilege). A user-defined function may reference Rdb external functions. Any SQL expression can also include constants and other parameters.

While any of the multiple uses of MapResult has value, it is executing a function in the FilterMap database combined with the ability to reference a table stored in the FilterMap database that addresses the greatest number of the translation needs that Loader users have raised.

Caution: Any expression that references a table in the FilterMap database, must explicitly name the FilterMap database, as in

```
MapResult~people~state_name~                                    \
(select state_name from filter_db.states where state_code = state)
```

In the example, MapResult is the keyword, people is the table in the target, state_name is the column in the target, and the backslash is the continuation character (used in this case because the full syntax wouldn't fit well without a line wrap). The SQL expression is on the second line. The portion to notice for this caution is 'filter_db.'. Filter_db is a database alias that the

1. See also the keywords Table and MapTable (and Column and Map Column) for how to define the source and target columns.
2. See "New Use for the FilterMap Database" on page 270.

Loader uses at runtime to attach to the FilterMap database. The table, states, is included in the FilterMap database.

Caution: In the example just given, note the parentheses. What's inside the parentheses is a sub-select expression. There is a select embedded in the MapResult processing.

See the Loader kit for a comparison of the statement above to encapsulating the SQL expression in a user-defined function and storing the function in the FilterMap database.

Examples

Illustrative examples are shown in the following chart. These examples use the folllowing column names.

Source Columns for Employees	Target columns for Employee
last_name	company_name
first_name	employee
begin_date	name
department	begin_year
company_name	department_name
employee_number	others ...
others ...	

These examples are for illustrative purposes only. All expressions[1] begin with

```
MapResult~Employee~
```

TABLE 4. Examples for MapResult

Rest of the expression MapResult~Employee~	Effect
company_name~'NewCorp'	applies the value 'NewCorp' to the company_name for all rows.
company_name~department	writes the value in department (in the source) as the value for company_name (in the target).
employee~cast(employee_number as char(12))	uses Rdb's built-in function cast to change the datatype of employee number to 12 character text, whatever the datatype is in the source.
name~trim(last_name)	trims blanks from last name. See discussion of trim and nulls on page 497.
begin_year~extract(YEAR from begin_date)	writes the year from the date stamp for begin_date to begin_year in the target.
name~concat(last_name,',',first_name)	writes to name in the target the combination last name, first name.
department_name~jcc$code2text(department)	states that an example user-supplied function, jcc$code2text, defined in the FilterMap database and a reference table also defined there, must use the code column in the table to lookup and return the text to write to department_name.

1. In this expression, Map Result is the keyword. Employee is the target table. The next part of the expression (company_name or employee or name or begin_year or department_name is a column in the target table. The SQL expression, when it uses column names, refers to the source database.

New Use for the FilterMap Database

The original purpose of the FilterMap database was to provide an Rdb database for analyzing the SQL provided as a user-defined filter on the rows to be written to the Loader target. With the introduction of MapResult, the same database can be used to store user-defined functions and tables to support those functions.[1]

Costs of Using the MapResult Transforms

When using MapResult, you will need to be alert to the demands that you make on the system.

Power and Complexity

The data transform capacity in the Loader is now powerful and flexible. Using this power, of course, requires careful attention to the complexities that may be inherent in your architecture.

The first two of the examples here suggest some of the issues to consider and all of the examples point to the value of encapsulating the complexity and defining it once.

Best Practices

The SQL expression can, of course, be much more complex than those shown in the examples. For ease of maintenance and enhanced readability of the Control File, JCC recommends encapsulating complex expressions in stored functions and calling the necessary function with MapResult, as in

```
MapResult~Employee~department_name~jcc$code2text(department)
```

1. While it is possible to use any database accessible on the system for the MapResult keyword, nothing other than the FilterMap database should be used without good reason. In particular, using the source database should only be done with care and a thorough understanding of the implications. It is conceivable that MapResult processing on the source could be blocked by the Continuous LogMiner when confronting a full mailbox, yielding an undectable deadlock.
See also “Keyword: FilterMap” on page 242.

where MapResult is the keyword, Employee is the target table, department_name (which might be 20 or 30 characters of text or whatever else makes sense[1]) is the column name in the target, jcc$code2text is the name given to the function, and department (which might be a four character code or anything else defined) is the source column passed as a parameter to the function.

Storing the logic for particularly complex functions also avoids byte limits for the MapResult keyword.

The examples provided with the kit show several instances of stored functions which encapsulate the MapResult logic so that the logic can be reused without recoding.

See also "Controls for the Filter Database" on page 461 and "Data Transforms" on page 497.

Keyword: MapTable

MapTable is the target specific version of the keyword Table. MapTable supports writing a single input row to multiple target rows. Use of MapTable opens the way for increased flexibility and features.

MapTable is not required in the Control File, unless there is to be more than one target table for a source table. Using the Table keyword is the equivalent of defining the source table *and* defining one target table with MapTable. See "Automatic Generation of MapTable" on page 290.

The Loader supports up to 63 uses of the MapTable keyword for a given source table with a different target table name for each use.

MapTable must be followed by a set of MapColumn keywords. Each column to be written to the target or used in a filter must be defined. (A column used in a filter, although defined, can be excluded as a data column using mapexclude.)

1. The exact data type is defined by the data type returned by the function used.

See “Logging and Performance” on page 258, as well as the other sections referenced there. See also “Keyword: Table” on page 286.

Syntax

MapTable~<source table name>~<map table name>[,<target table rename>] \
[~<actions>[~<options>]]

Parameters

<source table name>. Name of the table in the source database. Note that the Table keyword definition must also exist.

<map table name>. Unique name given for this mapping of source table to target table. Other map keywords build on this mapping.

<target table rename> ***optional*****.** The target table, by default, is named the same as the source table. It can be re-named through use of the target table name in the Table keyword or through the target table rename parameter of this (MapTable) keyword.

Target table name is the table name in the target database. Note that multiple (up to 63) MapTable keywords -- with different target table names -- can be included for the same source database table.[1]

<actions> ***optional*****.** See the definition of the actions supported in the description of the Table keyword, “<actions> optional” on page 287.

<options> ***optional*****.** See the definition of the options supported in the description of the Table keyword, “<options> optional” on page 289. Only the [NO]ignore_delete_EOS option is applicable for the target.

1. For use with an Oracle target, target table rename may be required to rename tables or columns to coincide with Oracle’s thirty character name limit. Also note that the target table rename can be composed of the schema name and table name, provided that the combination does not exceed the thirty character limit. Alternately, if a single schema is used, it can be specified within Oracle and eliminate the need to specify it with the keyword MapTable.

Example

MapTable~customer~commercial_customer,commercial_customer

Interaction of MapTable and Table Keywords

For the MapTable keyword to be valid requires that the source table be defined for the LogMiner options file and for the Loader Control File. See “Keyword: Table” on page 286.

Using the table keyword is a shortcut to defining a MapTable. The MapTable so defined will automatically inherit all of the columns, virtual columns, excluded columns, and primary keys defined for the source table. It will not inherit filters. (See “Keyword: FilterMap” on page 242.) The target table so defined will have the name of the source table, unless the optional parameter for specifying the target table rename gives it a different name.

The automatic definition of one target table is completely compatible with uses in previous versions of the Loader. MapTable must be used if there is to be more than one target table.

Excluding the source table also excludes the MapTable. See also “Keyword: Exclude” on page 239 and “Keyword: MapExclude” on page 264 for the distinction.

A different list of actions in the MapTable keyword from the one in Table for the same source table is possible. The same set of alternatives exist. See “<actions> optional” on page 287.

Options supported for the MapTable keyword are limited to those that have meaning for the target, namely only [NO]ignore_delete_EOS. See “<options> optional” on page 289.

MapTable and Sort Order

See “Sort Order and MapTables” on page 284.

Keyword: Operator

The keyword OPERATOR can be used to set one or more operator classes to receive failure messages. The default is central. Any number of classes can be specified in a comma separated list or ALL may be specified.

OPCOM messages generated by the license and command line validation routines are generated before the Control File is processed.

Syntax

OPERATOR~ALL|<operator class>[,<operator class>[,...]]

Parameters

ALL. Use all operator classes.

<operator class>. The operator classes are

Operator Class cards	
central	the default
cluster	
devices	
disks	
license	
network	
security	
tapes	
oper1 oper2 ... oper12	

Example

OPERATOR~central, oper1

Keyword: Output

The output keyword specifies the kind of target data store the Loader will be maintaining. Additional parameters are appropriate for different output types.

If the output keyword is not specified, an output type of Rdb will be assumed. If the output keyword is specified, the output type must be explicitly given.

The output keyword should be specified *before* any metadata definitions for source or target. See *"Statement Ordering" on page 224*.

Syntax

OUTPUT~<output type>[~<synchronous>[~<output target> \
[~<message contents>[~<output conversion>]]]

Parameters

<output type>. The following output types are supported for specifying the Loader target. Additional control of the Loader output is specified through the additional parameters.

- API: The Loader calls customer-supplied API routines to perform its work.
- FILE: The target is a disk file on the local machine
- JDBC: The target is a remote database accessed via a class4 JDBC driver.
- OCI: The target is an Oracle database.
- Rdb: The target is an Rdb database. (This is the default.)
- TUXEDO: the target is a Tuxedo application.
- Kafka: The target is a Kafka messaging system.[1]

<synchronous> ***optional*.** The optional keyword SYNCH or ASYNCH may be added. SYNCH is the default. ASYNCH is only supported for Tuxedo applications.

1. The Kafka Option is separately licensed and documented and additional comments relevant to the Kafka Option are not included here.

<output target> ***optional.*** Specifies the name of the target. This name is interpreted differently depending on the output type. For Rdb, it will be an Rdb database. For OCI (Oracle), it will be the OCI TNSnames entry.[1] For a file, it will be the file name. For an API, it will be a shareable image name. The <output target> for JDBC is the JDBC driver specific reference to the database. The <output target> for a TUXEDO application is the qspace used for tpenqueue calls. It is not used for the tpcall calls of the Tuxedo interface.

<message contents> ***optional.*** This parameter indicates the format that the message should take. Possible values are:

- RECORD causes the Loader to send one record for each record in the LogMiner output. This is the default and is required for Rdb, OCI, and JDBC targets.
- TRANSACTION[2] is useful for XML output formats. It groups all rows for a single transaction in a single XML document.[3]

<output conversion> ***optional.*** This is a comma-separated list of options that specifies how the output record is to be configured.[4]

- TEXT: Convert input numeric datatypes to strings for output
- TEXT_SUBSTITUTION: Replace defined message format tags with record/txn values
- XML: Generate an XML document. Required for the output type API.
- TRIM: Remove trailing white space characters from string data for targets other than Rdb. This trims SPACE(CHR(32)), TAB(CHR(9)) and LINEFEED (CHR(10)).[5]

---

1. The Oracle 11.2 interface for OpenVMS requires inclusionof the fullTNSs name specification in the Output keyword and in the dump checkpoint procedures, while the 10.2 interface does not.
2. If you have large transactions, the XML for output by transaction can grow large and use noticeable amounts of memory because the entire XML document is built in memory before it is output. Ten rows in a transaction will not, generally, be a problem. but 10,000 rows will be cumbersome. Adjust your memory allocation - pagefile quota, working set,and so forth, appropriately.
3. Use of "~transaction" was supported with JDBC targets, originally. With the substantial improvement in performance with the re-write for Version 3.2, support for message contents of transaction with JDBC was eliminated. See "Output Keyword" on page 150.
4. See also "Keyword: MapResult" on page 266.

- ESCAPE: For character string data, replace XML reserved characters with specified tags and replace any non-printable characters with "&#xx;" notation
- HEADER: (only valid for FILE, API, JDBC, or Tuxedo) For XML, causes the header documented in the XML chapter to be placed at the beginning of each document. For Tuxedo, causes the pseudo header fields to be included in each FML32 packet.

Note: Providing XML is sufficient to indicate all of the conversions, except HEADER.

Examples

Different examples for different circumstances.

Rdb Output. output~rdb~synch~target_db~record

Oracle Output . output~oci~synch~oracle_9_db~record

The TNSNAMES.ORA file contains the following.

```
oracle_9_db.jcc.com =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(host = zeus)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = oracle)
    )
  )
```

API Output.

```
output~api~synch~api_shr~transaction~xml,header
```

File Output.

5. For Rdb targets, TRIM has no affect. For other targets, see “Comparing Character Data” on page 135. The section, as written, is specifc to Oracle. However, it has wide applicability.

output~file~synch~jcc_xml_output~transaction~xml,header[1]

JDBC Output.

```
output~jdbc~synch~                                    \
jdbc:sqlserver://kong:1433;DatabaseName=JCCLoader
```

or

```
output~jdbc~synch~                                                  \
jdbc.rdbThin://atlas:1701/loader_regression_test_jdbc~record
```

See “Data Types and Details with JDBC Targets” on page 162 for discussion of the optional parameter “message contents” when used with JDBC targets.

Tuxedo Output.

```
output~tuxedo~synch~tuxedo_target~record~header
```

Keyword: Output_failure

The output_failure keyword defines what happens in the case of output failure. With it you answer the questions: How long a pause indicates a stall? How many stalls constitute a failure?

Syntax

OUTPUT_FAILURE~<timeout seconds>~<message retry attempts>

Parameters

<timeout seconds>. is the number of seconds to wait on output before requesting a re-send. The default is 1.

1. When the output type is FILE, the output target must be a logical name that points to a directory. This logical name will be used to create the output file name as follows where YYYYMMDDhhmmss is the year, month, day, hour, minute, and second when the file is opened.

```
<output target directory>:<LoaderName>_YYYMMDDhhmmss.jcc_logminer_loader
```

This parameter is only used with API output; with the other targets, it is ignored. This parameter is passed to the API and the API is responsible for issuing the re-send request.

<retry attempts>. is the number of re-sends to attempt before failing. The default is 10. The minimum is 1. Unlike <timeout seconds>, this parameter has value with all target types.

Depending on the target, it may be advisable to set a delay between the retry attempts, when output_failue is used. See “Retry Delay” on page 434 for more on this setting.

Deciding that the retry default is too low and raising it dramatically can have unexpected consequences. For example, the following line

```
output_failure~5~1000
```

coupled with an unknown privilege problem, caused the Loader to try the initial connection to the database by the CLML process a thousand times. It failed one thousand times or was on the way to doing so. Obviously, that took a bit of CPU time. The person attempting to run the Loader lost patience and aborted. Thus, the re-try setting masked the real issue. Eventually, Loader support identified the privilege issue, but also recommended dropping the re-try count somewhat.

Keyword: Parallel

The Loader can run Loader processes in parallel. These parallel processes are referred to as threads and the parallel operation as multi-threading. The Loader does not, however, use OpenVMS P-threads.

Multi-threading uses multiple simultaneous processes to move your data to the target efficiently. Each process represents a completely independent thread sending data to the target.

- Threads may be configured to synchronize with each other so that they avoid buried updates.
- Tuning of a multi-threaded use of the Loader is possible by adjusting the Parallel and Threads keywords in the Control File.
- Multi-threading can be dynamic. The number of threads can change in response to the workload. Alternately, multi-threading can be controlled manually.

See also “Parallelism and Loader Threads” on page 383 and “I/O Management” on page 393 for additional discussion of tuning and performance.

Requirements

Using the multi-threaded option, requires certain settings.

- The parallel keyword must be specified in the Control File.
- The thread keyword may be specified in the Control File.
- The Input keyword must *not* be set to ASYNCH. (SYNC, which is required for multi-threaded operation, is the default.)
- The input type for the Input keyword must be IPC (mailbox).
- The Input_failure keyword may be used to set timeouts.[1]
- To support parallel operations, each Loader thread must maintain its own high-water record in the target database. (If checkpointing to a file, then each Loader thread must contain two entries.)[2]

Syntax

PARALLEL~<minimum threads>~<maximum threads>[~CONSTRAINED | AUTOMATIC | UNCONSTRAINED]

Parameters

<minimum threads>. the minimum number of Loader threads that will be started. There is no default. The minimum must be stated and must be between one and the number set for the maximum, inclusive.

Once set, minimum threads can be changed while the Loader is running with the command

```
JCC_CLML_minimum_threads <LoaderName> <new minimum>
```

1. The Input_Failure keyword may also be used with single threaded Loaders.
2. The requirement for highwater information for *each* thread caused a format change from version 1 of the Loader to version 2. Procedures are included in the kit to upgrade pre-2.0 high water tables.

<maximum threads>. the maximum number of Loader threads that can be started. There is no default. The maximum must be stated and must be between the minimum and thirty-two, inclusive.

Once set, maximum threads can be changed while the Loader is running with the command

```
JCC_CLML_maximum_threads <LoaderName> <new maximum>
```

<consistency mode> ***optional.*** The consistency mode determines the level of attention given to avoiding burying update transactions. There are three options. Constrained is the default.

- CONSTRAINED: If the threading mode is set to constrained, the Loader will use OpenVMS locks (in a Loader lock tree) to synchronize updates to the target. The result is that updates to the same row in successive transactions will be sent to the target in the same order that they were applied in the source database
- UNCONSTRAINED: With consistency mode of UNCONSTRAINED, each Loader thread will write to the output target immediately with *no synchronization between the threads*. Unconstrained operation supports insert only applications with no surprises. However, writes may occur out of order. If some of the writes are updates, an update from a later transaction may be buried by an earlier version of the data that happens to get written later. Similarly, deletes can lead to surprises with a consistency mode of UNCONSTRAINED.

  JCC recommends using this mode only when the table is, essentially, being audited (table actions “insert,noupdate,nodelete”). It is normal when using that table mode to materialize one or several auditing columns such as Commit Timestamp and Loader Sequence Number. (See “Keyword: VirtualColumn” on page 299.)
- AUTOMATIC: Consistency mode AUTOMATIC uses consistency mode UNCONSTRAINED for tables that are set to insert only ("insert,noupdate,nodelete", where updates cannot be buried because there are none) and uses consistency mode CONSTRAINED for all other tables.

Keyword: Primary Key

The keywords Primary Key and Column use the same syntax and are discussed together on page 235.

It should be noted that Column and Primary Key are both keywords that are source table specific. The important thing about defining the primary key is to be able to uniquely identify a row in the target. Although the keyword Primary Key is upwardly compatible, use of the target specific keyword MapKey is recommended. See “Keyword: MapKey” on page 265.

The tool for generating the Metadata Control File and the default statements for mapping to the target will “guess” at the primary key, but some editing may be required. The “guess” relies on the primary key constraint, if there is one, and chooses among any unique indices, if there is not a primary key constraint. If neither approach produces a candidate, the Loader tool which generates the Metadata Control File “assumes” that the db-key approach will be used.

The primary key must be unique in the target table.[1] The primary key may consist of several columns.[2] No column in the primary key may be null. Each column of the primary key will require a separate primary key keyword.

The primary key keyword is not consistent with a table that is marked as being maintained via the originating dbkey mechanism. However, it is consistent with using an identity attribute. See “Identifying Rows in the Target” on page 39.

Limits

Null values for primary keys (or portions of primary keys) are not supported by the Loader.[3] Unless there is an alternate candidate primary key,[4] it will be necessary to use the originating dbkey mechanism.

Having all columns as part of the primary key is incompatible with updates of rows in the target. You must use NOUPDATE if you specify that each column in the

1. Generally, the key is the same in both source and target. If you are having the Loader write source table data to multiple target tables, see “Keyword: MapKey” on page 265. If you are using advanced mapping to combine data from several rows, tables, or databases, see “Performance Implications” on page 504. For complex mappings, see also “Schema and Data Transforms” on page 489.
2. Defining the primary key as all columns in a table is incompatible with actions other than insert and delete. See “Identifying Rows in the Target” on page 39.
3. NULLs in primary keys are not supported by the SQL standard and are not supported for declared primary keys in Rdb.
4. See “Identity Attribute” on page 40 for an alternative.

table is part of the primary key. (Alternately, you can use the originating_dbkey approach.)

Keyword: Sort

The sort keyword is used to manage the order in which records are sent to the output stream. It requires additional CPU time on the host, but the algorithm is replacement-insertion, a fast in-place sort.

Sorting by_record can substantially improve performance for Oracle targets. The default is by_record.[1]

JCC recommends using this keyword only in very special circumstances. The *default behavior* of the sort has been tuned following the experience of a large number of Loader customers with varied application challenges.

Syntax

SORT~<sort type>

Parameter

<sort type>. There are two sort types supported and an option for disabling sorting:

- BY_RECORD is useful for OCI connections and environments that do updates as a sequence of read-delete-insert. BY_RECORD sort order is table name, Loader sequence number,[2] and action.[3] Sorting by table name causes the Loader to process all rows for one table in a transaction prior to processing rows for another table. This adds efficiency to the OCI interface by reducing the number of messages required to support dynamic SQL. Sorting by action places delete before insert which addresses issues with the read-delete-insert sequence.

1. Prior to V 2.0, the default was BY_TRANSACTION.
2. The sequence number is assigned by the Loader when it executes.
3. The action is modify (‘M’) or delete (‘D’).

- BY_TRANSACTION causes the sort order to be Loader sequence number, action, table name. Sort type of BY_TRANSACTION provides correct results if the application, as some do, performs updates as a combination of the operations of delete and insert.
- DISABLE disables sorting. However, even with DISABLE, deletes precede updates to avoid anomalies. DISABLE is the default.

Sort Order and MapTables

If the sort order selected is BY_RECORD, all input rows for a given table are processed for the first MapTable definition. Then, all input rows for the table are processed for the next MapTable definition.

Otherwise (BY_TRANSACTION or DISABLE), each input row is processed for all associated MapTable definitions before moving on to the next input row.

For example, assume two source tables, X and Y, that each have two rows updated in transaction A such that the rows arrive as X1, X2, Y1, Y2. If source table X has maptables Xa and Xb defined and source table Y has maptables Ya and Yb defined, the following will result from the sort selection.

TABLE 5. Sort Order and MapTables

By_record	Other (By_transaction and None)
Xa 1	Xa 1
Xa 2	Xb 1
Xb 1	Xa 2
Xb 2	Xb 2
Ya 1	Ya 1
Ya 2	Yb 1
Yb 1	Ya 2
Yb 2	Yb 2

Sort Order and TableOrder

If the TableOrder keyword[1] is used, it can change the sort order. With the default values for TableOrder,[2] the example in Table 5, "Sort Order and MapTables," on page 284 does not change. If Table Order is used to request that table Y be presented before table X, the TableOrder keyword might be as shown in the example that follows. The Sort Order for our example would then change to that shown in Table 6, "Sort Order and MapTables with TableOrder," on page 285.

```
TableOrder~Y~999
```

This syntax requests that Y arrive before X because X still has the default value of 1000. The result is shown in the chart to follow.

TABLE 6. Sort Order and MapTables with TableOrder

Default TableOrder		TableOrder Used to Specify Y Sorts Before X	
By_record	Other (By_transaction and None)	By_record	Other (By_transaction and None)
Xa 1	Xa 1	Ya 1	Ya 1
Xa 2	Xb 1	Ya 2	Yb 1
Xb 1	Xa 2	Yb 1	Ya 2
Xb 2	Xb 2	Yb 2	Yb 2
Ya 1	Ya 1	Xa 1	Xa 1
Ya 2	Yb 1	Xa 2	Xb 1
Yb 1	Ya 2	Xb 1	Xa 2
Yb 2	Yb 2	Xb 2	Xb 2

Logging

When the sort keyword is encountered, a message is emitted to the log. An example of such a message is

1. See "Keyword: TableOrder" on page 290.
2. The default for each source table is 1000.


```
8-APR-2003 10:40:02.84 20202A54 ||5 REGTESTTUX
Default sort type set to BY_RECORD
```

Keyword: Table

The table keyword specifies a table in the source database and determines the version of that table. To review the handling of columns within the table, see “Keywords: Column and Primary Key” on page 235 and “Keyword: VirtualColumn” on page 299.

Table is a source specific keyword. To review alternatives for defining the target table rows, see “Logging and Performance” on page 258 and the other sections referenced there, most particularly “Keyword: MapTable” on page 271. See also “Automatic Generation of MapTable” on page 290. If the maptable concept is not employed, then the keyword table implicitly defines a map table and the actions on the map table.

Syntax

TABLE~<table name>[,<map table rename>]~<record version> \
[~<actions>][~<options>]

See “Example of a Control File Portion” on page 220.

Parameters

<table name>. Specifies the name of the table in the source database.

<map table rename> ***optional.*** This optional parameter is preceded by a comma, when used. By default a MapTable is created to define the target table. By default, the MapTable and the target table are given the name of the source table. If this parameter is included, the MapTable and the target table are given the name specified by this parameter. If more than one target table is to be mapped from this source or if a more complex mapping is desired, you must specifically utilize the MapTable and related keywords. See “Keyword: MapTable” on page 271.

This parameter may be required with an Oracle database because Oracle names are limited to 30 characters and Rdb names may be 31. Also, for Oracle, the output table name may include both schema name and table name (schema.table), provided the total is thirty characters or less.[1]

<record version>. The record version of the table is defined in the source database. To get the version number of a table from Rdb, use:

```
select rdb$relation_name,rdb$max_version from rdb$relations
```

The initial value of each record version is set correctly by the procedure that generates the metadata portion of the Control File. However, if the metadata changes, you need a way of checking it.

<actions> ***optional.*** This is a comma-separated list of values that indicate the disposition of a row in the output database. These values may all be negated. Possible values are

- INSERT causes the Loader, when processing an input record marked as modify, to insert rows into the target if a row with the primary key does not already exist. If this action is negated (NOINSERT), the Loader does not attempt to perform the insert operation.
- UPDATE causes the Loader, when processing an input record marked as modify, to update rows in the target database if they exist. If this option is negated (NOUPDATE), no update attempt is tried. If the Insert option was also selected with a negated update (INSERT,NOUPDATE), for each transaction that changes in the source a row will be inserted into the target database. The result will be a history of changes to each row.

  The row to update is identified by the primary key. If it is possible to change a primary key column of a table, the originating dbkey technique must be used.
- DELETE causes the Loader to process delete records and attempt to delete rows from the target database. If the row does not exist, no exception will be reported by default. If this option is negated (NODELETE), delete records from the LogMiner will be treated as modify operations.

  Note that there are different implications for NoDelete in the LogMiner and in the Loader.

  1.If you specify NoDelete to LogMiner

1. Alternately, if you are using a single schema, you can set the default schema within Oracle and, therefore, eliminate the need to specify it with the Table or MapTable keywords.

(rmu/unload/after/include=nodelete), the LogMiner does not pass on the delete records.

2. If you specify NoDelete to the Loader

(table~<table>~<version>~nodelete), you get an update.

- Named composite actions provide single word action definitions that specify a combination of the [no]insert, [no]update, and [no]delete choices. The named collections of actions and their meanings are shown in the table.

Name	Action Equivalents	Used for
Replicate	Insert,Update, Delete	Replication This is the DEFAULT.
Rollup	Insert,Update,NoDelete	Replication, plus maintaining the last version of a deleted row, generally with the action as a materialized column to serve as a logical delete. For example TABLE~customer~ROLLUP will maintain, in the target, the list of all customer records, even those that have been deleted on the source.
Audit	Insert,NoUpdate,NoDelete	Maintaining all versions of a row, generally with materialized data to indicate the action, identify when the row was changed or deleted and who did it. For example TABLE~payment~AUDIT will maintain, in the target, all the different states that a row has traversed. When auditing, it is, generally, wise to materialize a timestamp and session user and, perhaps, the action. Audit with materialized columns enables tracing the temporal history of a row. See “Keyword: VirtualColumn” on page 299.

<options> ***optional*.** Options allows you to specify one of a variety of situations that you may have for a table. Possible options are:

- Originating_dbkey: This parameter will force the use of the column ORIGINATING_DBKEY when no primary key exists. (There is no negation for this parameter.) Here is an example

  ```
  TABLE~CUSTOMER_ASSIGN_PKG_DETAIL~1~Replicate      \
  ~Originating_dbkey
  ```

  The ROLLUP option and ORIGINATING_DBKEY options are in conflict and cannot be specified simultaneously for the same table.
- [NO]ignore_delete_eos: If a target database is changed other than by the Loader, there may be a delete record in the LogMiner output for which there is no record in the target. In this case, SQL will emit an EOS (End-of-stream) signal. IGNORE_DELETE_EOS causes the Loader to ignore an EOS signal from SQL. This is the default behavior. If the option NOIGNORE_DELETE_EOS is included, an EOS will cause the Loader to fail.
- [NO]length: Length causes the Loader to accept the length of the input record and null bit vector for the source LogMiner data varying from that which is specified in the Control File.
- [NO]maptable: NOmaptable suppresses the automatic definition of maptable, mapcolumn, and mapkey keywords to reflect the source database definition in target database terms. The default is maptable.

Tables Processed

The Loader will process only those tables that are specified in the Control File.

Exceptions

There are exceptions that you can generate in the Control File.

Duplicate Definitions. if a duplicate table definition (a second definition for a table already defined) is found in the Control File, rather than failing or confusing the table definition, the Loader gives an exception message. The following exception message is an example.

```
%dba_parse_init_file invalid record - table already defined in
file: SCP_PROD_DATA_DEFINITIONS.INI
line: TABLE~CREDIT_EXTENSION~1~Replicate
```



```
%DBA_E-INV_INIT_RECORD, Invalid initialization record encountered.
```

Table with Too Many Columns. When the Loader works with a table with many columns and/or columns with long names, the Loader may generate dynamic SQL beyond the abilities of the Rdb dynamic SQL parser. This is not, however, relevant in most environments. See “Exception” on page 237 for a discussion of the limits.

Automatic Generation of MapTable

The table keyword is source specific. However, the table keyword, by default, acts as if a single MapTable had been defined between the source and the target. Automatic generation of MapTable can be disabled with the NOMAPTABLE option. See “<options> optional” on page 289. See also “<map table rename> optional” on page 286, as well as “Logging and Performance” on page 258 and the sections referenced there.

Keyword: TableOrder

In some contexts, it is important to control the order of presentation of rows from a transaction to the target. It may be important, for example, to send the commit record last or the data for table B before the data for table A.

Examples of when receipt order may matter include:

- Targets such that a downstream process controls the transaction integrity[1]
- Targets with triggers that may fire incorrectly if rows arrive in an unanticipated order

The Loader provides this level of control over order.

Sorting JCCLML$commit Last

The Loader is set to sort the virtual table jcclml$commit last, such that all records for a transaction are sent prior to the commit record for the transaction. This is the

1. For example, XML targets or Tuxedo targets. If the eventual target of a Loader JDBC target is not a database that protects transactional integrity, this will apply as well.

default behavior. See “Control of Table Presentation Order” on page 291 for how to modify the default.

Sorting commits last occurs whether the sort order is set to by_record or by_transaction. When sort order is disabled, the Loader does not explicitly send jcclml$commit last. It does, however, happen implicitly that jcclml$commit is last, as the commit record is always the last record passed by the LogMiner for a transaction.

The default behavior can be overridden with the TableOrder keyword.

Control of Table Presentation Order

The Loader can be directed to sort by table within a transaction. The mechanism for providing this control is the TableOrder keyword.

Syntax

The syntax is

```
TableOrder~<source table>|jcclml$commit~<ordinal position>
```

<source table>. Names the source table. Note that the Table must be defined before this keyword is encountered in the Control File. TableOrder is implemented for sorting of input table rows. If multiple targets are defined for a single input table, see Table 6, “Sort Order and MapTables with TableOrder,” on page 285.

Note that it is also possible to set the ordinal position for jcclml$commit. Doing so enables overriding the default of presenting the commit last.

<ordinal position>. An unsigned four-byte integer in the range 1 through 4,294,967,294. The default is 1,000. Zero is reserved for future use and 4,295,967,295 is reserved for jcclml$commit.

Example

For a short example, if you have two tables X and Y, the default TableOrder is 1000. Therefore, to indicate that Y should be presented first, use

```
TableOrder~Y~999
```


An example that shows the complete interaction and impact of the keywords MapTable, TableOrder, and Sort is included in “Sort Order and TableOrder” on page 285.

Keyword: Thread

See “Keyword: Parallel” on page 279 for a discussion of multi-threading. The Thread keyword establishes how rapidly the number of threads changes following triggering events. (See “Automatic and Dynamic Adjustments to the Number of Threads” on page 384 for further discussion of the multi-threading operation.)

The thread keyword is not required even when the parallel keyword is specified.

Syntax

THREAD~STARTUP~<delay seconds>

THREAD~SHUTDOWN~<delay seconds>

Parameters

STARTUP and SHUTDOWN. STARTUP is used to determine thread startup characteristics. SHUTDOWN is used to determine when threads are shut down. One Thread keyword is used for each.

<delay seconds>. number of seconds to delay prior to performing the action. The default is 30.00.

Note that delay seconds may be specified as a floating point number.

Keyword: Tuxedo

Essentially, there is a set of Tuxedo keywords. They are reflected here one at a time.

- Tuxedo~FieldHeader
- Tuxedo~MaxPacketSize
- Tuxedo~NullValue
- Tuxedo~<Output format>
- Tuxedo~<Output type>
- Tuxedo~WSNADDR

Keyword: Tuxedo~FieldHeader

This keyword, if needed, must come before any table definitions.

This command specifies the Tuxedo FML32 field header files for the Tuxedo application to which the Loader will attach and send data. There will, usually, be at least two instances of this keyword in the Control File. The first file will specify fields materialized by the loader, including some in the FML32 header. The second will specify fields associated with database columns.

The field header files are created using the Tuxedo mkfldhdr32 procedure; see the Tuxedo FML documentation for further information.

The jcc_tool_com:create_log_miner_tux_field_def.com procedure, supplied with the Loader kit, will generate a file which can be used as the input to the Tuxedo mkfldhdr32 procedure. The file is generated based on the metadata of the source database. The output of the Tuxedo mkfldhdr32 procedure is a C header (.h) file. The file name is specified in the Tuxedomkfldhdr32 command and is compiled into the Tuxedo application.

The jcc_tool_source:loader_virtual_columns.tbl defines the Virtual Columns for the Loader and can be processed using the Tuxedo mkfldhdr32 command. The only change that should be made to this file is the “base” value.

The .h file supplied in the kit can be used if no changes to the .tbl file are required.

The .h file must be specified since the Loader always sends materialized (virtual) columns, such as the count of rows in the FML32 buffer. The .h file is specified with the Tuxedo~FieldHeader command.

Syntax

Tuxedo~FieldHeader~<filename>

Parameter

<filename>. One of the filenames created with the Tuxedo mkfldhdr32 procedure.

Note that there can be more than one instance of this command. If a header is generated, it will require specification of the materialized columns. Therefore, if a header is generated, at least one of these files is required. See “Keyword: VirtualColumn” on page 299.

Keyword: Tuxedo~MaxPacketSize

Use this to specify the maximum packet size to be sent to Tuxedo. You may adjust this to the characteristics of your network and target computers.

Syntax

Tuxedo~MaxPacketSize~<value>

Parameter

<value>. Value defines the maximum number of bytes that will be allocated for FML32 buffers. Therefore, this also represents the largest TCP/IP packet that will be sent. To estimate a good size, calculate the size of the largest row (including materialized columns) and make the MaxPacket Size at least this large. If you want to represent a number of rows in a packet, you will need to multiply the row size by the number of rows to get MaxPacketSize.

The optimum value will depend on a combination of factors, including network characteristics and operating system capabilities on the system supporting the target Tuxedo application.

Keyword: Tuxedo~NullValue

Handling of null values with a Tuxedo target relies on an architecture which uses each FML32 buffer for a specific table. The FML32 buffer has a single repetition of header information and as many repetitions as required of row information.

Within this format, there are two options for indicating nulls. Note that you do not need this keyword, if there are no nulls in the source database.

Syntax

Tuxedo~NullValues~<null representation>

Parameter

There are two options for the null representation.

NullColumnList. Create a comma-separated list of the column names which are null. If no column is null, the NullColumnList will be set to "".

SynthesizeColumn. Create a new integer column for each column. Call the new column <ColumnName>_ni and give it one of the values of 0 (not null) or -1 (null).

Note that if the column is not null, the null indicator field may or may not be included in the FML32 buffer. If a null is encountered later in the repeating group, Tuxedo will set the null indicator for previous rows.

Keyword: Tuxedo~<Output Format>

Select DOMAIN or WORKSTATION mode with this keyword.

Syntax

Tuxedo~<output format>

Parameter

<output format>. There are two options.

- WORKSTATION specifies Workstation mode for the Tuxedo work.
- DOMAIN specifies Domain mode for the Tuxedo work.

Keyword: Tuxedo~<Output Type>

Use this to specify whether the target is a Tuxedo service or a Tuxedo queue and, optionally, to indicate a single target.

Syntax

Tuxedo~<output type>[~<single target name>]

Tuxedo~CALL|QUEUE[~<single target name>]

Parameters

<output type>. May be either a TPCall or a TPQueue

- Call causes the Loader to Tpcall a named Tuxedo service and pass the data packet in the FML32 buffer.
- Queue causes the Loader to Tpenqueue the FML packet in a named Tuxedo queue.

<single target name> ***optional.*** Call the same service or enqueue to the same queue for all output record types. If <single target name> is not specified, the table name will be used as the service or queue name. If the table name is 'renamed',[1] then the rename value is used as the queue or service name.

1. Rename refers to using the Table keyword to specify a target table of a different name than the source table's name.

Keyword: Tuxedo~Transaction

Tuxedo~transaction enables control by the Loader of the distributed XA transaction of the Tuxedo servers. This option is only available for Tuxedo targets and only if the call interface is used.

Syntax

```
Tuxedo~transaction
```

If this syntax is used without the Tuxedo call interface, it will have no effect. If this syntax is not used, the Loader will follow the default behavior.

By default the TPNOTRAN flag is passed to the Tuxedo tpcall routine. The TPNOTRAN disassociates the Tuxedo service that the Loader calls from the Tuxedo transaction that the Loader starts.[1]

The default behavior produces a higher throughput in some systems, but requires special programming in the servers.

Keyword: Tuxedo~WSNADDR

Tuxedo~WSNADDR is appropriate only when using the Tuxedo workstation client. See “Keyword: Tuxedo~<Output Format>” on page 295.

Tuxedo~WSNADDR is used to specify up to eight addresses.

The Loader uses each address specified for each Tuxedo attach in round robin fashion. Thus, if one Tuxedo server fails, the Loader will, in response to the failure, rollback the current Tuxedo transaction, detach from Tuxedo, attach to an alternate server, and attempt again to send the commit interval. When using multiple threads, each Loader thread will connect to the next WSNADDR that is specified. When the

1. The default behavior is to use the TPNOTRAN flag. The default excludes the Loader from the global XA transaction that includes the called services. Using Tuxedo~transaction disables that flag with the result that the Loader is included in the global XA transaction to the target such that the transaction includes the entire commit interval.

last value is used the selection wraps to the first value specified. This mechanism provides a modest amount of load balancing and fault tolerance for the Loader.

Syntax

Tuxedo~WSNADDR~<value>

Parameter

Value. Defines a WSNADDR value. Use the syntax repeatedly to define up to eight values. The Loader threads will cycle through the list of WSN addresses, circularly, as threads start and stop.

Keyword: Validation

This keyword allows specification of the username and password to be used in some targets. Targets that use validation may include Oracle, JDBC, and Tuxedo. Also, the validation keyword can also be used to provide username and password for a remote TCPIP connection for an Rdb database. If the validation keyword is specified, the database attach statement is augmented with "user <username> using <password>" syntax when attaching to the remote database.[1]

Syntax

VALIDATION~<username>~<password>

Parameters

<username>. Supply a string containing the username to be passed to the target.

<password>. Supply a string containing the password to be sent to the target.

1. JCC strongly recommends including the Control File in the log (logging~initialization). Beginning with Version 3.3.1, the password is not shown with the validation keyword in the log.

Example

validation~system~manager

Keyword: VirtualColumn

Virtual columns are columns that provide additional information that applies to the row (such as transaction commit time, username, and many others.) Virtual columns do not exist in the source database, but can be materialized for the target database.

Virtual (or "materialized") columns are optional, but are useful, or even critical, in some applications. Virtual columns have many uses in monitoring the full application, as the possibilities include numerous timestamps. Virtual columns also have some uses in tuning the target due to randomvalue and modvalue options. The Control File may contain all, some, or none of the possible virtual column definitions.

General Syntax

VIRTUALCOLUMN~<table name>|*~<virtual column name> \
[,<output column name>]

Special Syntax

Other virtual (or materialized) columns require additional parameters. For these, the syntax is

VIRTUALCOLUMN~<table name>|*~JCCLML_CONSTANT \
[,<target column name>]='<value>'

or

VIRTUALCOLUMN~<table name>|*~RANDOMVALUE \
[,<output column name>]~<high value>[~<low value>]

or

VIRTUALCOLUMN~<table name>|*~MODVALUE \
[,<output column name>] ~<mod value>

Parameters

<table name>|*. Set the source table name or indicate, with the asterisk that all tables should have this VirtualColumn added. See “Wildcarding” on page 303.

<virtual column name>. The table “Virtual Columns” on page 301 shows the possible values for the virtual column name parameter. The table also includes the required datatypes and comments. In addition, the table notes which virtual columns require additional parameters.

<output column name> optional. The column name in the target can be different than the virtual column name. Particularly for some of the VirtualColumns that have additional parameters, it may be important to define a given type of VirtualColumn more than once for a given source table. Then, it becomes important to be able to refer to the output column name for the VirtualColumn to be able to distinguish the separate uses of it. The default is to assume the same name as the virtual column name.

Additional Parameters

<value>. the value for JCCLML_CONSTANT. This parameter might be a city name in combining regional databases or any number of other options that meet a need for a constant.

<high value>. the maximum value to return. The value must be a valid signed 4-byte integer. This parameter is used with the RANDOMVALUE virtual column only.

<low value> ***optional.*** the minimum value to return. The value must be a valid signed 4-byte integer. If <low value> is omitted, it defaults to zero. This parameter is used with the RANDOMVALUE virtual column only.

<column name>. the name of the column on which to perform the MOD function. This column must have a non-scaled, integer data type. The column name cannot be itself (MODVALUE.) If the data in the source record is marked as NULL, then this column will also be marked NULL. This parameter is used with the MODVALUE virtual column only.

<mod value>. the value to use in the modulo function. The value must be a valid signed 4-byte integer. This parameter is used with the MODVALUE virtual column only.

.

TABLE 7. Virtual Columns

Virtual Column	Data Type	Comment	Add'l. Param.
LOADER_ SEQUENCE_ NUMBER	8 byte integer	Ordinal sequence number of transactions delivered to the Loader by the LogMiner, preserved number across sessions. Materializing this column in the target indicates, relatively, when the row was changed.	N
ORIGINATING_ DBKEY	8 byte integer except for Rdb (for Rdb, see comment)	Dbkey of the row in the source database. Note that the data type for Rdb is an 8 byte string by default, but a data type of BIGINT is available for Rdb. For other targets, the data type is 8 byte integer.	N
LOADERNAME	31 characters	Loadername used internally	N
LOADER_VERSION	15 characters	Loader assigned version number. For example V02.01.00ev6.	N
LOADER_LINK_ DATE_TIME	date-time	Timestamp for when this version of the Loader was linked.	N
TSN	8 byte integer	Transaction Sequence Number in the source database	N
TRANSACTION_ COMMIT_TIME	date-time	Time committed in the source db	N
TRANSACTION_ START_TIME	date-time	Time started in the source db	N
ACTION	1 character	'M'odify or 'D'elete	N
RECORD_VERSION	2 byte integer	Rdb assigned version of the metadata for the table. Also used in the Control File keyword table.	N
RANDOMVALUE	4 byte integer	Random number for horizontal partitioning or other application use.	Y

TABLE 7. Virtual Columns

Virtual Column	Data Type	Comment	Add'l. Param.
MODVALUE	4 byte integer	Number for horizontal partitioning derived from a mod function. The column name cannot be itself (MODVALUE.) If the data in the source record is marked as NULL, then this column will also be marked NULL.	Y
PARALLELTHREAD	4 byte integer	Number to distinguish a particular thread.	N
TABLE_NAME	31 characters	Source db table name	N
PID	4 byte integer	Process ID for the process that wrote the transaction.	N
TID	4 byte integer	Distributed transaction ID. *Note that it is impossible for the Loader to understand all the resource managers in a distributed transaction. In providing the TID, JCC does not imply that distributed transactions can be reassembled in a reliable or cost-effective manner.*	N
JCCLML_ CONSTANT	varies	Optionally provide a target column name and provide a value.	Y
JCCLML_READ_ TIME	date-time	Time at which a row was read from the input stream. For the original static loading, this is the time at which the record was read from the input file. For continuous loading, it is the time at which the row was read from the Oracle Rdb LogMiner mailbox.	N
JCCLML_AERCP	char(73) or larger	Rdb's AERCP is used for application restart. Each transaction has a unique AERCP value. Use of this VirtualColumn causes the Loader to write the transaction's AERCP to the designated table and column.	

TABLE 7. Virtual Columns

Virtual Column	Data Type	Comment	Add'l. Param.
TRANSMISSION_ DATE_TIME	date-time	Time sent by the Loader to the target Putting this virtual column last makes the value more precise.	N
JCCLML_ USERNAME	char(12) (standard OpenVMS username)	With Version 7.2.1.0, Oracle Rdb added a column to the commit record for the *username of the process that performed the transaction in the source database*. This virtual column definition will capture that username.	N

Virtual Columns and Filters

Using filters with any column — including virtual columns — offers an opportunity to partition the target or to exclude rows from the target. However, the following VirtualColumns cannot be used for filtering:

- PID
- TID
- Transmission_date_time
- RandomValue
- ModValue

See also "Keyword: Filter" on page 240 and "Keyword: FilterMap" on page 242.

Wildcarding

When the Loader Administrator wants a virtual column, for example the transaction commit datetime stamp, to be included in each target table, it can be very useful to have a way of defining that the inclusion is needed in each and every table, instead of having to make a separate statement for each table.

Wildcarding supports doing that by adding to the syntax an asterisk instead of a table name.

A wildcard virtual column directive can use any VirtualColumn. The example of wildcarding below shows some of the possibilities.

```
!
! JCC LML V3.6 Wildcard VirtualColumn
! Add virtual columns to all table definitions
!
VirtualColumn~*~action,jcclml_action
VirtualColumn~*~loader_sequence_number,jcclml_sequence_number
VirtualColumn~*~loadername
VirtualColumn~*~loader_version
VirtualColumn~*~loader_link_date_time
VirtualColumn~*~transaction_commit_time
VirtualColumn~*~transaction_start_time
VirtualColumn~*~jcclml_read_time
VirtualColumn~*~jcclml_aercp
VirtualColumn~*~transmission_date_time
```

Note: VirtualColumn must be defined in the Control File after TABLE definitions and before MAPTABLE definitions.

To be effective the VirtualColumns so defined must also be mapped. To map the wildcarded virtual columns to target columns can also use wildcarding.

```
!
! JCC LML V3.6 Wildcard MapColum
! Add virtual columns to all MapTable definitions
!
MapColumn~*~jcclml_action
MapColumn~*~jcclml_sequence_number,loader_sequence_number
MapColumn~*~loadername
MapColumn~*~loader_version
MapColumn~*~loader_link_date_time
MapColumn~*~transaction_commit_time
MapColumn~*~transaction_start_time
MapColumn~*~jcclml_read_time
MapColumn~*~jcclml_aercp
MapColumn~*~transmission_date_time
```

MapColumn must be defined in the Control File after MapTable definitions and before FilterMap or MapResult definitions.

Example Output for XML with Virtual Columns

This example is formatted for readability, rather than leaving it as one long string.

```
502C310E00000000000000001000004C1XACTSSUBRDB05200201161818493795200010
10200200201161817091700000000000000000000
<?xml version='1.0'?>
<!DOCTYPE pkt SYSTEM 'packet-commented.dtd'>
<pkt>
<trxn tsn='200' strt='2001071816251363592' end='2001071816251415150'>
<row name='CANDIDATES' actn='M'>
<col name='LAST_NAME' type='str' len='14' val='Mustard'/>
<col name='FIRST_NAME' type='str' len='10' val='Teeman'/>
<col name='MIDDLE_INITIAL' type='str' len='1' val='H'/>
<col name='CANDIDATE_STATUS' type='str' len='255' val='This guy looks like a winner!'
<col name='LOADER_SEQUENCE_NUMBER' type='num' val='1'/>
<col name='ORIGINATING_DBKEY' type='num' val='22517998188691460'/>
<col name='LOADERNAME' type='str' len='8' val='SUBRDB05'/>
<col name='LOADER_VERSION' type='str' len='9' val='D01.02.00'/>
<col name='LOADER_LINK_DATE_TIME' type='dt' val='2002011618170917000'/>
<col name='TSN' type='num' val='512'/>
<col name='TRANSACTION_COMMIT_TIME' type='dt' val='2001071816251415150'/>
<col name='TRANSACTION_START_TIME' type='dt' val='2001071816251363592'/>
<col name='ACTION' type='str' len='1' val='M'/>
<col name='RECORD_VERSION' type='num' val='1'/>
<col name='TRANSMISSION_DATE_TIME' type='dt' val='2002011618184937952'/>
</row>
</trxn>
</pkt>
```

FIGURE 5. Example of Virtual Columns in XML

Unsigned Values for Materialized Data

See “Unsigned Values for Materialized Data” on page 474.

Keyword: VirtualTable

The LogMiner supplies commit records as part of the output. It is possible to direct the Loader to write these commits to the target in a virtual table.

For notes on controlling the order of receipt of this table within the transaction, see “Keyword: TableOrder” on page 290.

Syntax

VirtualTable~JCCLML$COMMIT[,<map table name>]

Parameters

JCCLML$COMMIT. The required specification of the source.

<map table name>. Optional specification of the map table name to use in mapping to the target.

Columns

The table has no columns by default. The columns that are possible for the virtual table are most of the virtual columns. Any column included must, however, be explicitly added. At least one virtual column is required.

Virtual columns not available for use with the VirtualTable are those that have no meaning in the virtual table context, namely

- Originating_dbkey
- Action
- Record_version

Example

```
virtualtable~JCCLML$COMMIT
virtualcolumn~JCCLML$COMMIT~transaction_commit_time
```

Keyword: XML

The XML keyword is not required when using XML and is extraneous for any other target. The defaults (“Defaults” on page 307) show how the XML keyword is interpreted by default. The syntax (“Syntax” on page 307) shows additional options. The XML keyword provides the opportunity for greater control of XML output.

The Output keyword must be specified, as API, before the XML keyword is used.

Defaults

The XML keyword is generally not used, even when using XML as a Loader target. For XML output, it is as if you had specified the XML keyword as follows.

```
XML~DTD~packet-commented.dtd
XML~packet
XMP~Transaction~End, Start, TSN
XML~Table~Action
XML~Column~ValueOrNull, Type
XML~header~Prolog,doctype
XML~NULL~explicit
```

These illustrate the defaults and do not have to be specified. The last two are the most likely to be used to change the defaults.

Syntax

XML~<element>~[<attribute>[,<attribute>[,<attribute>]]]

Parameters

<element>. The element options are

- DTD
- Packet
- Transaction
- Table

- Column
- Header
- NULL

Some of these have attributes.

<attribute> for DTD. The attribute for the DTD element provides the name of the DTD (Document Type Definition) file. You can change the default by using

```
XML~DTD~<your own file name>
```

<attribute> for Packet. By default packet has no attribute. Providing TSNCount as an attribute for packet will include the TSNCount for the packet. The TSNCount is the number of Rdb transactions. For this, use the syntax

```
XML~packet~TSNCount
```

<attributes> for Transaction. By default, the XML includes the TSN, start, and end attributes for each transaction. See “Transaction (trxn)” on page 203 for an explanation of these attributes. With the XML keyword, these attributes, plus RowCount, can be included or excluded by listing them or not in a comma separated list of attributes. For example, the default is XML~transaction~End, Start, TSN. Using XML~transaction without attributes would suppress the end, start, and TSN. Using XML~transaction~end would show only the end. Using XML~transaction~end, start, TSN, rowcount would cause all of the usual information, plus the row count to be included for each transaction.

<attributes> for table. By default, the Loader indicates in the XML the action for each table.“Schema and Data Transforms” on page 489 The actions that might be indicated are ‘D’ or ‘M’ (delete or modify/insert).

Note that this is not the keyword to specify which of these actions should be published by the Loader. That specification is done with the keyword Table or the keyword MapTable.

For example,

```
XML~table~action
```

<attributes> for column. By default, the Loader indicates in the XML, for each column, the value of the column or that the column is null and the data type of the column, plus additional attributes.

Including XML~column~<attributes> provides a means for you to leave off the comma separated list any of the default attributes and also provides the option of adding the attribute Key. If Key is specified, the Loader will include in the XML the word key for any column that is part of the key used.

Example

Here are two examples.

Example 1. shows output if the keyword is not used and the defaults apply.

```
XML~column~ValueOrNull,type
```

Example 2. shows output that involves all of the extra attributes for each element.

```
XML~column~ValueOrNull,type,key
```

<attributes> for Header. By default, the Loader includes a header for XML output. The header has two parts: one called Prolog and one called DocType.

Prolog consists of "<?xml version='1.0'?>" which specifies the version of the XML standard that is used for this XML document.

DocType consists of "<!DOCTYPE pkt SYSTEM 'packet-commented.dtd'>".[1]

Because some application architects do not want this header, the Loader has been modified to include a keyword for the XML header. The syntax has four options.

TABLE 8. XML Header Options

Syntax	Result
`XML~Header~Prolog,DocType`	The default includes both parts of the header.
`XML~Header~`	Includes neither part of the default header.

1. The JCC LogMiner Loader's DTD's can be found in JCC_TOOL_ROOT:[SOURCE].

TABLE 8. XML Header Options

Syntax	Result
`XML~Header~Prolog`	Include the part of the default header called prolog, but not the part called DocType
`XML~Header~DocType`	Include the part of the default header called DocType, but not the part called Prolog

<attributes> for NULL. By default, the Loader uses explicit to indicate that columns with NULL values will be shown. There are actually two possible values.

```
XML~NULL~explicit|implicit
```

Implicit causes the Loader to not list null columns.

However, there are actually three possible results, depending on how the column itself is defined. The keyword MapColumn includes the option of defining <value if null>. If a value definition for null is specified, that value replaces NULL. In which case, for this column, implicit and explicit have the same effect.

Summary

The Loader is controlled with keywords in the Control File.[1] Each keyword has been described in this chapter. To provide a handy review, here is the syntax for each.

TABLE 9. Summary of Keywords in the Control File

API~CONNECT~<routine name>[~p2 text string>[~<p3 text string>[~<p4 text string>]]]
API~SEND~<routine name>
API~DISCONNECT~<routine name>
CHECKPOINT~<commit interval>[~<checkpoint stream type>[~<synchronous> \ [~<checkpoint target>]]]
COLUMN|PRIMARY KEY~<table name>~<column name>[,<output_column_name>] \ ~<position> ~<length>~<scale> ~<type>~<sub type> *
COLUMN~<table name>~<column name>[,<output_column_name>]~<position> \ ~<BLOB IGNORE>|<BLOB LOG>|<IGNORE> *
DATE_FORMAT~<date format>
EXCLUDE~<table>[~<column>] *
FILTER~INCLUDE | EXCLUDE~<table name>~<column name>=<value> *
FILTERMAP~<map table name>~[where]<SQL where clause on the row>
INCLUDE_FILE~<filename> *
INPUT~<input type>[~<synchronous>[~<input source>]]
INPUT_FAILURE~<timeout seconds>
JDBC~<element>~<attribute>
LOADERNAME~<text string>
LOGGING~OUTPUT | INPUT | STATISTICS~<logging options> *
LOGGING~TRACE|LOCKING|INITIALIZATION *
MAPCOLUMN~<map table name>~<source column name>[,<target column rename>] \ [~<value if null>]
MAPEXCLUDE~<map table name>~<target column name>
MAPKEY~<map table name>~<target column name>[~<target column name> \ [,<target column name>[,<target column name>][,...]]]]

1. Logical names offer additional control. These are described in relevant sections throughout the documentation.

TABLE 9. Summary of Keywords in the Control File

MAPRESULT~<maptable name in target>~<column name in target>~<sql expression>
MAPTABLE~<map table name>~<source table name>[,<target table name>] \ [~<action list>[~<options list>]]
OPERATOR~ALL|<operator class>[,<operator class>[,...]]
OUTPUT~<output type>[~<synchronous>[~<output target>[~<message contents> \ [~<output conversion>]]]
OUTPUT_FAILURE~<timeout seconds>~<message retry attempts>
SORT~<sort type>
TABLE~<table name>[,<output table name>]~<record version> \ ~[[NO]INSERT,[NO]UPDATE,[NO]DELETE,REPLICATE|ROLLUP|AUDIT] \ ~[ORIGINATING_DBKEY,[NO]IGNORE_DELETE_EOS,[NO]LENGTH], \ [NO]MAPTABLE] *
TABLEORDER~<source table>|jcclml$commit~<ordinal position>
THREAD~STARTUP|SHUTDOWN~<delay seconds>
TUXEDO~FIELDHEADER~<file name> *
TUXEDO~MAXPACKETSIZE~<value>
TUXEDO~NULLVALUE~<null representation>
TUXEDO~DOMAIN|WORKSPACE
TUXEDO~CALL|QUEUE[~<single target name>]
TUXEDO~TRANSACTION
TUXEDO~WSNADDR~<value> *
VALIDATION~<username>~<password>
VIRTUALCOLUMN~<table name>~<virtual column name>[,<output column name>] \ [~<additional parameters for some virtual columns>] *
VIRTUALTABLE~JCCLML$COMMIT[,<map table name>]
XML~<element>[~<attributes>] *

** Asterisked items may be repeated multiple times.*

CHAPTER 14

Monitoring an Ongoing Loader Operation

The JCC LogMiner Loader includes tools for analyzing its operation. These tools are also useful for identifying some behaviors in related areas, such as the target or the network.

The statistics monitor, the log files, operator messages, the locking diagnostic tool, and tools for revealing checkpoint, database, and loader information are powerful resources. Use them to

- Test your application
- Analyze performance
- Find bottlenecks
- See the results of changes that you make in Loader options
- Answer resource questions
- Monitor for problems
- Satisfy curiosity
- See the LogMiner's progress through AIJ files

Do not expect to use all of the tools, all of the time. Many of these provide so much information that they are suitable for development and problem solving, but not suitable for routine production. Others, such as the tardiness threshold provide important early warning of production issues.

Online Statistics Monitor

While the JCC LogMiner Loader is running, it creates and maintains an OpenVMS global section in which it posts aggregated information about its progress. This information is reported by Loader threads when the Loader shuts down, but may also be displayed continuously by means of a special monitoring tool.

This section discusses setting up the monitoring tool. Sections to follow give examples of each of the types of output and discuss in detail how to use the results. The section “Statistics Output with Other Tools” on page 352 discusses other aspects of the statistics monitor and how the output can be combined with other tools to produce graphs and other useful results.

Interactive and Batch

The online monitor is designed to provide and interactive, ongoing, realtime view of Loader operations. However, it can also be run in batch. The default settings will be different for interactive and batch.

Start the Monitor

Before invoking the statistics monitor, ensure that the procedure JCC_LML_USER described in “User Procedures” on page 105 has been run to establish context.

The monitor routine may, then, be invoked with the simple DCL command:

```
$ jcc_lml_statistics <Loader family name>  -
     [<report interval>]                    -
     [<report type>]                        -
     [<tardiness threshold>                 -
     [operator class]]
```

Loader Family Name. The Loader family name typed here should match the Loader family name that has been specified. (The Loader family name may be specified with either the LOADERNAME keyword in the Control File or the logical name JCC_LOGMINER_LOADER_NAME. See also "Keyword: Loadername" on page 245.)

Report Interval *optional.* The report interval is a number specifying the number of seconds to wait between reports. The default value is 3 seconds.

Report Type *optional.* Report type specifies one of the report types: Full, Detailed, Brief, State, CSV (CommaSeparatedValues), and T4. These values may be abbreviated to 'F', 'D', 'B', 'S', 'C' and 'T'. The default is full. Examples are included in the following.

- The full report provides a comprehensive analysis of the total pattern of work done by the loader during the reporting interval and also includes some summary statistics. See "Full Report Example" on page 332.
- The detailed report is a subset of the full report. The detailed report is a single screen of information. See "Detail Report Example" on page 326.
- The brief report includes a highly summarized activity report. See "Brief Report Example" on page 338.
- The state report provides detailed information on the state (status) of each thread. See "State Report Examples" on page 339.
- The CSV report provides some of the fields of the detailed report in a comma separated format suitable for loading into a spreadsheet or database. See "Comma Separated Values Report Example" on page 342 and see "Putting Statistics in a Database" on page 565 for an explanation of how to load the CSV data into an Rdb database.
- The T4 report output is a version of comma separated values output that is specifically designed to meet the calling standards of the Total Timeline Tracking Tool ("T4") developed by OpenVMS Engineering.

Tardiness Threshold *optional.* The Loader can never get ahead of the source database. Updates to the target will always lag because the information is not available until after the commit occurs in the source. The size of this lag is dependent on a number of things. Things that will affect this lag are: the rate of update of the source database, the performance of the target, the number of Loader threads in the family, the timeout interval set for communicating with the LogMiner, the commit interval for the Loader session, and the reporting interval selected.

How much the target updates lag will vary. What makes an acceptable lag will differ with the application and circumstances. For example, one application uses the Loader to replicate the transaction processing database to many remote query databases. The lag time is one to two seconds with 500 transactions per second, committed, captured by the LogMiner, and written by the Loader. That lag time is regarded as well within the requirements.

If unexpected lags occur, it may be desirable to alert someone. In our example, a lag of twenty seconds would probably indicate a problem that requires intervention. (The network might be having difficulty or a target computer might be down.)

The tardiness threshold is the number of seconds of acceptable lag. If the lag equals or exceeds the tardiness threshold, an opcom message is displayed.

There is also a message to report catching up. If a trailing message is issued, the caught up message will be issued when the Loader is no longer beyond the tardiness threshold. Once issued, the caught up message will not be issued again, unless another tardiness message is issued.

See “Choosing the Tardiness Indicator” on page 458 for additional information and examples.

Operator Class. Operator class supports Administrator definition of where to send opcom messages. The default is Central. See “Operator Classes and OPCOM Messages” on page 379.

Display and Scroll

For the online monitor, whichever report type is chosen, the default operation is to provide a screen of data and update it in place as the values change.[1] This is the display mode of the statistics monitor.

The default for operations run in batch is scroll mode. In scroll mode, the monitor writes a continuous stream of data, one display after another, scrolling older data up.

When running interactively, it is possible to change between scroll and display with the interactive control.

1. This statistics mode is only available with the JCC LogMiner Loader version 3.6 and later.

Interactive Control

While running JCC_LML_STATISTICS interactively, there are keys that can be pressed to get additional information or change the behavior of the Monitor. The keys that are defined and their impact are shown in the table and are also shown online by pressing '?' or 'h'.[1]

TABLE 1. Control Keys for Interactive Statistics Display

Key	Description
?	Display help message with all of the keys defined.
h	
b	Switch to the 'b'rief display.
f	Switch to the 'f'ull display (the default).
d	Switch to the 'd'etail display.
s	Switch to the 's'tates display.
r	Show the rates, rather than totals. Available only for the full display.
t	Show the totals, rather than rates. Available only for the full display.
ctrl t	Print runtime information.
ctrl m	Switch between display mode and scroll mode.
ctrl w	If in display mode, clear the screen.
ctrl c	Exit the jcc_lml_statistics utility.
ctrl y	
ctrl z	

Exceptions

When the Loader statistics program (jcc_lml_statistics) starts, if the Loader (for which it is to report statistics) is not running, the program will emit a message to sys$output that it is waiting for the named Loader to start. As soon as the named Loader family has created the statistics global section, the Loader statistics program will begin to report information.

1. Lower case keys are shown in the table. The uppercase values have the same impact.

This behavior locks the command prompt until the Loader starts or the timeout is met.

For interactive sessions the timeout default is 3 seconds. For batch sessions the timeout default is 60 seconds. In either case, it is possible to modify the default number of seconds for the monitor to wait for a running Loader session to start. To change the wait, define the logical name provided to the number of seconds that you would like.

```
$ define JCC_LOGMINER_LOADER_STAT_WAIT_SECONDS <number of seconds>
```

Exiting

The statistics program will detect when the Loader family stops running and will exit.

The statistics program will also exit if you type <control-Z> or <control-C> or <control-Y> at your keyboard. These are hard exits.

Control-t and Statistics Running Time

Control-t generates output which is shown at the bottom of the screen. The information supplied is

- Start time of the Loader session (LoaderStart)
- Timestamp of the first commit row processed by the Loader family (1stCommit)
- How long the Loader session has been running (UpTime, the current wall time minus LoaderStart)
- Current AIJ sequence number for the most recent AERCP that the Loader has written to the target

Control-t works with each of the statistics types. This example is from the detail display:

```
[Detail]
LoaderStart: 30-MAR-2007 08:28:47.33                UpTime:     0 00:50:25.66
 1st Commit: 30-MAR-2007 08:28:39.80          Cur AERCP AIJ: 3
```

Loader Processing Cycle

While it is doing its work, a Loader thread will progress through input, processing, and output stages. The processing can be analyzed in much greater detail than this, but, first, consider an overview.[1]

Input. Input requires reading the input mailbox that is loaded by the LogMiner. The mailbox is filled with one record per row that was updated during a transaction, plus one commit record. The Loader always reads at least one full transaction.

When the checkpoint keyword is used to increase the commit interval, the Loader reads a full commit interval's worth of data, unless there is a read timeout set and there is no activity on the database. See "No Work Transactions and Checkpoint Intervals" on page 389.

As rows are read from the mailbox, the Loader will optionally request Loader-dbkey locks on each of the rows. This request is made when the Loader session is configured to be constrained.

Processing. Once the Loader has completed reading all of the records in a commit interval, it will sort the resulting data. For replicating to a database, the default sort order of by_record (the default) is particularly useful for performance. (See "Keyword: Sort" on page 274.) Sorting by_record is required for Tuxedo output.

Processing also may include synchronization of the threads and conversion of the data into formats requested through the Control File.

Output. Once the sorting is completed, *and* the Loader has received *all* of the dbkey locks that it requested, the Loader sends data to the target. If the target is a database or Tuxedo, the Loader commits the transaction, at this point. If the Loader target is Rdb or OCI, the Loader checkpoints to the database as part of this transaction. If the checkpoint type is LML_INTERNAL (all other Loader targets), the Loader writes the checkpoint to its own local highwater file.

1. For the breakdown used for analyzing latency, see "Loader Latency Reporting" on page 322. For the very detailed display of what each thread is doing at the moment the statistics are captured, see "The Monitor and Loader Threads" on page 320. Each of these are reflected in the Detail and Full reports. Latency is also reflected in the CSV and T4 statistics.

Repeat. After all work is complete, the Loader thread will compete, again, against the other Loader threads for access to the mail box in order to read more information from the LogMiner. Access to the mailbox is controlled via the Loader Sequence Number lock and Loaders will stall waiting for exclusive access to this lock. The Lock holder is the only thread that will read from the mailbox.

The Monitor and Loader Threads

With multiple threads at work, following the action of the Loader requires additional information. Each thread may be in any of several states. A number of the report types include information on thread states that rely on single character codes. These are displayed in two lines. The top line indicates the thread number and the second indicates the thread state. The complete list of states and the symbols that represent them is shown in the chart to follow.

TABLE 2. Thread State Codes

State	Meaning	Used for
R	thread is waiting to get the 'read' lock (the one that controls access to the input mailbox)	Read Phase
z	thread has read lock, but is waiting for input	
<	thread is currently reading from the input mailbox	
*	there is data in-flight and the thread is not currently waiting for any Loader resource (may also occur during write phase)	
W	thread is waiting for dbkey locks to be granted prior to the write phase	Write Phase
s	thread is currently sorting the buffered input data	
>	thread is currently writing to the target	
*	there is data in-flight and the thread is not currently waiting for any Loader resource (may also occur during reads)	
T	thread is waiting as part of a realtime throttle (See “Realtime” on page 480.)	
t	thread is waiting as part of a fixed throttle (See “Fixed” on page 481.)	
d	thread is waiting as part of a retry delay (See “Retry Delay” on page 434.)	
|	thread has reached lock threshold and is waiting for WriteLock in protected write mode (See “Stall States and Statistics Display” on page 406.)	Stall states related to processing extremely large transactions
]	thread has requested WriteLock in concurrent write mode and is blocked by a process waiting for WriteLock in protected write mode	
.	(period) Loader is active, but not in any of the above states	
	(space) the specified thread is not running. (This will only happen if there is a thread that is displayed to the right of the space and if the Loader is configured with dynamic startup/shutdown of threads. This is a characteristic of threads starting or stopping.)	

The State report type provides much more information on the threads. The codes used for that are more extensive, but still need some interpretation. See “State

Report Examples" on page 339 for a discussion of the State report type and "Thread Details for the Statistics Monitor" on page 595 for an interpretation of the states reported.

Total Time and Threads

Sometimes, the statistics for elapsed time and CPU time, on the full display, can seem wrong. If there were only one Loader thread, the elapsed and CPU times could not exceed the wall time that the Loader has been active. However, in a multi-threaded family, these numbers represent the aggregation across all members of that family. Accordingly, it is quite possible, and indeed probable, that these numbers can substantially exceed the wall time that the Loader family has been instantiated.

Loader Latency Reporting

Sometimes, it is difficult to understand where a delay occurs. The Loader cannot load data before it is available from the LogMiner and the LogMiner cannot write it to the mailbox until a transaction is committed. The latency may also be in the network or target. See also "Target Latency Reflected in the Log" on page 365.

The statistics show the latency in the overall Loader application. Using these statistics to determine where the latency is occurring is an important component of tuning the use of the Loader.

For the detail screen (and in the first section of the full report), look for the latency numbers in the lower right. In the brief and CSV output, look for the latency numbers to follow the throughput numbers. See "Detail Report" on page 326 for an example.

The Detail and Full statistics displays shows the CLM latency, the total LML latency, and latency numbers for six phases of the Loader.

The Detail screen (and the first section of the Full report) also shows the input and output trailing numbers immediately below the timestamp for those columns.

The State report shows the latency for each thread.

Summary Latency Numbers

CLM and LML latency numbers are shown in units of time. (See also "Latency Scale" on page 324.) Output Latency, which is included in the T4 and CSV data, is shown in seconds.

CLM. CLM latency is available in the T4 data as '[.CLM]Ave CLM Latency'.[1] The CLM Latency is calculated as the Loader read time minus the transaction commit time. If the database is only open on a single node, then, this number is both accurate and intuitive. If the database is written to by processes on more than one OpenVMS node and the system timestamps are not properly synchronized (or if system time is altered), the value can be misleading. (Should the value be calculated to be negative, it will be set to zero to minimize the variance from reality.)

LML. The total Loader latency is available in the T4 data as '[.LML]Ave Total Latency'. This number is the average (for the current statistics reporting interval) of the number of seconds required for the Loader threads to read the data from the CLM mailbox and write the data and the checkpoint. The LML latency number includes all six of the detail phases ("Detail Latency Numbers" on page 323).

Output Latency. Output latency continues to be reflected in the CSV data and in the T4 data as '[LML]Ave Output Latency.' It is the average (for the current statistics reporting interval) of the number of seconds the Loader threads spent writing data to the target and checkpointing. Output Latency includes sync, cnvt, trgt, and ckpt latencies.

Detail Latency Numbers

The latencies for the detailed phases of Loader operation are shown as percentages to reduce confusion about the individual values as well as to offer a more precise value.

Inpt. Input Latency is included in the T4 statistics as '[.LML]Ave Input Latency,' This is calculated as the average (for the current statistics reporting interval) of the number of seconds the Loader threads spent reading data or stalled waiting for data from the CLM mailbox.

1. Raw T4 output is cumbersome to read. See the following for additional information and examples.

Sort. Sort Latency is included in the T4 data as '[.LML]Ave Sort Latency.' It is the latency that is attributed to the Loader sort operation. In versions prior to 3.1, this value was included in the overall latency (LML), but not included in either the input or output latencies.

Sync. Lock Synchronization Latency is included in the T4 data as '[.LML]Ave Synch Latency.' It is the latency attributed to the Loader parallel thread locking which coordinates the sequence of updates to the target. In versions prior to 3.1, it was included in the output latency value, but not displayed separately.

Cnvt. Conversion Latency is included in the T4 data as '[.LML]Ave Convert Latency.' It is the latency attributed to the Loader during the output phase as it converts the input data into the data format and style that the Control File specifies for the target. In versions prior to 3.1, it was included in the output latency value, but not displayed separately.

Trgt. Target Latency is included in the T4 data as '[.LML]Ave Target Latency.' It is the latency attributed to the target data store. It includes only the latency encountered by calls into the target data store software. In versions prior to 3.1, it was included in the output latency value, but not displayed separately.

Ckpt. Checkpoint Latency is included in the T4 data as '[.LML]Ave Checkpoint Latency.' It is the latency attributed to writing the checkpoint information to the user declared data store. In versions prior to 3.1, it was included in the output latency value, but not displayed separately.

Latency Scale

The numbers that are displayed as times, CLM and LML, are included as seconds in the T4 and CSV output. For the Detail, State, and Full displays, numbers that are shown as amounts of time are adjusted to use units that are tuned to the value and make the best use of available space. See "Time Scale Conversion" on page 325 for details.

When the conversion occurs is determined by the latency scale. The default latency scale is 1.0. The default can be changed using the statistics options. See "Statistics Options" on page 352.

Tardiness Definition

The logical name JCC_LOGMINER_LOADER_STAT_TARDY_FIELD accepts only the latency fields: Total Latency, Input Latency, and Output Latency. See “Operator Classes and Tardiness Messages” on page 457 for more information.

Statistics for Filtered Rows

The statistics also show information on filtered rows. See “Keyword: Filter” on page 231 and “Keyword: FilterMap” on page 233.

Filter shows both input and output values. Filtering at the input level reduces the processing required. Rows are filtered at the input, rather than output, level if either

- a Filter on a source table excludes the row
- the row is excluded from all targets through FilterMaps for each MapTable

Find the filter statistics on the detail screen about half way down and reflected on both the input and output sides. The full report reflects filtering in the first section that looks like the detail screen. CSV and Brief report types do not show filtering.

Time Scale Conversion

The statistics display shows a scale conversion from seconds to other units when other time units can reflect a more meaningful result or greater precision, depending on the circumstances.

Seconds is the default unit and seconds will be displayed without the ‘s’ to define the units as seconds. By default, the number will be displayed in seconds until one minute is reached, will be displayed in units of minutes between one minute and one hour, and will be displayed in units of hours between one and twenty-four hours and in days beyond 24 hours. If the number is converted to minutes, it will be displayed followed by an ‘m’. If converted to hours, it will be displayed followed by an ‘h’. If converted to days, it will be displayed followed by ‘d’.

Likewise, if the number of seconds is less than 0.01, it is converted to milliseconds and followed by ‘ms’ and, if it is less than 0.00001 seconds, it is converted to microseconds and followed by ‘us’.

Example conversions are shown in the table.

TABLE 3. Example Time Conversions for Statistics Display

Calculated (in seconds, if not noted as otherwise)	Displayed for Readability
28.1h	1.2d
36.4h	1.5d
90.02	1.5m
90.25m	1.5h
0.001	1.0ms
0.0000001	1.0us

See “Detail Report Example” on page 326 for an example. In the example, the trailing times are given in minutes rather than the default seconds. Time scale conversions are also reflected in other report types.

Detail Report Example

The Detail Report provides a screen’s worth of information. The Detail Report has the same layout as the first section of the Full Report. As such, it is a good place to illustrate each of the sections and statistics that are available.

```
Rate:   6.00                      REGTESTJDBR            20-JAN-2019 16:43:29.00
================================================================================
   Input: 20-JAN-2019 16:32:39.71                JDBC: 20-JAN-2019 16:32:39.63
--[Trail:   10.8m]---------------            ---[Trail:   10.8m]---------------
Transactions              125072            Checkpoints                    9312
Records                   503489               Timeout                       13
   Modify                 376404               BufferLimit(   120)            0
   Delete                   2013               NoWork                         0
   Commit                 125072            Records(   3)                489738
Discarded                                      Messages(  N/A )             N/A
   Filtered                    0               Filtered                       0
   Excluded                    0            Failure                           0
   Unknown                     0            Timeout                           0
   Restart                     0         - Current ---------------- Ave/Second -
   NoWork                  13688         Checkpoints            36          6.00
   Heartbeat                   0         Records              1456        242.67
Timeout                     1013         Rate                 7.18%
--- Restart Context ------               - Latency(sec) ------ LML detail ------
M|AIJ#                  10 |             CLM   10.8m |  Inpt   0.9%  Cnvt  49.0%
Q|VBN              349609 |             ------------   Sort   0.0%  Trgt  38.6%
P|TSN              163596 |             LML    2.21    Sync  11.4%  Ckpt   0.1%
 CTSN              163596 |         - Loaders - 0123456789abcdefghij ------------
  LSN              111300 |         - States  - RRRRR RR>R>RRRRRRRRR
```

FIGURE 1. Detail Report

Information in the detail report is compact. Begin at the top of the example and use these notes to understand the information given.

Rate. The rate shown (first line, left) is 6.0. The refresh rate for the statistics is every 6 seconds. This is specified in starting the monitor. (See "Start the Monitor" on page 314.)

Title. REGTESTJDBR is the name of the thread family, the Loader name. (In the example, the title indicates that this is a run of the regression testing that is using the Loader target of JDBC to write to Rdb. The Loadername may be specified with the Loadername keyword in the Control File or with the logical name JCC_Log-Miner_Loader_Loader_Namw.)

Dates. The date on the title line is date/time for the statistics generation. The date for input is the timestamp of the last source transaction read from the Continuous LogMiner mailbox or input file. The date for output is the timestamp of the last source transaction written to the target.

Trailing. These are the lags for input and for output. Input is trailing by over 10.8 minutes (10.8m)[1] and output is trailing as well by the same amount. That is, the Loader is running almost 11 minutes behind. If this were a production environment, these rates might require tuning. Concern or lack of it would be determined by the purpose of the architecture. As is, the regression testing is running on a heavily loaded system, writing multiple target rows per source row. The Loader will eventually catch up because the test will complete. Meanwhile, this illustrates some warning signs to consider.

Transactions. The first line on the input side shows that there have been 125,072 transactions since the Loader session was started. (We'll come back to the numbers below that.)

Checkpoints. The first line on the output side shows that there have been 9,312 checkpoints since the Loader session was started. The detail offered below that shows that there have been 13 timeouts ("Keyword: Input_failure" on page 248),[2] zero bufferlimit ("Interpretation of Lock Conflicts" on page 403) and zero "no

1. Note that this time is shown as minutes, not seconds. See "Time Scale Conversion" on page 325.
2. This number is often zero. That it is greater than zero, shows that there have been checkpoints that occurred due to an input timeout.

work" transactions ("No Work Transactions and Checkpoint Intervals" on page 389). The parentheses by BufferLimit shows average for all the threads of the current input buffer threshold.

Input records. Most of the rest of the left side of the screen shows information on the input records. There were 503,489 of them. Of that, 376,404 were modifies[1], 213 deletes, and 125,072 commit rows.[2] The input statistics continue with information on discarded rows. Listed are:

- Filtered: These would be the records filtered on input, meaning filtered due to use of the Filter keyword or due to use of the MapFilter keyword in such a way that the filtering applies to all targets.
- Excluded: These would be the records excluded on input through use of the keyword Exclude. See "Keyword: Exclude" on page 239.
- Unknown: Number of records from tables not described in the Control File. This number can only exceed zero if logging~input~ignore_unknown_tables is set. (NOignore_unknown_tables is the default.)
- Restart: Count of records sent by the LogMiner to the Loader and ignored by the Loader as part of a restart.[3]
- NoWork: These are the records in transactions that have no records that are to be written to the target. See "No Work Transactions and Checkpoint Intervals" on page 389. See also Table 3, "Logging Keyword Usages and Meanings," on page 257.
- Heartbeat: See "Logical Name Controls for Loader Procedures" on page 463. The heartbeat is not enabled for the Loader family for which we are examining the statistics. Therefore, there were no records written in support of it.

Loader Target Type. On the output side, instead of labeling the column "Output:" the report gives the Loader Target type. In this case, it is JDBC. The end target might be any of a wide range of options.

---

1. Other than commit records, the LogMiner writes only Modify or Delete records. (The Loader performs an "upsert" to handle insert/modify.)
2. Not surprisingly, the number of commit rows is the same as the number of transactions.
3. Because a restart begins with a specified AERCP and sends the rows from that transaction, it is expected that one transaction's worth of rows will be duplicate (and, therefore, discarded) on restart. How many rows are discarded depends on how many are in the transaction.

Output Records. On the output side, there are 489,738 output records shown. (Note that the output records are unlikely to exactly equal the input records, even adjusted by "no work" records and other numbers that you can see. In a busy system, some records will be buffered mid-way between the read and the write. Also, the maptable keyword supports definition of several output rows for one source row.) Details listed are:

- Messages: This indicates the number of messages sent to the target. For database targets it is not applicable. For Tuxedo messages, it is the number of FML32 buffers and, in the API output, it is XML documents. When applicable, the number in parenthesis is the average number of bytes in the message sent to the target.
- Filtered: This is a count of the records filtered with the keyword MapFilter. Note that this count doesn't include rows filtered from all targets and shown on the input side. See "Input records" on page 328 and "Statistics for Filtered Rows" on page 325
- Parenthesis: The 3 in parenthesis after the label 'Records' is the number of distinct tables that the Loader threads have processed.

Output Failure. This is the number of times that the Loader has received a failure from the target.

Output Timeout. This is the number of times that the Loader has received a timeout from the target.

Input Timeout. Timeout (on the input side) is the count of times that the timeout specified in the Control File has been triggered. See "Keyword: Input_failure" on page 239.

Restart Context. The section in the lower left corner supplies the restart context.[1] See "Transactions and Recoverability" on page 35. The M, Q, and P at the beginning of the first three labels indicate that these three items comprise the MQP (MicroQuietPoint). In detail, the labels are

- M|AIJ# The AIJ sequence number is 10.
- Q|VBN Virtual Block Number is 349609.

1. The AIJ, VBN, and TSN are for the most recent MQP, Micro Quiet Point. (See "Quiet Points" on page 101.) The most recent MQP may or may not be in the AIJ that is currently being processed by the Oracle Rdb LogMiner. The most recent MQP is the location in the AIJ files where the LogMiner will start reading on a restart.

- P|TSN The Transaction Sequence Number is 163596.
- CTSN Cutoff Transaction Sequence Number is 163596.
- LSN The Loader Sequence Number is 111300.

Rates. Just above the bottom on the right is the rate information for the data checkpointed to the target in the current interval.

- There have been 36 checkpoints in the current interval, for an average per second of 6.
- There have been 1,456 records processed in the current interval for an average per second of 242.67.
- The target is being updated at a rate of 7.18% of the source update rate.

Latency. Below Rate, the latency figures are given. See “Loader Latency Reporting” on page 322 for a more complete definition of the latency figures, the scale used, and the interpretation.

See “Loader Latency Reporting” on page 322 for more on the meaning and use of the latency numbers.

Threads and States. The final two lines on the right supply labels for up to 32 threads and the state that each is in at the moment that the statistics are collected. When these statistics were collected, thread 2 was writing, and the rest were waiting to get the read lock.

See “The Monitor and Loader Threads” on page 320 for more on the states that can be represented for each thread.

Statistics on the Statistics Monitor. When control-t is pressed, the bottom of the screen shows statistics on the statistics collection itself. These are discussed in “Control-t and Statistics Running Time” on page 318.

The following example shows just the lower lines of another Detail screen example, this time with Control-t display. For purposes of the example, the results of Control-t are shown in red.

Note that this example shows eight (8) threads and that they are all writing to the target.

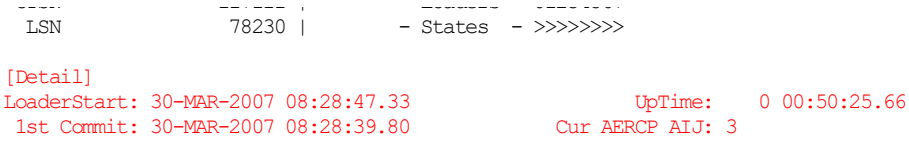


FIGURE 2. Control-t for the Statistics Monitor

Full Report Example

For the Full report the report header is the same as a detail report. The body of the full report details the performance and costs of all portions of this processing cycle. The tail of the report includes a histogram summarizing performance for all commit intervals.

The report follows the stages of the processing cycle (“Loader Processing Cycle” on page 319 and “Loader Latency Reporting” on page 322). The detail is intended to help highlight any delays that occur. Each section has two parts, a summary detailing the performance of the entire Loader session and a “Rate” section which displays the associated information for the particular reporting interval.

A sample Full report is reproduced on the following pages. (This is the report that will be generated when an Rdb or Oracle target is selected.) For convenience, the example is broken into sections and comments are inserted between sections. The full report is generally used in batch mode and inserted into a log. When displayed in the log, it is all run together without the breaks included here for comments.

The first section of the full report is the same style as the detail report.

```
Rate:   6.00                      REGTESTJDBR            20-JAN-2019 16:46:49.72
================================================================================
   Input: 20-JAN-2019 16:32:42.74                 JDBC: 20-JAN-2019 16:32:42.47
--[Trail:    0.00]--------------             ---[Trail:    0.00]---------------
Transactions               129076             Checkpoints                   9620
Records                    519805                Timeout                      37
   Modify                  388639                BufferLimit(   120)          0
   Delete                    2090                NoWork                       0
   Commit                  129076             Records(   3)               505304
Discarded                                       Messages(  N/A )             N/A
   Filtered                     0                Filtered                     0
   Excluded                     0             Failure                         0
   Unknown                      0             Timeout                         0
   Restart                      0         - Current ---------------- Ave/Second -
   NoWork                   14128         Checkpoints             0          0.00
   Heartbeat                    0         Records                 0          0.00
Timeout                      1013         Rate                0.00%
--- Restart Context ------                - Latency(sec) ------ LML detail ------
M|AIJ#                 10 |               CLM    0.00 |  Inpt   0.0%  Cnvt   0.0%
Q|VBN             378537 |               ------------   Sort   0.0%  Trgt   0.0%
P|TSN             167782 |               LML    0.00    Sync   0.0%  Ckpt   0.0%
 CTSN             167782 |         - Loaders - 0123456789abcdefghij ------------
  LSN             114691 |         - States  - WR>>> R>>>RR><>W>>>>
```

FIGURE 3. Full Report - Section 1

The next section provides data on the entire cycle from the read through the write. Note that the total statistics are given first with the rates in the lower section.

```
                              Read txn -> Write txn
---------------------------------- Total ------------------------------------
Count: 9620

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:01.24    0 00:00:00.24            5         3080           33

Min    0 00:00:00.06    0 00:00:00.04            1         1612            0
Max    0 00:00:54.05    0 00:00:00.34            1         2844            0
1st    0 00:00:05.61    0 00:00:01.60         2023         1800         5180

Tot    0 03:19:18.09    0 00:38:49.85        50173     29634816       326550


- - - - - - - - - - - - - - - - - - - Rate - - - - - - - - - - - - - - - - - -
Count: 0

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:00.00    0 00:00:00.00            0            0            0
```

FIGURE 4. Full Report - Section 2

The next sections relate to thread states discussed in “The Monitor and Loader Threads” on page 320. The Wait for Input section (applicable only to the Continuous LogMiner) provides the statistics and rate for the ‘z’ state. The thread has the read lock, but is waiting for input while there is no data to read.

```
                                 Wait For Input
---------------------------------- Total ------------------------------------
Count: 2076

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:00.98    0 00:00:00.00            0            0            0

Min    0 00:00:00.00    0 00:00:00.00            0            0            0
Max    0 00:00:01.01    0 00:00:00.00            0            0            0
1st    0 00:00:01.00    0 00:00:00.00            0            0            1

Tot    0 00:33:57.16    0 00:00:00.06            0            0            2


- - - - - - - - - - - - - - - - - - - Rate - - - - - - - - - - - - - - - - - -
Count: 0

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:00.00    0 00:00:00.00            0            0            0
```

FIGURE 5. Full Report - Section 3

The Read Input section provides the statistics and rate for the actual reads, the ‘<‘state.

```
                                  Read Input
------------------------------------ Total -------------------------------------
Count: 9634

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:00.02    0 00:00:00.00            0          106            0

Min    0 00:00:00.00    0 00:00:00.00            0           15            0
Max    0 00:00:01.69    0 00:00:00.01            0           13            0
1st    0 00:00:00.08    0 00:00:00.00            1           28            4

Tot    0 00:03:22.22    0 00:00:41.08           57      1027639         2304


- - - - - - - - - - - - - - - - - - - - Rate - - - - - - - - - - - - - - - - - - -
Count: 0

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:00.00    0 00:00:00.00            0            0            0
```

FIGURE 6. Full Report - Section 4

The sort portion of the cycle, the ‘s’ state, includes the statistics and rate.

```
                                   Sort Data
------------------------------------ Total -------------------------------------
Count: 9634

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:00.00    0 00:00:00.00            0            0            0

Min    0 00:00:00.00    0 00:00:00.00            0            0            1
Max    0 00:00:00.35    0 00:00:00.00            0            0            0
1st    0 00:00:00.00    0 00:00:00.00            0            0            1

Tot    0 00:00:03.96    0 00:00:01.16            0            0           20


- - - - - - - - - - - - - - - - - - - - Rate - - - - - - - - - - - - - - - - - - -
Count: 0

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:00.00    0 00:00:00.00            0            0            0
```

FIGURE 7. Full Report - Section 5

Output Synch Stall provides statistics and rate for the time spent waiting (in the 'W' state) for the dbkey locks to be granted prior to the write phase.

```
                                Output Synch Stall
---------------------------------- Total -----------------------------------
Count: 701

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:02.34    0 00:00:00.00            0            0            7

Min    0 00:00:00.00    0 00:00:00.00            0            0            0
Max    0 00:00:37.46    0 00:00:00.01            0            0            0
1st    0 00:00:02.03    0 00:00:00.00            0            0            0

Tot    0 00:27:24.03    0 00:00:04.56           15            0         5089


- - - - - - - - - - - - - - - - - - - Rate - - - - - - - - - - - - - - - - - - -
Count: 0

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:00.00    0 00:00:00.00            0            0            0
```

FIGURE 8. Full Report - Section 6

Write Target reports statistics and rates while the threads are writing to the target, the '>' state.

```
                                 Write Target
---------------------------------- Total -----------------------------------
Count: 9620

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:01.04    0 00:00:00.23            4         2960           32

Min    0 00:00:00.06    0 00:00:00.04            1         1539            0
Max    0 00:00:40.12    0 00:00:00.39            1         3490            0
1st    0 00:00:05.53    0 00:00:01.60         2022         1769         5174

Tot    0 02:48:18.62    0 00:37:59.61        50101     28474515       319117


- - - - - - - - - - - - - - - - - - - Rate - - - - - - - - - - - - - - - - - - -
Count: 0

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:00.00    0 00:00:00.00            0            0            0
```

FIGURE 9. Full Report - Section 7

The LSN (Loader Sequence Number) Lock Stall provides statistics and rates for the time spent waiting to get the ‘read’ lock again. This corresponds to the ‘R’ state.

```
                                LSN Lock Stall
---------------------------------- Total -----------------------------------
Count: 9635

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:00.65    0 00:00:00.00            0            1            5

Min    0 00:00:00.00    0 00:00:00.00            0            1            0
Max    0 00:00:11.90    0 00:00:00.00            2            4            1
1st    0 00:00:00.00    0 00:00:00.00            0            1            1

Tot    0 01:44:38.48    0 00:00:22.53          102         9726        48445


- - - - - - - - - - - - - - - - - - - - Rate - - - - - - - - - - - - - - - - - -
Count: 1

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:02.03    0 00:00:00.00            0            1            0
```

FIGURE 10. Full Report - Section 8

A histogram concludes the report. The histogram shows the distribution of elapsed time per read/write transaction. On the left are the time periods, expressed in seconds/10,000 (from under 0.01 of a second to greater than or equal to 10 seconds.)

The rate columns are for the current statistics interval; the total columns for the entire run of the statistics monitor. Each shows both a count and a percentage with the totals at the top. At the far right, find the current (current interval) and total (total session) averages.

The Minimum and Maximum indicate that the quickest time for a transaction was 0.0689 seconds and the maximum was 54.0550 seconds.

Read txn -> Write txn

```
                          Elapsed time (secondsx10,000)
-------------------------------------------------------------------------------
Minimum: 689
Maximum: 540550

                          ___Rate____          ___Total___
Limit                    Count     %         Count     %
                             0  100%           9620  100%

<                  100       0    0%             0    0%  <- CurAve: 0.00
<                  500       0    0%             0    0%
<                 1000       0    0%            72    0%
<                 2000       0    0%           932    9%
<                 3000       0    0%          1886   19%
<                 4000       0    0%          1491   15%
<                 5000       0    0%           899    9%
<                 6000       0    0%           544    5%
<                 7000       0    0%           432    4%
<                 8000       0    0%           308    3%
<                 9000       0    0%           279    2%
<                10000       0    0%           190    1%
<                20000       0    0%          1143   11%  <- TotAve: 12430.45
<                30000       0    0%           580    6%
<                40000       0    0%           322    3%
<                50000       0    0%           152    1%
<                60000       0    0%            84    0%
<                70000       0    0%            84    0%
<                80000       0    0%            40    0%
<                90000       0    0%            27    0%
<               100000       0    0%            27    0%
>=              100000       0    0%           128    1%
```

FIGURE 11. Full Report - Section 9

Because the full report is so long and so detailed, it is normally written to the log, instead of viewed on-line.

Brief Report Example

The following sample shows a brief report for 20 threads (only 19 of which are currently running). It also shows that the processing is caught up, after previously reporting latency.

```
================================================================================
20-JAN-2019 16:44:55.19 -----------------     REGTESTJDBR      -----------------

              0123456789abcdefghij
Loader States: <>>>> >WW>W>W>>>>>>>
                   Records               Transactions
Read:                514557           127805 thru 20-JAN-2019 16:32:41.86
     JDBC:           500127             9500 thru 20-JAN-2019 16:32:41.64
  Current:                0                0
Rdb CLM: AIJ SeqNo          VBN              TSN             CTSN
                10       363689           165087          165087

Throughput: 0.00 per 0.00 (0.00%) Latency: 0.00 (In: 0.00 Out: 0.00)

Processing caught-up
```

FIGURE 12. Brief Report

The date, title, and thread states should be familiar from the detail report, although they are arranged differently here. The records and transactions read, processed, and current provide somewhat different information than the Detail report. The two lines beginning “Rdb CLM:” provide the checkpoint restart information.

State Report Examples

Here is an example of a State report with three threads. Note the similarity of the first few lines to the Detail report format.

```
Rate:   6.00                        REGTESTORAJ            20-JAN-2019 16:41:22.46
================================================================================
 In: 20-JAN-2019 16:41:22.46 [5.000ms]    OCI: 20-JAN-2019 16:41:22.33 [   0.13]
                                         Chkpt              0  Recs            1
                                         CLM    0.00 |  Inpt   0.0%  Cnvt   0.0%
 012 ----------------------------        ------------   Sort   0.0%  Trgt   0.0%
 <>R                                     LML    0.00    Sync   0.0%  Ckpt   0.0%
 [  0.00] 0 VA->Append
 [  0.00] 1 Write->OCI
 [  0.01] 2 Input->MBX wait
```

FIGURE 13. State Report with Three Threads

The report type ‘S’tate provides expanded information about the current state of each of the Loader threads.[1] The example shown first includes 3 threads.

The first lines of the display are the same as the Detail report type.

Rate. The rate shown (first line, left) is 6.00. The refresh rate for the statistics is every 6 seconds. This is specified in starting the monitor.

Title. REGTESTORAJ is the name of the thread family, the Loader name. (In the example, the title indicates that this is a run of the regression testing.)

Dates. The date on the title line is date/time for the statistics generation. The date for input is the timestamp of the last source transaction read from the Continuous LogMiner mailbox or input file. The date for output (on the right) is the timestamp of the last source transaction written to the target. The trailing times are shown just after the dates, 5ms in the case of input and .13 in the case of output. (Seconds is the default units and 0.13 should be read as .13 seconds or .13s.)

Target Type. The target type of ‘OCI’ is shown just before the date for output. OCI is the transport used for an Oracle database target.

The next lines give familiar information, but in a compact format.

1. The “state” could alternately be called the “status” or the “current operation”.

Latency. The CLM and LML items show the percentages that represent the latency for the Continuous LogMiner and for the Loader. That the values are both 0.00, meaning there is no latency.

Latency details are also available from “Inpt”, “Sort”, “Sync”, “Cnvt”, “Trgt”, and “Ckpt” which reflect the latencies for Input, Sort, Lock Synchronization, Conversion, Target, and Checkpoint.

Loader States Summary. The two lines on the left that are 012 followed by hyphens and <>R give the same summary of thread states that is shown in the Detail report type. The top line identifies the threads and the next line shows their state. That is, the first thread < is currently reading from the input mailbox, the next > is writing to the target, and the last R is waiting to get the read lock. See “Thread State Codes” on page 321 for definitions of these single character codes.

Detail for Each Thread. The list that comes next is the reason for the State report type. Each line shows the latency for the thread, the thread number, and the state. These descriptions of the state of the thread are wordier and, perhaps, more readily interpreted than the single character descriptions. These options and what they mean are included in “Thread Details for the Statistics Monitor” on page 595.

There are 3 lines because there are 3 threads. There are 16 lines available for the threads which means that the bottom of the screen is blank in this case. If there are more than 16 threads, the State report type uses 2 columns.

The second example has 16 threads. To interpret the thread states shown, note that DETAILS and PEOPLE are table names and MATERIALIZED_TABLE also identifies a table.

```
Rate:   6.00                        REGTESTJDBR            20-JAN-2019 16:34:36.00
================================================================================
 In: 20-JAN-2019 16:32:03.67 [   2.5m]   JDBC: 20-JAN-2019 16:32:02.94 [   2.6m]
                                         Chkpt             45  Recs          2177
                                         CLM    2.5m |  Inpt   1.5%  Cnvt  39.1%
 0123456789abcdefghij ------------       ------------   Sort   0.0%  Trgt  41.6%
 W>W>>>>>>>>>>WW>>>>>                    LML    2.71    Sync  15.7%  Ckpt   2.1%
 [  0.84] 0 Output->Synch wait           [  0.38] g Write->JDBC
 [  0.19] 1 DETAILS[batch:19]            [  0.11] h Output->Process buffer
 [  1.82] 2 Output->Synch wait           [  0.20] i PEOPLE[batch:10]
 [  0.05] 3 MATERIALIZED_TABLE[batch:12] [  0.77] j DETAILS[batch:20]
 [  0.30] 4 Output->Process buffer
 [  0.07] 5 DETAILS[batch:14]
 [  0.20] 6 DETAILS[batch:16]
 [  0.53] 7 PEOPLE[batch:10]
 [  1.18] 8 Output->Process buffer
 [  0.18] 9 Write->JDBC
 [  0.94] a Output->Process buffer
 [  1.03] b Output->Process buffer
 [  0.31] c PEOPLE[batch:10]
 [  1.77] d Output->Synch wait
 [  5.70] e Output->Synch wait
 [  0.47] f Checkpoint
```

FIGURE 14. State Report with Over Sixteen Threads

Comma Separated Values Report Example

The CSV report uses the following headings

TABLE 4. Columns in the CSV Output Format

Column Heading	Meaning
RptDtTm	Report date/time
LdrName	Loadername
CommitDtTm	Last source database commit date/time
Rows	Rows -Number of rows processed in interval
Txns	Transactions - Number of transactions processed in interval
ProcDurSec	Process Duration Seconds - the number of seconds worth of source database transactions processed in the interval
IntvlDurSec	Interval Duration Seconds - the number of seconds in the interval
ThruRatio	Throughput Ratio - ProcDuSec/IntvDuSec
TrailTmSec	Trailing Time Seconds -Number of seconds that the target is trailing the source
InTimOut	Input timeouts
OutFail	Output failure - May be a deadlock, the target down, or other.
LdrThr	Number of Loader Threads
TotLat	Average (for the current statistics refresh) of the number of seconds required for the Loader threads to read the data from the CLM mailbox and write the data and the checkpoint. Since it includes all phases of the Loader processing, this value is frequently larger than the sum of other two numbers which represent input and output.
InLat	Average (for the current statistics refresh) of the number of seconds the Loader threads spent reading data from the CLM mailbox.
OutLat	Average (for the current statistics refresh) of the number of seconds the Loader threads spent writing data to the target and checkpointing.

All targets will generate a report with the same column list. Note that the first few lines may show “(none)” for the commit date time because the actual workload was not started when the statistics job was run.

The examples aren’t designed for readability. The following includes line wraps that are not part of the output.

```
RptDtTm,LdrName,CommitDtTm,Rows,Txns,ProcDurSec,IntvlDurSec,ThruRatio,TrailTmSec,InTmOut,
OutFail,LdrThr,TotLat,InLat,OutLat
 3-FEB-2004 11:56:11.12,REGTESTRDB, 3-FEB-2004
11:43:49.41,0,0,0.00,0.00,0.00,0.00,0,0,3,0.00,0.00,0.00
 3-FEB-2004 11:56:17.13,REGTESTRDB, 3-FEB-2004
11:43:49.58,90,1,0.17,6.00,0.03,747.54,0,0,2,5.16,2.18,2.96
 3-FEB-2004 11:56:23.13,REGTESTRDB, 3-FEB-2004
11:43:49.74,116,2,0.15,6.00,0.03,753.39,0,0,2,9.33,2.83,6.49
 3-FEB-2004 11:56:23.69,REGTESTRDB, 3-FEB-2004
11:43:49.74,17,0,0.00,0.57,0.00,753.96,0,0,2,0.00,0.00,0.00
```

FIGURE 15. CSV Report Example

See “Statistics Output with Other Tools” on page 352 for additional discussion of how to combine the Loader’s CSV output with other tools and additional settings that can be used with the CSV output type.

Date Format in the CSV Output. Date format can be a stumbling block in coordinating between tools. To provide more control of date output, the logical name, JCC_LOGMINER_LOADER_STAT_CSV_DATE, modifies the date format of the dates in the CSV option. This logical name enables the user to tailor the date format to a specific tool or database.

Note that T4 output is best left with the default OpenVMS date time format, as TLViz is designed to analyze the OpenVMS data.

Acceptable values for the logical name are any valid LIB$FORMAT_DATE_TIME date conversion string. (This is the same as used by the DATE_FORMAT keyword. See “Keyword: Date_format” on page 229.) The default, if it is not defined or if it is invalidly defined, is the standard OpenVMS date time format.[1]

The following example shows an exception message. The exception message and the dates are shown in red. Note that, due to the exception message, the dates default to the standard OpenVMS date time format.

1. For valid formaats accepted by the LIB$FORMAT_DATE_TIME routine, see the OpenVMS documentation..


```
$ define JCC_LOGMINER_LOADER_STAT_CSV_DATE "|!Y4-!MN0-!D0 !H04:!M0:!S0.!C2"
$ jcc_lml_statistics regtestsrdb 6 csv

JCC LogMiner Loader Statistics D02.01.00 (built 12-DEC-2003 15:56:00.29)

%jcc_lml_statistics: JCC_LOGMINER_LOADER_STAT_CSV_DATE = "|!Y4-!MN0-!D0
!H04:!M0:!S0.!C2".
%jcc_lml_statistics: failure setting time format: %LIB-F-ILLINISTR,
illegal initialization string

RptDtTm,LdrName,CommitDtTm,Rows,Txns,ProcDurSec,IntvlDurSec,ThruRatio,
TrailTmSec,InTmOut,OutFail,LdrThr,InLat,OutLat,TotLat
12-JAN-2004 11:55:30.50,REGTESTRDB,12-JAN-2004 11:33:29.06,
0,0,0.00,0.00,0.00,0.00,0,0,3,0.00,0.00,0.00
12-JAN-2004 11:55:36.50,REGTESTRDB,12-JAN-2004 11:33:31.31,
969,5,2.25,6.00,0.38,1325.19,0,0,2,1.26,1.20,2.49
12-JAN-2004 11:55:42.50,REGTESTRDB,12-JAN-2004 11:33:32.93,
1046,4,1.63,6.00,0.27,1329.57,0,0,3,1.15,1.18,2.42
```

FIGURE 16. Exception Message for Date Format

This example shows the exception corrected. Again, the dates are shown in red. (The correction is the last character before the end quote.)

```
$ define JCC_LOGMINER_LOADER_STAT_CSV_DATE "|!Y4-!MN0-!D0 !H04:!M0:!S0.!C2|"
$ jcc_lml_statistics regtestrdb 6 csv

JCC LogMiner Loader Statistics D02.01.00 (built  6-JAN-2004 15:48:35.78)

%jcc_lml_statistics: JCC_LOGMINER_LOADER_STAT_CSV_DATE = "|!Y4-!MN0-!D0
!H04:!M0:!S0.!C2|".
RptDtTm,LdrName,CommitDtTm,Rows,Txns,ProcDurSec,IntvlDurSec,ThruRatio,
TrailTmSec,InTmOut,OutFail,LdrThr,InLat,OutLat,TotLat
2004-01-12 11:58:49.17,REGTESTRDB,2004-01-12 11:34:31.11,
0,0,0.00,0.00,0.00,0.00,0,0,3,0.00,0.00,0.00
2004-01-12 11:58:55.17,REGTESTRDB,2004-01-12 11:34:33.82,
1099,6,2.71,6.00,0.45,1461.34,0,0,2,0.98,1.16,2.21
2004-01-12 11:59:01.17,REGTESTRDB,2004-01-12 11:34:36.03,
1121,5,2.21,6.00,0.37,1465.13,0,0,3,1.20,1.13,2.35
```

FIGURE 17. Successful Date Re-Formatting

T4 Report Example

The actual output from the Loader with T4 as the report type is comma separated values. However, it differs from the Loader CSV output type in ways that enhance compatibility with some existing tools. T4 is an OpenVMS toolset that warranted the additional output type.

In order to support T4 tools, the T4 output type has

- Extra header lines
- More meaningful column headings
- Extra columns — including all of the numeric fields from the Full display (displayed as deltas)

Also, to aid use of Loader output with the T4 tools, all output types were enhanced with:

- Alignment of statistics collection to clock time boundaries (to permit collaboration with other T4 Friends in the same environment)
- Options of closing and reopening output files with meaningful names that include timestamps
- Options for controlling file placement

The Loader kit includes JCC_LML_T4_TEMPLATE.COM, a template DCL procedure for generating T4 statistics. Find the template in JCC_TOOL_COM. This procedure can be copied, modified to suit the need, and executed (in batch or otherwise) to capture T4 statistics for a Loader session.

Raw T4 output is cumbersome to read. An example is given below. However, see “Statistics Output with Other Tools” on page 352 for illustration of the value of T4.

```
ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLA
S,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATL
AS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,AT
LAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS,ATLAS
31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-
2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-
2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-
2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-
2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-
2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-
2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004,31-AUG-2004
21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:2
4,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:
24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24,21:24
[REGTESTTUX]Sample Time,[REGTESTTUX.LML]Threads,[REGTESTTUX.LML]Sample Seconds,[REGTESTTUX.LML]Processed Source
Seconds,[REGTESTTUX.LML]Throughput Ratio,[REGTESTTUX.LML]Trailing Seconds,[REGTESTTUX.CLM]Ave CLM Latency,[REGTESTTUX.LML]Ave Total
Latency,[REGTESTTUX.LML]Ave Input Latency,[REGTESTTUX.LML]Ave Output Latency,[REGTESTTUX.LML]Input Transactions,[REGTESTTUX.LML]Input
Records,[REGTESTTUX.LML]Input Records:Modify,[REGTESTTUX.LML]Input Records:Delete,[REGTESTTUX.LML]Input
Records:Commit,[REGTESTTUX.LML]Input Read Timeouts,[REGTESTTUX.LML]Input Records:Filtered,[REGTESTTUX.LML]Input
Records:Excluded,[REGTESTTUX.LML]Input Records:Unknown,[REGTESTTUX.LML]Input Records:Restart,[REGTESTTUX.LML]Input Txns:No
Work,[REGTESTTUX.LML]Input Txns:Heartbeat,[REGTESTTUX.LML]Input Checkpoint:Timeout,[REGTESTTUX.LML]Input Checkpoint:Buffer
Limit,[REGTESTTUX.LML]Output Checkpoints,[REGTESTTUX.LML]Output Records,[REGTESTTUX.LML]Output Messages,[REGTESTTUX.LML]Output
Message Ave Size,[REGTESTTUX.LML]Output Failures,[REGTESTTUX.LML]Output Timeouts,[REGTESTTUX.LML]Output Checkpoint:No
Work,[REGTESTTUX.LML]Output Tables,[REGTESTTUX.LML]Output
Records:Filtered,[REGTESTTUX.CLM]CPU,[REGTESTTUX.CLM]BIO,[REGTESTTUX.CLM]DIO,[REGTESTTUX.CLM]PageFaults,[REGTESTTUX.LML]Read txn
-> Write txn:Rate Count,[REGTESTTUX.LML]Read txn -> Write txn:Ave Elapsed,[REGTESTTUX.LML]Read txn -> Write txn:Ave CPU,[REGTESTTUX.LML]Read
txn -> Write txn:Ave DIO,[REGTESTTUX.LML]Read txn -> Write txn:Ave BIO,[REGTESTTUX.LML]Read txn -> Write txn:Ave PgFaults,[REGTESTTUX.LML]Wait
For Input:Rate Count,[REGTESTTUX.LML]Wait For Input:Ave Elapsed,[REGTESTTUX.LML]Wait For Input:Ave CPU,[REGTESTTUX.LML]Wait For Input:Ave
DIO,[REGTESTTUX.LML]Wait For Input:Ave BIO,[REGTESTTUX.LML]Wait For Input:Ave PgFaults,[REGTESTTUX.LML]Read Input:Rate
Count,[REGTESTTUX.LML]Read Input:Ave Elapsed,[REGTESTTUX.LML]Read Input:Ave CPU,[REGTESTTUX.LML]Read Input:Ave
DIO,[REGTESTTUX.LML]Read Input:Ave BIO,[REGTESTTUX.LML]Read Input:Ave  PgFaults,[REGTESTTUX.LML]Sort Data:Rate
Count,[REGTESTTUX.LML]Sort Data:Ave Elapsed,[REGTESTTUX.LML]Sort Data:Ave CPU,[REGTESTTUX.LML]Sort Data:Ave DIO,[REGTESTTUX.LML]Sort
Data:Ave BIO,[REGTESTTUX.LML]Sort Data:Ave PgFaults,[REGTESTTUX.LML]Write Target:Rate Count,[REGTESTTUX.LML]Write Target:Ave
Elapsed,[REGTESTTUX.LML]Write Target:Ave CPU,[REGTESTTUX.LML]Write Target:Ave DIO,[REGTESTTUX.LML]Write Target:Ave
BIO,[REGTESTTUX.LML]Write Target:Ave PgFaults,[REGTESTTUX.LML]LSN Lock Stall:Rate Count,[REGTESTTUX.LML]LSN Lock Stall:Ave
Elapsed,[REGTESTTUX.LML]LSN Lock Stall:Ave CPU,[REGTESTTUX.LML]LSN Lock Stall:Ave DIO,[REGTESTTUX.LML]LSN Lock Stall:Ave
BIO,[REGTESTTUX.LML]LSN Lock Stall:Ave PgFaults,[REGTESTTUX.LML]Read txn -> Write txn:Minimum,[REGTESTTUX.LML]Read txn -> Write
txn:Maximum,[REGTESTTUX.LML]Read txn -> Write txn:Current Average
31-AUG-2004
21:25:21.00,5,59.72,0.00,0.00,39.49,9.50,1.27,0.39,0.88,884,4451,3555,12,884,0,0,0,0,0,95,0,0,0,78,4301,358,6811,0,0,0,3,0,1.59,4938,505,2026,77,1.27,0.05,3
,99,32,1782,0.01,0.00,0,0,0,77,0.39,0.01,0,64,5,77,0.00,0.00,0,0,0,77,0.87,0.04,3,34,26,85,0.88,0.00,1,0,0,3447,49010,12744.03
31-AUG-2004
21:26:21.00,5,60.00,6.49,0.11,93.00,80.29,1.19,0.38,0.79,1161,5328,4151,17,1160,0,0,0,0,0,126,0,0,0,102,5310,424,7064,1,0,0,0,0,1.16,5327,158,3,102,1.19,0.
05,4,95,30,1781,0.01,0.00,0,0,0,104,0.38,0.01,0,60,4,104,0.00,0.00,0,0,0,102,0.80,0.04,3,34,25,116,0.89,0.00,1,0,0,1044,75066,11861.81
31-AUG-2004
21:27:21.00,3,60.00,34.01,0.57,118.98,107.46,0.47,0.12,0.35,5090,23760,18559,111,5090,0,0,0,0,0,580,0,0,0,452,23415,1646,8071,5,0,0,0,0,4.29,23759,534,7,
452,0.47,0.05,2,97,7,8370,0.00,0.00,0,0,0,451,0.12,0.01,0,61,1,451,0.00,0.00,0,0,0,452,0.35,0.04,1,35,6,462,0.18,0.00,1,0,0,585,0,4688.52
31-AUG-2004
21:28:21.00,4,60.00,13.65,0.23,165.33,142.02,0.96,0.40,0.55,1582,7386,5778,26,1582,0,0,0,0,0,148,0,0,0,143,7275,512,8031,1,0,0,0,0,1.55,7388,171,0,143,0.9
6,0.05,2,93,2,2034,0.01,0.00,0,0,0,143,0.40,0.01,0,60,0,143,0.00,0.00,0,0,0,143,0.55,0.04,1,32,2,144,0.69,0.00,1,0,0,0,0,9635.97
31-AUG-2004
21:29:21.00,7,60.00,132.78,2.21,92.55,126.77,0.40,0.08,0.32,7034,27626,20529,62,7035,0,0,0,0,0,728,0,0,0,628,27244,2245,6764,8,0,0,0,0,5.36,27624,762,0,6
28,0.40,0.04,2,88,3,10458,0.00,0.00,0,0,0,631,0.08,0.01,0,52,0,631,0.00,0.00,0,0,0,628,0.32,0.03,1,34,2,634,0.13,0.00,1,1,0,0,0,4011.28
31-AUG-2004
21:30:21.00,4,60.00,52.38,0.87,100.17,93.51,0.32,0.07,0.25,9276,29639,20322,41,9276,1,0,0,0,0,1088,0,1,0,824,29041,2548,6193,13,0,0,0,0,5.87,29639,735,0,
824,0.32,0.03,2,83,0,10767,0.00,0.00,0,0,0,820,0.07,0.01,0,45,0,820,0.00,0.00,0,0,0,824,0.25,0.03,1,37,0,824,0.17,0.00,1,0,0,0,101067,3200.87
31-AUG-2004
21:31:21.00,4,60.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0,0,0,0,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.00,0,0,0,0,0.00,0.00,0,0,0,6,10.00,0.00,0,0,0,0,0.00,0.00,0,0,
0,0,0.00,0.00,0,0,0,0,0.00,0.00,0,0,0,0,0.00,0.00,0,0,0,0,0,0.00
31-AUG-2004
21:32:21.00,4,60.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0,0,0,0,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.00,0,0,0,0,0.00,0.00,0,0,0,6,10.00,0.00,0,0,0,0,0.00,0.00,0,0,
0,0,0.00,0.00,0,0,0,0,0.00,0.00,0,0,0,0,0.00,0.00,0,0,0,0,0,0.00
31-AUG-2004
21:33:01.70,0,40.70,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0,0,0,0,0,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.00,0,0,0,0,0.00,0.00,0,0,0,5,8.56,0.00,0,0,0,1,173.92,0.66,34
,1165,9,0,0.00,0.00,0,0,0,0,0.00,0.00,0,0,0,3,169.45,0.01,2,0,0,0,0,0,0.00
```

FIGURE 18. Example of Raw T4 Statistics Output

The example is color coded to assist discussion. It includes four “lines” of header and n “lines” of data. (The data is the blue.) Each line contains m columns. The first three header lines (black, green, and lilac) are fairly dull. Each gives m repetitions of a value. The first line is the node name (“Atlas”); the second the date; and the third the time.

The fourth header line provides meaningful column headings. These are made up of the Loadername, an indicator of source, and the column name. The indicator of source is missing for the sample time and, for the rest, is either ".LML" or ".CLM." (The latter, CLM, indicates that the LogMiner is the source of the statistic.) These are highlighted in red in the list to follow.[1] The remainder are Loader statistics. To save space, this list does not include the Loadername.

[]Sample Time	[.LML]Output Checkpoints	[.LML]Read Input:Ave BIO
[.LML]Threads	[.LML]Output Records	[.LML]Read Input:Ave PgFaults
[.LML]Sample Seconds	[.LML]Output Messages	[.LML]Sort Data:Rate Count
[.LML]Processed Source Seconds	[.LML]Output Message Ave Size	[.LML]Sort Data:Ave Elapsed
[.LML]Throughput Ratio	[.LML]Output Failures	[.LML]Sort Data:Ave CPU
[.LML]Trailing Seconds	[.LML]Output Timeouts	[.LML]Sort Data:Ave DIO
[.CLM]Ave CLM Latency	[.LML]Output Checkpoint:No Work	[.LML]Sort Data:Ave BIO
[.LML]Ave Total Latency	[.LML]Output Tables	[.LML]Sort Data:Ave PgFaults
[.LML]Ave Input Latency	[.LML]Output Records:Filtered	[.LML]Output Synch Stall:Rate Count
[.LML]Ave Sort Latency	[.CLM]CPU	[.LML]Output Synch Stall:Ave Elapsed
[.LML]Ave Synch Latency	[.CLM]BIO	[.LML]Output Synch Stall:Ave CPU
[.LML]Ave Convert Latency	[.CLM]DIO	[.LML]Output Synch Stall:Ave DIO
[.LML]Ave Target Latency	[.CLM]PageFaults	[.LML]Output Synch Stall:Ave BIO
[.LML]Ave Checkpoint Latency	[.LML]Read txn -> Write txn:Rate Count	[.LML]Output Synch Stall:Ave PgFaults
[.LML]Ave Output Latency	[.LML]Read txn -> Write txn:Ave Elapsed	[.LML]Write Target:Rate Count
[.LML]Input Transactions	[.LML]Read txn -> Write txn:Ave CPU	[.LML]Write Target:Ave Elapsed
[.LML]Input Records	[.LML]Read txn -> Write txn:Ave DIO	[.LML]Write Target:Ave CPU
[.LML]Input Records:Modify	[.LML]Read txn -> Write txn:Ave BIO	[.LML]Write Target:Ave DIO
[.LML]Input Records:Delete	[.LML]Read txn -> Write txn:Ave PgFaults	[.LML]Write Target:Ave BIO
[.LML]Input Records:Commit	[.LML]Wait For Input:Rate Count	[.LML]Write Target:Ave PgFaults
[.LML]Input Read Timeouts	[.LML]Wait For Input:Ave Elapsed	[.LML]LSN Lock Stall:Rate Count
[.LML]Input Records:Filtered	[.LML]Wait For Input:Ave CPU	[.LML]LSN Lock Stall:Ave Elapsed
[.LML]Input Records:Excluded	[.LML]Wait For Input:Ave DIO	[.LML]LSN Lock Stall:Ave CPU
[.LML]Input Records:Unknown	[.LML]Wait For Input:Ave BIO	[.LML]LSN Lock Stall:Ave DIO
[.LML]Input Records:Restart	[.LML]Wait For Input:Ave PgFaults	[.LML]LSN Lock Stall:Ave BIO
[.LML]Input Txns:No Work	[.LML]Read Input:Rate Count	[.LML]LSN Lock Stall:Ave PgFaults
[.LML]Input Txns:Heartbeat	[.LML]Read Input:Ave Elapsed	[.LML]Read txn -> Write txn:Minimum
[.LML]Input Checkpoint:Timeout	[.LML]Read Input:Ave CPU	[.LML]Read txn -> Write txn:Maximum
[.LML]Input Checkpoint:Buffer Limit	[.LML]Read Input:Ave DIO	[.LML]Read txn -> Write txn:Current Average

FIGURE 19. Column Headings for the T4 Output

The Performance Group, OpenVMS Engineering, has developed and advocated Timeline Tracking tools to study performance. JCC has adopted the T4 (Total Timeline Tracking Tool) methodology and has found it highlights performance issues very effectively.

1. The [.CLM]Ave CLM Latency is calculated as the Loader read time minus the transaction commit time. If the database is only open on a single node then this number is accurate. If the database is written to by processes on more than one OpenVMS node and the system timestamps are not properly synchronized (or if system time is altered), the value can be misleading. (Should the value be calculated to be negative, it will be set to zero to minimize the variance from reality.)

There are both upstream and downstream tools that enhance the impact of T4. All of T4's "Friends" follow TimeLine Collaboration (TLC) format. Because of this, it is possible to show timelines that include characteristics of OpenVMS, Rdb, and Loader performance — on the same graph — and to use the timelines to discover issues or illustrate the results of changed parameters.

T4 on OpenVMS zips the resulting files in such a way that Windows treats them as normal text files which permits mailing them to a designated recipient.

T4 is supplied as unsupported freeware. As T4V33, it is incorporated in OpenVMS beginning with Alpha Version 7.3-2[1] or can be downloaded from

http://h41379.www4.hpe.com/openvms/products/t4/

For additional pointers to "Friends" TLViz and CSVPNG, contact JCC LogMiner Loader support.

1. Find it in the SYS$ETC: directory with the file name T4_V33_KIT.EXE. There is also a .TXT description of the tool in the same directory.

The graph to follow is created with Excel from T4 output gathered during the regression testing of the Loader. The raw data is rescaled to highlight the correlations between different data points. The graph shows a jump in the number of threads (blue) after the Loader got behind (pink) and shows the resultant jump in the throughput ration.

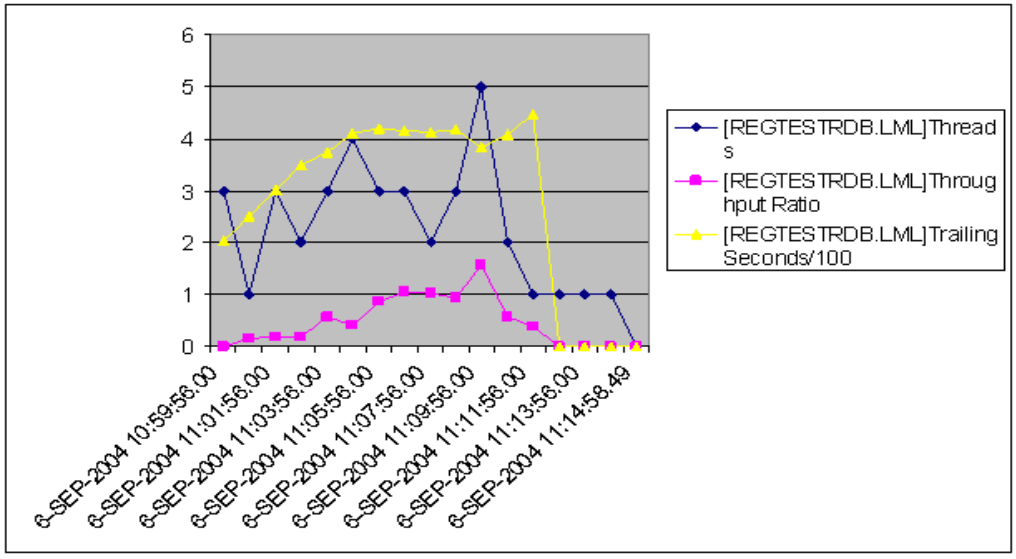


FIGURE 20. Excel Graph of Loader CSV Data

The graph to follow is done with TLVIZ (one of the T4 family of tools). It also illustrates the correlation between throughput ratio (green) and number of threads (red).

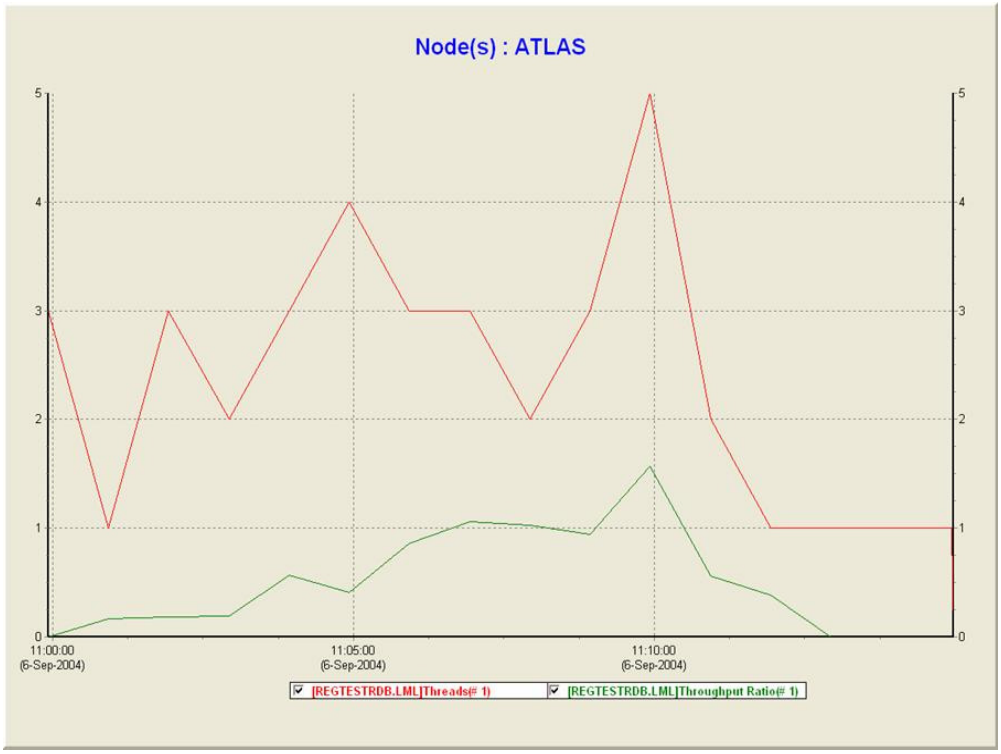


FIGURE 21. TLVIZ Graph of Loader T4 Data

The graph to follow is far more interesting. It presents several variables and scales two of them.[1]

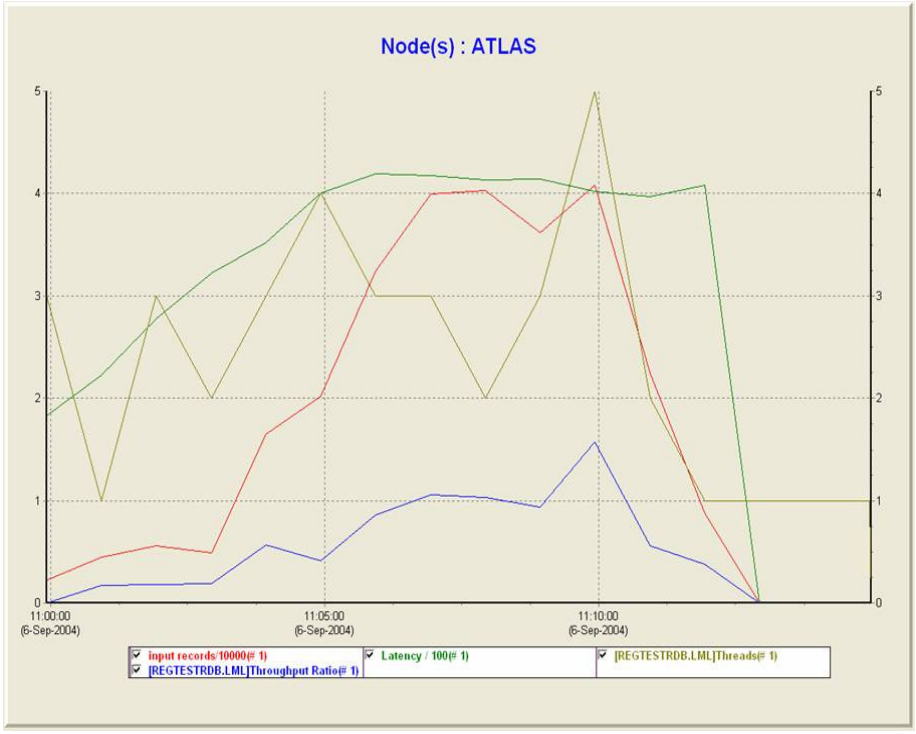


FIGURE 22. TLVIZ Graph with Scaled and Other Data

Graphs such as these can be used to identify and communicate problem areas and significant interactions.

1. It is reasonable to note that the system running the regression tests — from which this data is gathered — was extremely busy.

User Control for Flushing Statistics

The logical name JCC_LOGMINER_LOADER_STAT_FILE_SECONDS provides control over how often the statistics output is flushed to a file. This logical name can be set to a number of seconds at or above 10. JCC_LML_STATISTICS will then use this value to set the interval for flushing output buffers to disk. If this logical name is not set the default value is 600 seconds (ten minutes).

Statistics Output with Other Tools

This section discusses the relationship of output from the statistics monitor with additional tools.

Putting Statistics in a Database

The Comma Separated Values (CSV) statistics output is suitable to loading into a database for further analysis. The Loader kit includes procedures to support loading the CSV output into an Rdb database. See “Putting Statistics in a Database” on page 565.

To load data from CSV log files created by previous versions, you will need a special procedure which is also included with the kit. The procedure converts the CSV lines from the pre-V2.1 logs into the current CSV format. Fields that were not previously included will be set to NULL. The appendix includes specifications for this.

See also “Date Format in the CSV Output” on page 343 and “Statistics Options” on page 352.

Statistics Options

The logical name, JCC_LOGMINER_LOADER_STAT_OPTIONS, provides control over some statistics options. The options provided are particularly relevant for sessions that are loading CSV data directly into a database or into another tool.

Syntax is

```
$ define JCC_LOGMINER_LOADER_STAT_OPTIONS "<option>[,<option>]"
```

Options are

[NO]Header. Display (or not) header information, including the column names line, the program version and link information, and any wait related text. The default is Header. The default is illustrated in "CSV Report Example" on page 343.

[NO]Interactive. Make (or not) the session interactive. The default is interactive for interactive sessions and nointeractive for batch. Interactive attempts to read input from a terminal device. If there is no such input available, as when running JCC_LML_statistics interactively within a 'pipe' command, this causes unnecessary processing. NOinteractive does not have this issue.

File. The "File" option causes the statistics utility to send the output of the statistics program to a file rather than to the terminal or log file. This option can be used with any of the output types. It is most frequently used with T4. The file is set as shared read, which enables users to read the data that has been flushed to disk prior to the file being closed. The data is flushed approximately every 10 minutes. See "Example" on page 354.

Reopen. The "Reopen" option causes the statistics utility to close the output file at midnight and open a new file. The "File" option must also be specified for this option to have any affect. This option can be used with any of the output types. See "Example" on page 354.

LatencyScale. The default LatencyScale is 1.0. ("Latency Scale" on page 324 describes how this is used in the display of the latency numbers.) To change the LatencyScale define JCC_LOGMINER_LOADER_STAT_OPTIONS to include a different LatencyScale. For example, to set the display to switch from seconds to minutes when the number is equal to or greater than ten minutes and switch from minutes to hours when the number is equal to or greater than ten hours (and if there are not other options to set) use

```
$ define JCC_LOGMINER_LOADER_STAT_OPTIONS "LatencyScale=10.0"
```

Directory Placement

To change the directory for the output file set the logical name JCC_LOGMINER_-LOADER_STAT_OUTPUT_DIR to the directory specification desired. Two sample formats are:

```
$ define JCC_LOGMINER_LOADER_STAT_OUTPUT_DIR <disk>:[<dir>]
$ define JCC_LOGMINER_LOADER_STAT_OUTPUT_DIR <outputdir>:
```

Additional examples are shown in the following.

File Naming

With the “file” option specified, a file is created and named in the following format.

```
jcc_tool_logs:<type>_<system name>_<Loadername>_<yyyymmdd>_<hhmm>.<ext>
```

See “Example” on page 354 for an example.

Alternately, it is also possible to name the file by defining the logical name JCC_LOGMINER_LOADER_STAT_OUTPUT_FILE. The value of this logical name affects only the name and extension portions of the output file. More complete control also requires definition of the JCC_LOGMINER_LOADER_STAT_OUTPUT_DIR logical name.

For example:

```
$ define JCC_LOGMINER_LOADER_STAT_OUTPUT_DIR  mystats:

$ define JCC_LOGMINER_LOADER_STAT_OUTPUT_FILE this_loader.csv
```

Example

In the example, notice that the first file name shows the date and time that the file was initially opened and the second file name shows when the file was closed and reopened at midnight.

```
$ define JCC_LOGMINER_LOADER_STAT_OPTIONS "file,reopen"
$ define JCC_LOGMINER_LOADER_STAT_OUTPUT_DIR TEST:[T4]
$ jcc_lml_statistics regtestrdb 60 t4

JCC LogMiner Loader Statistics D02.02.00 (built 3-AUG-2004
10:41:18.35)

%jcc_lml_statistics: JCC_LOGMINER_LOADER_STAT_OUTPUT_DIR =
"TEST:[T4]".

Output> TEST:[T4]T4_ATLAS_REGTESTRDB_20040803_1239.csv
Output> TEST:[T4]T4_ATLAS_REGTESTRDB_20040804_0000.csv
```

FIGURE 23. Example: Setting Statistics Options

Modifying CLM Statistics Output

The Oracle Rdb LogMiner can, optionally, report it's runtime resource usage and processing statistics. This is enabled and frequency of reporting is controlled by the /statistics qualifier in the Oracle Rdb LogMiner command.[1] The default reporting interval is 3600 seconds (one hour). By default, the LogMiner statistics are disabled when the Loader heartbeat function is enabled.

The JCC LogMiner Loader includes a logical name — JCC_ADD_CLM_STATISTICS — that can modify or disable the Rdb LogMiner statistics. Set the value of this logical name equal to the interval (number of seconds) to pass between emitting table statistics to the log file. If the value is set to less than or equal to zero or is not numeric, the statistics output is disabled.

Note that there are performance implications. There is a cost for the CLM process to generate and write the information to the logging mailbox. There is a cost to the CTL process reading the mailbox and writing the log file. The override capability is provided, in fact, as part of a response to locking issues between writing these statistics and the Loader heartbeat mechanism. See “Logical Name Controls for Loader Procedures” on page 463.

JCC recommends setting the value to 0, if the statistics are uninteresting or never reviewed.

JCC recommends never setting the number too low, as low numbers cause more frequent statistics generation and more frequent statistics generation will negatively impact the CLM performance. An example of too low is sixty, but consider the impact before setting it down at all.

Disabling CLM Statistics. Turn off the CLM statistics with:

```
$ define JCC_ADD_CLM_STATISTICS 0
```

Changing the CLM Statistics Interval. Double the interval (reporting statistics half as often) with:

```
$ define JCC_ADD_CLM_STATISTICS 7200
```

1. Other realtime information is available with Oracle Rdb’s RMU/SHOW STATISTICS command.

The Log Files

The JCC LogMiner Loader Control Process acts as the logging sink for the Rdb Continuous LogMiner and for the Loader processes. The log files can be extremely useful in understanding what Loader families are doing and in resolving issues.

Note that the log files are also useful if you wish to communicate with Loader support.

There are three sets of log files

- The CLML (parent) log file which will be named according to the batch that is running it.
- The LogMiner log file which is named
  `jcc_tool_logs:jcc_run_clm-<LoaderName>.log`
- The thread log files which are named
  jcc_tool_logs:jcc_run_clm-<LoaderName>_n.log
  where n represents the thread number (using 0-9 and a-v)

Note: There is also an Activation Log. See "Activation Log" on page 368.

Logging Control

The Loader Administrator can control the logging through the keyword Logging and through various logical names. See "Keyword: Logging" on page 256 and the following.

Note that not all possible inclusions in the log file are always wise. Include the echo of the Control File[1] and whatever else you need to get your work done. Including unnecessary information in the logs leads to processing time and buffered i/o costs that are wasted. It also risks disguising useful information in the verbosity.

Choose different approaches for running in production than you might choose to diagnose an issue.

1. See "Echoing the Control File in the Log" on page 358.

Logging and the Rdb Continuous LogMiner

Logical names can be defined to control the level of logging that the is enabled for the Oracle Rdb LogMiner. It is useful to set these logical names during development or when you need to understand an issue or examine performance. Generally, these logical names should not be set for production, as the output can become quite verbose.

Setting one of these logical names adds that option to the command line for starting continuous mining.

```
DEFINE JCC_ADD_CLM_LOG TRUE
DEFINE JCC_ADD_CLM_TRACE TRUE
DEFINE JCC_ADD_CLM_DEBUG TRUE
```

LOG. adds the /log qualifier to the Oracle Rdb LogMiner command. See the Oracle Rdb LogMiner documentation for more information regarding the output generated by your version of Oracle Rdb.

NOLOG is the default.

TRACE. adds the /trace qualifier to the Oracle Rdb LogMiner command. See the Oracle Rdb LogMiner documentation for more information regarding the output generated by your version of Oracle Rdb.

NOTRACE is the default.

DEBUG. adds the *undocumented* debug option to the Oracle Rdb LogMiner command. The output varies with the version of Oracle Rdb. JCC recommends that this qualifier only be enabled at the request of JCC or of Oracle Support.

Not including debug is the default behavior.

Thread Log Files

There will be one log file for all threads[1] or one thread log file per thread or one thread log file per thread initialization. Which of these choices to use is determined

1. See “Parallelism and Loader Threads” on page 383

by the logical name JCC_CLML_logging_style. Valid values are D(efault), S(ingle), and R(euse).

Default. Provides a log file per thread instance. Should the logical not be defined or be defined improperly, default is used. The name of the log file will be

```
JCC_tool_logs:jcc_run_LML-<LoaderName>_<thread number>.log
```

Single. Writes the output for all threads to a single log file. In the case of logging style single or of a single thread, the log name is

```
JCC_tool_logs:JCC_run_LML-<LoaderName>.log
```

Reuse. Initially creates a log file per thread, but does not close these files. If a thread exits, the log file may be reused by a later thread that uses the same thread number. The log files will be named as for the default.

Splitting Log Files Into a File for Each Thread

For analyzing difficulties, it may become important to see each thread reflected in a separate log. If you have been using a “single” log for all the threads, you can achieve a split into a log for each thread with the procedure

JCC_EXTRACT_THREAD_LOGS <LML single file format log>

This separates the log file named into one file per thread.

Example. The following example illustrates four threads.

```
$ JCC_EXTRACT_THREAD_LOGS logs:jcc_run_lml-REGTESTSRDB.log;1
[Thread 0] -> JCC_RUN_LML-REGTESTSRDB_0.LOG
[Thread 1] -> JCC_RUN_LML-REGTESTSRDB_1.LOG
[Thread 2] -> JCC_RUN_LML-REGTESTSRDB_2.LOG
[Thread 3] -> JCC_RUN_LML-REGTESTSRDB_3.LOG
$
```

Echoing the Control File in the Log

The Loader can be set to echo the Control File in the log. This can be a major benefit in problem reporting and resolution.

JCC strongly recommends placing this line near the beginning of the Control File.

```
logging~initialization
```

The result is similar to set verify in DCL.

See also “Keyword: Logging” on page 256.

Deltas or Cumulative Statistics

When the Loader is used in complex environments with high throughput demands, it is important to be able to trace exactly where any latencies are introduced. Some Administrators prefer to see cumulative statistics and some prefer to see deltas.

When the logical name JCC_LOGMINER_LOADER_STAT_INTERVAL is set,[1] the Loader provides detailed data on each table. The information in the log, by default, is shown as cumulative statistics. It is possible to display deltas from one interval to the next, instead of cumulative numbers. To do so, define the logical name JCC_LOGMINER_LOADER_STAT_TYPE as follows

```
$ Define JCC_LOGMINER_LOADER_STAT_TYPE DELTA
```

The information in the log will then look something like the following. (Space is squeezed out to make this example more readable in this context.)

```
-------------------------------------------------------------------------------
-Src---Table name---Tgt- ------------------Deltas------------- -------------Flags-----------
                        -Insert/Modify- --Delete-- -Filtered- Len DBK IgnD Action
VIRTUAL                             20           0           0
                VIRTUAL             20           0           0               InsertOnly
PEOPLE                               0           0           0  N
                 PEOPLE              0           0           0      Y   Y    Ins:Y Upd:N Del:N
DETAILS                            680           0           0  N
                DETAILS            680           0           0      Y   Y    Ins:Y Upd:N Del:N
[3 excluded tables suppressed.]
-------------------------------------------------------------------------------
```

FIGURE 24. Example: Showing Deltas in the Log

To reset the to the cumulative default, use[2]

```
$ Define JCC_LOGMINER_LOADER_STAT_TYPE CUMULATIVE
```

1. See “Table Activity Reflected in the Log” on page 360.
2. Actually, any value except one starting with “D” will reset the logical name to the default.

Process Failure Reflected in the Log

When either the CLM or LML process fails, the CTL (control) and LML processes translate the exit status to a message and print it to their respective log files. The CTL process also translates the OpenVMS return status for the failed process. The processes will, then, suggest that the user view the actual message in the relevant log file.

```
%dba_clm_ast:  CLM process exited with an unexpected status %RMU-F-NOMSG, Message
number 02C8AB24

%dba_clm_ast:  See the CLM logfile (jcc_tool_logs:jcc_run_clm-SUBR09_T.log) for
more detailed information about this exception.

o
o
o

%jcc_continuous_logminer_loader: exit status
   %RMU-F-NOMSG, Message number 02C8AB24
%RMU-F-MSG, Message number 02C8AB24
$error_exit:
$!
$ save_status = $status
$!
$ jcc_find_message 46705444.

Looking up message number 46705444. - 02C9AB24 (HEX) 00262125444 (8)

This error message was found in SYS$COMMON:[SYSMSG}RMUMSG70,exe;8

%RMU-F-AIJSBQAFT, incorrect AIJ file sequence !SL when !UL was expected

o
o
o
```

FIGURE 25. Example of Reporting Process Failure

Table Activity Reflected in the Log

To cause the Loader threads to periodically display a summary of activity in the log, define the logical name

```
jcc_logminer_loader_stat_interval
```

to the number of seconds that should occur between logging events.

The default value is zero, which disables the feature. The maximum value is 86400 seconds. Invalid values are presumed to be zero and disable the feature.

When table activity logging is enabled, each Loader of a Loader family will, at the specified interval, log the number of rows encountered for each table on which it is configured to operate. The following provides an example.[1]

```
-Src----Table name-------Tgt- -------------Counts------------- ----------Flags--------
                             ---Modify- --Delete-- -Filtered- DBK Len IgnD  Action
JOBS                                 1          0          0  N   Y   Y    REPLICATE
                        JOBS         1          0          0  N   Y   Y    REPLICATE
DEPARTMENTS                          1          0          0  N   Y   Y    REPLICATE
                 DEPARTMENTS         1          0          0  N   Y   Y    REPLICATE
JOB_HISTORY                          1          0          0  N   Y   Y    REPLICATE
                 JOB_HISTORY         1          0          0  N   Y   Y    REPLICATE
WORK_STATUS                          0          0          0  N   Y   Y    REPLICATE
                 WORK_STATUS         0          0          0  N   Y   Y    REPLICATE
SEMA4_SAL_LEAVE_AUTH                 1          0          0  Y   Y   Y    REPLICATE
       SEMA4_SAL_LEAVE_AUTH          1          0          0  Y   Y   Y    REPLICATE
DEGREES                              1          0          0  N   Y   Y    REPLICATE
                    DEGREES          1          0          0  N   Y   Y    REPLICATE
RESUMES                              0          0          0  N   Y   Y    REPLICATE
                    RESUMES          0          0          0  N   Y   Y    REPLICATE
COLLEGES                             1          0          0  N   Y   Y    REPLICATE
                   COLLEGES          1          0          0  N   Y   Y    REPLICATE
SALARY_HISTORY                     203          0          0  N   Y   Y    REPLICATE
             SALARY_HISTORY        203          0          0  N   Y   Y    REPLICATE
CANDIDATES                           2          0          0  N   Y   Y    REPLICATE
                 CANDIDATES          2          0          0  N   Y   Y    REPLICATE
EMPLOYEES                            1          0          0  N   Y   Y    REPLICATE
                  EMPLOYEES          1          0          0  N   Y   Y    REPLICATE
```

FIGURE 26. Table Activity Reflected in the Log

Filtering Reflected in the Log

Filter is a keyword that provided an early way of including or excluding rows from the target based on certain limited criteria. FilterMap is a more sophisticated way to control what is and is not included in the target. FilterMap analyzes an SQL predicate and includes rows that satisfy the predicate. Filter is discussed on page 231 and FilterMap is discussed on page 233.

When analyzing an issue, it can be important to see which rows were excluded or included as part of a filter.

The keyword Logging (also in the Control File chapter) makes it possible to set a wide range of logging options.

To turn on logging of rows excluded, add the following to the Control File:

1. Spacing in the example is slightly modified for presentation purposes.


```
Logging~Input~Filter
```

This addition causes excluded rows (those that were input to the Loader and not output to the target) to be written to the log file.

To turn on logging of rows included in the output to the target, add the following to the Control File

```
Logging~Output~Filter
```

This will turn on logging of all rows which are published to the target.

Note: If both are used, the log will show one entry for every record that is tested by a filter. The log will, however, indicate whether the record is included or not. Example output for each is shown here:

```
FilterRemoved> Commit TAD: 26-MAR-2003 17:13:18.30 Read TAD: 16-SEP-
2015 14:57:44.29 tsn: 481 LSN: 795 action: M table: COLLEGES dbkey:
81:10:15
COLLEGES record (81:10:15) skipped due to data FilterMap
  where 25 > LOADER_SEQUENCE_NUMBER
         LOADER_SEQUENCE_NUMBER = 795

...

FilterPassed> Commit TAD: 26-MAR-2003 17:13:18.30 Read TAD: 16-SEP-
2015 14:59:43.77 tsn: 481 LSN: 4 action: M table: COLLEGES dbkey:
81:10:15
COLLEGES record (81:10:15) passed FilterMap
```

Heartbeat Reflected in the Log

Beginning with Version 3.5 of the JCC LogMiner Loader, the logging was improved for the use of heartbeat. The improvement is to add to the log when heartbeat begins and when it ends.

When heartbeat begins the log will include the time stamp and

```
Heartbeat update start
```

When heartbeat ends, one of two messages will occur in the log. Which one occurs depends on the version of Rdb, since earlier versions do not support returning the TSN (transaction sequence number). Either message begins with the time stamp. The messages are:

```
Heartbeat update committed
Heartbeat update (TSN <tsn>)committed
```

In the second case, '<tsn>' will be replaced with the TSN of the heartbeat transaction.

JDBC Exceptions in the Log

JDBC drivers vary in how target exceptions are reported. Some provide a single exception message that reports the cause of the problem; others provide a list of exceptions which, taken as a whole, report the problem.

The JCC LogMiner Loader Version, beginning with version 3.5, reports the entire list of exceptions reported by a JDBC driver.

AIJ Switches Reflected in the Log

Knowing which AIJ file is being processed can be important. The Loader thread log provides information which can help to understand where the Loader is in its processing. It is, however, slightly indirect. The Loader thread log reveals the AIJ sequence number from the Micro Quiet Point of the AERCP in the current commit record. The Micro Quiet Point may trail the journal that is currently being processed. Whether or not it does is dependent on the update activity in the source database. See also "Quiet Points and AIJs" on page 43.

On the first commit record that the Loader reads from the CLM process, the Loader will emit a message to the log file of the form:

```
27-MAR-2003 11:29:24.83 2022DD03 ||0 REGTESTRDB     Starting read
from AIJ sequence number 1
```

On reading each subsequent commit record from the CLM process, the Loader tests whether the AIJ sequence number has changed from the previous commit record. If it has, then the Loader reports the change. The message emitted to the log file is of the form:

```
27-MAR-2003 11:42:41.25 2022DD03 ||0 REGTESTRDB     Switching read
from AIJ sequence number 1 to 2
```


Intentional Throttling and the Log

See “Throttling the Loader” on page 480 for a discussion of fixed and realtime throttling. They are represented in the list of threads by ‘t’ and ‘T’. In addition, the full report includes the following section to report on realtime throttling.

```
                                    RealTime Wait
---------------------------------- Total ----------------------------------
Count: 7862

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:00.00    0 00:00:00.00            0            0            0

Min    0 00:00:00.00    0 00:00:00.00            0            0            0
Max    0 00:00:03.13    0 00:00:00.00            0            0            0
1st    0 00:00:01.55    0 00:00:00.01            0            7            1

Tot    0 00:23:21.92    0 00:00:42.21            0           23            6


- - - - - - - - - - - - - - - - - - - - Rate - - - - - - - - - - - - - - - - - -
Count: 0

        Elapsed Time         CPU Time   Direct I/O Buffered I/O  Page Faults

Ave    0 00:00:00.00    0 00:00:00.00            0            0            0
```

FIGURE 27. Statistics Full Report Section that Shows Throttling

The log may also contain an exhaustive reflection of the effects of throttling.

```
26-SEP-2003 12:18:17.04 208004C4 LML REGTESTRDB   Output delay throttle
set to REALTIME
26-SEP-2003 12:18:19.72 208004C4 LML REGTESTRDB   Realtime throttle
wait 6.220287 seconds...
26-SEP-2003 12:18:26.06 208004C4 LML REGTESTRDB   Throttling too much
by 0.118321 seconds.  Adjusting...
26-SEP-2003 12:18:26.44 208004C4 LML REGTESTRDB   Realtime throttle
running late (-1.621780 seconds)...
26-SEP-2003 12:18:27.16 208004C4 LML REGTESTRDB   Throttling too little
by -10.460495 seconds.  Adjusting...
26-SEP-2003 12:18:27.27 208004C4 LML REGTESTRDB   Realtime throttle
wait 6.174160 seconds...
26-SEP-2003 12:18:33.51 208004C4 LML REGTESTRDB   Throttling too much
by 0.164199 seconds.  Adjusting...
26-SEP-2003 12:18:33.52 208004C4 LML REGTESTRDB   Realtime throttle
wait 0.560126 seconds...
26-SEP-2003 12:18:34.11 208004C4 LML REGTESTRDB   Throttling too much
by 0.022480 seconds.  Adjusting...
26-SEP-2003 12:18:34.25 208004C4 LML REGTESTRDB   Realtime throttle
wait is too long to wait (5524024.000000 seconds.)  Proceeding...
26-SEP-2003 12:18:34.30 208004C4 LML REGTESTRDB   Realtime throttle
wait is too long to wait (5275272.000000 seconds.)  Proceeding...
```

FIGURE 28. Effects of Throttling Shown in the Log

Target Latency Reflected in the Log

To help identify latency issues in the target, new logging was introduced with Version 3.2.4. The logical name JCC_LML_TARGET_LOG_THRESHOLD[1] can be set to a positive real value. If set, when an interaction with the target exceeds the value set, a message is written to the log file. The message will indicate that the threshold was exceeded and the duration of the event.

The exact text of the message varies depending on the type of target interaction. If the target interaction is based on a table row or set of rows, those rows are written to the log file. Therefore, the message may only describe the event itself — such as a database attach, commit, disconnect, etc. — or it may include several lines of detail about the exact database record or records involved in the event.

1. The logical name JCC_LML_TUXEDO_LOG_THRESHOLD is specific to the Tuxedo environment and is maintained for backward compatibility. If both are defined the more generic JCC_LML_TARGET_LOG_THRESHOLD will define the value used.

For the purposes of this logical name, the checkpoint reads and writes are considered part of the target, regardless of whether the checkpoint is a table in the target or a separate file.

The default value is 3600 which is one hour. The default is used if the logical name is not defined or is defined as zero or a non-numeric. If the value specified has more than seven digits of precision, the value is truncated to seven digits. If the value specified has more than two, but less than seven digits of precision, all of the digits are used, but no more than two digits are displayed.

Several parts of an example follow.

```
Logging threshold exceeded (   0 00:00:01.19) during processing of
'ExecuteMATERIALIZED_TABLE'
Commit TAD: 17-DEC-2009 09:03:49.34 Read TAD: 17-DEC-2009 09:07:36.95
tsn: 13988 LSN: 3250 action: M table: JCCLML$COMMIT
dbkey: 11792:1215168512:6547
[MATERIALIZED_TABLE->MATERIALIZED_TABLE]
[  1]          LOADER_SEQUENCE_NUMBER = 3250
[  2]                      LOADERNAME = REGTESTRDB
[  3]                  LOADER_VERSION = 03.02.04
[  4]           LOADER_LINK_DATE_TIME = 16-DEC-2009 18:18:47.72
[  5]                             TSN = 13988
[  6]                             TID = 6547
[  7]                             PID = 772819054
[  8]         TRANSACTION_COMMIT_TIME = 17-DEC-2009 09:03:49.34
o
o
^


Logging threshold exceeded (   0 00:00:04.22) during processing of
'Attach'
o
o
o



Logging threshold exceeded (   0 00:00:01.27) during processing of
'UpdateHW'
o
```

```
o
o
Logging threshold exceeded (   0 00:00:03.89) during processing of
'Commit'
o
o
o
Logging threshold exceeded (   0 00:00:01.94) during asynch call of
'DETAILS:1'
(FLDID(167772475))    5.465997696e+01
(FLDID(167772475))    3.677684326e+02
(FLDID(167772475))    1.691900787e+02
(FLDID(167772475))    7.587152100e+02
(FLDID(167772475))    1.292737427e+03
(FLDID(167772475))    3.251594925e+01
(FLDID(167772475))    3.194628906e+03
(FLDID(167772475))    2.726551514e+03
(FLDID(167772475))    2.247426758e+03
(FLDID(167772475))    1.694269409e+03
```

FIGURE 29. Multi-part Example: Showing Latency Reporting

Large Transactions Reflected in the Log

Large transactions can create resource demands. Improved logging helps identify issues. The line added to the log for large transactions is of the format

```
%d: LSN: %llu, PID: %08.8X, TID: %u, Records: %u, User: %s
```

For example,

```
4: LSN: 15820, TSN: 45626, PID: 202EB445, TID: 12002, Records: 12343,
User: JEFF3
```

The same output line may also be logged in other circumstances. The complete list of circumstances for including this in the logging is

- Logging~Input~Trace is enabled.
- Logging~Input~Synchronization is enabled.
- The number of rows in a transaction exceeds 10,000.

Sort Reflected in the Log

If NOSORT output logging is enabled, the log will show the benefits.

```
1-OCT-2003 12:19:31.44 208116A9 LML EXMP logging~output~NoSort
 o
 o
 o
1-OCT-2003 14:01:29.27 208116A9 LML EXMP Sort bypassed (200 rows) ...
```

Context Dependent Reporting

When exceptions are raised in the control process regarding the output of the log files for the Loader threads or the CLM process, one of the following lines is written. Which of the lines is written depends on where in the log file processing the exception was encountered.

```
Logging manager(dba_ol_start): Error on file <filename>
Logging manager(dba_ol_stop): Error on file <filename>
Logging manager(dba_ol_reopen): Error on file <filename>
```

Directory Placement of Log Files

Loader log files can be re-directed by defining the logical name JCC_TOOL_LOGS in the Loader process context.

Activation Log

There is a separate log file that records the starts and stops of all Loader families (that are using the same version of the Loader software). The name of this log file is jcc_tool_data:jcc_lml_activation.log.

Privileges and Re-tries

The Loader attempt to write to the activation log may provide the first sign that the account used to run the Loader has insufficient privilege. The default behavior of the Loader is to re-try. This is a valid response in many cases. However, if the Loader is attempting to write to the activation log and the account used to run the Loader has insufficient privilege, additional attempts are not going to achieve success. To avoid the appearance of the Loader's going into an infinite loop, the logical

name JCC_LML_ACTIVATION_LOG_ATTEMPTS was added to the Loader. The default is 200. It can be defined to whatever you like.

Example of the Activation Log

```
$ type/page jcc_tool_data:jcc_lml_activation.log
17-APR-2009 13:15:23.22 NodeName=JASON LoaderName=REGTESTAPIJ No AERCP LSN=0
$1$dga105:[dba_tools.base_level.][exe_ia64]jcc_continuous_logminer_loader.exe;91
$1$dga300:[regression_test.jason.regression_test_db]loader_regression_test.rdb;1
jcc_root:[jeff2.jcc.dba.regression_test.api]loader_regression_test_lm_unl.opt;1
jcc_root:[jeff2.jcc.dba.regression_test.api]loader_regression_test_control_api.ini;16
17-APR-2009 13:15:24.47 NodeName=JASON LoaderName=REGTESTRDBJ No AERCP LSN=0
$1$dga105:[dba_tools.base_level.][exe_ia64]jcc_continuous_logminer_loader.exe;91
$1$dga300:[regression_test.jason.regression_test_db]loader_regression_test.rdb;1
jcc_root:[jeff2.jcc.dba.regression_test]loader_regression_test_lm_unl.opt;1
jcc_root:[jeff2.jcc.dba.regression_test.rdb]loader_regression_test_control.ini;64
17-APR-2009 13:15:42.71 NodeName=JASON LoaderName=REGTESTORAJ No AERCP LSN=0
$1$DGA105:[DBA_TOOLS.BASE_LEVEL.][EXE_IA64]JCC_CONTINUOUS_LOGMINER_LOADER.EXE;91
$1$DGA300:[REGRESSION_TEST.JASON.REGRESSION_TEST_DB]LOADER_REGRESSION_TEST.RDB;1
JCC_ROOT:[JEFF2.JCC.DBA.REGRESSION_TEST]LOADER_REGRESSION_TEST_LM_UNL.OPT;1
JCC_ROOT:[JEFF2.JCC.DBA.REGRESSION_TEST.ORACLE]LOADER_REGRESSION_TEST_CONTROL_ORA.INI;70
17-APR-2009 13:27:52.91 NodeName=JASON LoaderName=REGTESTRDBJ SHUTDOWN %DBA-S-SUCCESS,
Routine completed successfully.
17-APR-2009 13:27:53.24 NodeName=JASON LoaderName=REGTESTAPIJ SHUTDOWN %DBA-S-SUCCESS,
Routine completed successfully.
17-APR-2009 13:27:53.70 NodeName=JASON LoaderName=REGTESTORAJ SHUTDOWN %DBA-S-SUCCESS,
Routine completed successfully.
```

FIGURE 30. Example of the Activation Log

Locking Diagnostic Tool

The JCC LogMiner Loader also includes a diagnostic tool to assist in reviewing the locks that the Loader acquires while performing the requested tasks.

Syntax

```
$ jcc_lml_show_locks [LoaderName] [-b[locking]]
```

[LoaderName] ***optional.*** LoaderName is optional, but may be used to limit the output to only one Loader, instead of all Loaders running on the system.

[-b[locking]] ***optional.*** When blocking is specified, only locks that are currently blocked by another process are shown.

Examples

Following are examples of the jcc_lml_show_locks utility using different parameters. Not all combinations of parameters are shown, but these should be illustrative.

```
$ jcc_lml_show_locks REGTESTRDB -blocking

JCC LogMiner Loader Show Locks D02.00.00 (built 22-JAN-2003 13:33:43.79)

--------------------------------------------------------------------------------
Loader Name: REGTESTRDB
Resource Name: Deadman
Granted Lock Count: 1, Parent Lock ID: 2B042132

          -Master Node Info-  --Lock Mode Information--     -Remote Node Info-
ProcessID Lock ID   SystemID  Requested Granted   Queue     Lock ID   SystemID
202073A5  770768AA  00010001            PW        Grant     770768AA  00010001
202077A6  7F0713DF  00010001  CW        NL        Convert   7F0713DF  00010001
```

FIGURE 31. Example of Show Blocked Locks for a Specific Loader

The next example is of the show locks command with no limit to a specific Loadername and no limit to blocked locks.

```
$ jcc_lml_show_locks

JCC LogMiner Loader Show Locks D02.00.00 (built 22-JAN-2003 13:33:43.79)

--------------------------------------------------------------------------------
Resource Name: Cluster Loaders
Granted Lock Count: 10

         -Master Node Info-  --Lock Mode Information--     -Remote Node Info-
ProcessID Lock ID   SystemID  Requested Granted   Queue     Lock ID   SystemID
20204683  22083CB3  00010001            CR        Grant     22083CB3  00010001
20206EA1  73008C5D  00010001            CR        Grant     73008C5D  00010001
20207DAC  7804DC33  00010001            CR        Grant     7804DC33  00010001
202076AF  5D00F675  00010001            CR        Grant     5D00F675  00010001
202073A5  3F01868C  00010001            CR        Grant     3F01868C  00010001
202077B6  5F00328E  00010001            CR        Grant     5F00328E  00010001
202076B0  3901A10F  00010001            CR        Grant     3901A10F  00010001
202067BC  64035717  00010001            CR        Grant     64035717  00010001
20206FD2  47078895  00010001            CR        Grant     47078895  00010001
20207ED4  51014581  00010001            CR        Grant     51014581  00010001
--------------------------------------------------------------------------------
Resource Name: REGTESTRDB
Granted Lock Count: 2, Parent Lock ID: 5F00328E

         -Master Node Info-  --Lock Mode Information--     -Remote Node Info-
ProcessID Lock ID   SystemID  Requested Granted   Queue     Lock ID   SystemID
202073A5  590529B4  00010001            PW        Grant     590529B4  00010001
202077B6  1200E13F  00010001            NL        Grant     1200E13F  00010001
--------------------------------------------------------------------------------
Resource Name: REGTESTORA
Granted Lock Count: 3, Parent Lock ID: 5D00F675

         -Master Node Info-  --Lock Mode Information--     -Remote Node Info-
ProcessID Lock ID   SystemID  Requested Granted   Queue     Lock ID   SystemID
20206EA1  7D0372CF  00010001            PW        Grant     7D0372CF  00010001
20207DAC  6103FD4E  00010001            NL        Grant     6103FD4E  00010001
202076AF  39078133  00010001            NL        Grant     39078133  00010001
--------------------------------------------------------------------------------
Resource Name: REGTESTAPI
Granted Lock Count: 2, Parent Lock ID: 3901A10F

                                    < and so forth >
```

FIGURE 32. Example of Show ALL Locks

For information on controlling locking levels, see “Interpretation of Lock Conflicts” on page 403.

Displaying Checkpoint Information

The JCC LogMiner Loader kit includes a procedure to format and display the stored checkpoint information. The procedure name is jcc_lml_dump_checkpoint. To invoke the procedure, use jcc_lml_dump_checkpoint.

Syntax

```
$ jcc_lml_dump_checkpoint <LoaderName> <name> [<type>]
```

Parameters

<LoaderName>. LoaderName is the LoaderName for the checkpoint.

<name>. Name is the checkpoint filename or the target (of the checkpoint) database or OCI service. (If an OCI service is specified, the dump checkpoint routine will prompt for proper credentials to access the target database.)

<type> ***optional.*** Type is the optional type of the checkpoint stream: LML_INTERNAL, OCI, or RDB (The default is LML_INTERNAL.)

Output

The formatted output of the JCC_LML_DUMP_CHECKPOINT procdure reports the checkpoint record that would be selected for restart at the time of the execution of the utility. Relevant checkpoint data is displayed in the formatted output. Each formatted item also has a DCL symbol created to make the information easily accessible from DCL.

For example, the checkpoint display might look like that shown in the following.[1]

1. Note that, if there is no AERCP and RM_TID, as will be true when commit records are not included in the Rdb LogMiner output, those symbols are set to the empty string in this display. Commit records may be omitted in the original static mode. They are always included when running in continuous mode.

$ jcc_lml_dump_checkpoint copyi loader_regression_test_db rdb

```
JCC LML Dump Checkpoint V03.05.00 (built 15-MAY-2017 13:38:08.34)

-- Checkpoint restart information --

--- Parallel mode ---
Write Timestamp:     30-MAY-2017 12:55:39.28
LoaderName:          COPYI
Completion Flag:     N
Checkpoint Interval: 1
Input Data Source:   LML_CONT_COPYI
Last Transaction:
        Start Time:  30-MAY-2017 12:42:16.67
        Commit Time: 30-MAY-2017 12:42:16.68
        TSN:         145777
        LSN:         39376
        AERCP:       1-28-21-351436-145777-145777
        RM TID:
```

The associated DCL symbols would be those shown here.

```
$ show sym jcclml$*
  JCCLML$AERCP == "1-28-21-351436-145777-145777"
  JCCLML$CHECKPOINT_INTERVAL == "1"
  JCCLML$COMMIT_TAD == "30-MAY-2017 12:42:16.68"
  JCCLML$COMPLETION_FLAG == "N"
  JCCLML$INPUT_SOURCE == "LML_CONT_COPYI"
  JCCLML$LOADERNAME == "COPYI"
  JCCLML$LSN == "39376"
  JCCLML$RM_TID == ""
  JCCLML$START_TAD == "30-MAY-2017 12:42:16.67"
  JCCLML$TSN == "145777"
  JCCLML$WRITE_TAD == "30-MAY-2017 12:55:39.28"
```

The meaning of the DCL symbols is shown in the chart to follow.

TABLE 5. DCL Symbols Created by JCC_LML_DUMP_CHECKPOINT

Symbol	Formatted Label	Meaning
JCCLML$AERCP	AERCP	AERCP stored in the checkpoint record
JCCLML$CHECKPOINT_INTERVAL	Checkpoint Interval	Configured checkpoint interval for the Loader
JCCLML$COMMIT_TAD	Commit Time	Commit timestamp stored in the checkpoint record
JCCLML$COMPLETION_FLAG	Completion Flag	Completion flag stored in the checkpoint record
JCCLML$INPUT_SOURCE	Input Data Source	Input source configured for the Loader
JCCLML$LOADERNAME	Loader-Name	Loader Name configured for the Loader
JCCLML$LSN	LSN	Loader Sequence Number stored in the checkpoint record
JCCLML$RM_TID	RM TID	Resource Manager Transaction ID stored in the checkpoint record
JCCLML$START_TAD	Start Time	Start transaction timestamp stored in the checkpoint record
JCCLML$TSN	TSN	Transaction Sequence Number stored in the checkpoint record
JCCLML$WRITE_TAD	Write Time-stamp	Timestamp for when the check-point record was written

Gather Database Information

The JCC_GET_DB_INFO procedure captures Rdb database information as DCL symbols.

Syntax

```
$ jcc_get_db_info <database-root>
```

DCL Symbols

This procedure creates the following DCL symbols:

TABLE 6. DCL Symbols Created by JCC_GET_DB_INFO.COM

Symbol	Possible Values
rdb_database_name	Expanded name of database root, blank if database does not exist
rdb_database_open	True, False
rdb_database_operator_open	True, False
rdb_database_open_timestamp	Date-time database was opened, in OpenVMS format
rdb_hotstandby_status	active, pending connection, ""
rdb_hotstandby_master	Name of the master database
rdb_hotstandby_standby	Name of the standby database
rdb_database_is_master	True, False, ""
rdb_database_is_standby	True, False, ""
rdb_hotstandby_remote_node	Node name of remote node
rdb_version_set	True, False, ""

Example

```
$ @JCC_GET_DB_INFO SYS$SYSDEVICE:[KEITH.MFP]MF_PERSONNEL.RDB;1
$ sho sym rdb*
  RDB_DATABASE_IS_MASTER == "True"
  RDB_DATABASE_IS_STANDBY == "False"
  RDB_DATABASE_NAME == "SYS$SYSDEVICE:[KEITH]MF_PERSONNAL.RDB;1"
  RDB_DATABASE_OPEN == "True"
  RDB_DATABASE_OPEN_TIMESTAMP == "16-JAN-2003 12:40:32.60"
  RDB_DATABASE_OPERATOR_OPEN == "True"
  RDB_HOTSTANDBY_ACTIVE == "False"
  RDB_HOTSTANDBY_MASTER == "SYS$SYSDEVICE:[KEITH]MF_PERSONNEL.RDB;1"
  RDB_HOTSTANDBY_REMOTE_NODE == "ATLAS::"
  RDB_HOTSTANDBY_STANDBY == "JCC_ROOT:[KEITH]MF_PERSONNEL"
  RDB_HOTSTANDBY_STATUS == ""
```

FIGURE 33. Example Output of JCC_GET_DB_INFO.COM

Gather Loader Information

The JCC_GET_LOADER_INFO.COM procedure captures information about a JCC LogMiner Loader session as DCL symbols.

Syntax

```
$ jcc_get_loader_info <Loadername>
```

DCL Symbols

This procedure creates the following DCL symbols: Example

TABLE 7. DCL Symbols Used by JCC_GET_LOADER_INFO.COM

Symbol	Possible Values
jcc_lml_name	LogMiner Loader session name
jcc_lml_active	True if a LML session for jcc_lml_name is active in the cluster, else False
jcc_lml_active_node	Name of the node on which the jcc_lml_name is active.

Example

```
$jcc_get_loader_info REGTESTORA
$show symbol jcc_lml*
JCC_LML_ACTIVE == "True"
JCC_LML_ACTIVE_NODE == "ATLAS"
JCC_LML_NAME == "RESTESTORA"
```

FIGURE 34. Example Output for JCC_GET_LOADER_INFO.COM

Get the Current AIJ Sequence Number

Since it is essential not to remove AIJ backup files from the system until the LogMiner has processed them, it is important to know the current AIJ sequence number. To do so, use these Oracle Rdb RMU commands.

```
$ rmu/show after/back/noout <db name>
$ show symbol rdm*
  RDM$AIJ_COUNT == "10"
  RDM$AIJ_CURRENT_SEQNO == "1221"
  RDM$AIJ_ENDOFFILE == "533768"
  RDM$AIJ_FULLNESS == "17"
  RDM$AIJ_LAST_SEQNO == "1220"
  RDM$AIJ_NEXT_SEQNO == "1221"
  RDM$HOT_STANDBY_STATE == "Inactive"
  RDML == "$RDML"
```

FIGURE 35. Display the Current AIJ Sequence Number

Note that you can also get the Checkpoint information (“Displaying Checkpoint Information” on page 372) or you can build into your backup procedures the safety test (“Safety Test for AIJ Backup” on page 418) described in the chapter for Loader Administrators.

Operator Classes and OPCOM Messages

JCC's LogMiner Loader provides control over where two sorts of OPCOM messages are sent. The operator classes supported are

- cards
- central
- cluster
- devices
- disks
- license
- network
- security
- tapes
- oper1, oper2, ..., oper12

Loader Failure

The keyword OPERATOR can be used to set one or more operator classes to receive failure messages. The default is central. Any number of classes can be specified in a comma separated list or ALL may be specified.

OPCOM messages generated by the license and command line validation routines are generated before the Control File is processed. Therefore, these messages will be sent to ALL operator classes.

See also the "Keyword: Operator" on page 274.

Tardiness Messages

The Loader statistics program generates messages if a set threshold is reached for "tardiness." Tardiness is defined as being more than the number (specified as the tardiness threshold) of seconds lag between the update on the source and passing the changes to the target. See "Tardiness Threshold optional" on page 315. A message is generated when the Loader catches up again.

By default, tardiness messages are sent to the CENTRAL operator class. A parameter enables setting the operator class to any desired collection or to set the operator class to ALL. The format is

```
jcc_lml_statistics <LoaderName>
                   [refresh seconds]
                   [brief|full|detail|csv]
                   [tardy threshold[operator class]]
```

The same set of operator classes are available as for failure messages.

See also “Operator Classes and Tardiness Messages” on page 457.

CHAPTER 15

Performance Considerations

The JCC LogMiner Loader faces a significant performance challenge. The entirety of the database update process — which was done by an entire set of application processes — is to be accomplished through the agency of a single database update engine. This is likely to be resource intensive.

The Loader is well-equipped with tuning options to help you meet your performance needs.

This chapter includes notes on both automatic performance tuning and tuning alternatives that are available to the Loader Administrator.

Topics

Tuning will consist of several dimensions. This chapter discusses some options.

- Primary Loader Tuning Options: These are the fundamental tools that the Loader makes available to the Loader Administrator to improve performance.
  - Parallelism through multiple Loader threads (“Parallelism and Loader Threads” on page 383)
  - Pseudo-parallelism through separate Loader families for separate tables (“Interpretation of Lock Conflicts” on page 403)
  - Checkpointing (control of the commit interval) (“Interpretation of Lock Conflicts” on page 403)
- Systems Tuning for Loader Use: There are systems options available that can enhance performance when tuned for the specific situation.
  - I/O Management for OpenVMS (“I/O Management” on page 393)
  - Tuning process quotas for OpenVMS (“Process Quotas” on page 394)
  - CPU Requirements (“CPU Requirements” on page 395)
  - Sortwork file control for the LogMiner (“Performance Improvements in the Loader” on page 407)
  - Using 64-bit memory (“Performance Improvements in the Loader” on page 407)
  - Controlling the buffer count
- Tuning for the Target: The Loader cannot run faster than the target can absorb the data. Overall system performance will also be dependent on an understanding of the target chosen.
- Use of the VMS Lock Manager: The Loader automatically adapts use of the VMS lock manager to a variety of situations. The Loader Administrator can control some things.
- Additional Loader approaches that improve performance.

For related topics, see also portions of “Aids for the Administrator” on page 409.

Parallelism and Loader Threads

Multi-threading is a powerful tool to boost the performance of the Loader for certain applications. Multi-threading uses multiple simultaneous threads to move data to the target efficiently. [1]

- Multi-threading can be dynamic. The number of threads can change in response to the workload.
- Tuning of a multi-threaded use of the Loader is possible by adjusting the Parallel and Threads keywords in the Control File.
- Multi-threading can be controlled manually, if required.
- Multi-threading, by default, is set to avoid buried updates; but other choices can be specified with the Parallel keyword.
- Additional tuning of multi-threading for extremely large transactions is available.
- Additional attention to details has further improved multi-threading performance with many of the releases since its introduction in Version 2.0.
- As with any tuning, some experimentation may be required to achieve the correct balance for a particular set of circumstances.

Requirements and Options

Using the multi-threaded option, requires certain settings.

- The Parallel keyword *must* be specified in the Control File.
- The Thread keyword *may* be specified in the Control File.
- The Input keyword *must not* be set to ASYNCH.
- The input type for the Input keyword *must* be IPC (mailbox).
- The Input_failure keyword *may* be used to set timeouts.[2]
- To support multi-threading, the Loader highwater table will contain a row for *each* thread. [3]

1. Multi-threading, which has been available since Version 2.0 of the Loader, does not use the VMS p-threads. Loader threads indicate separate processes, running in parallel.
2. Note that this keyword may also be used with single threaded Loaders.

Control File

Keywords used to establish multi-threading are Parallel and Thread. Parallel is required for multi-threading; Thread is optional. See “Keyword: Parallel” on page 270 and “Keyword: Thread” on page 284.

Automatic and Dynamic Adjustments to the Number of Threads

The Control Process (CTL) for the Continuous LogMiner Loader manages the family of threads in a session. At the beginning of the Loader session, the minimum number of threads[1] is started.

When the CTL process obtains the mailbox READ lock, the startup timer begins.[2] If no currently running thread requests the READ lock before the timer advances to the specified ‘startup interval’ and the thread count is not at the maximum, the CTL process will start another thread.

Similarly, when the CTL process releases the lock to a currently running Loader thread, the shutdown timer begins. If the CTL does not obtain the READ lock again before the timer has advanced to the specified “shutdown interval” and the number of threads is not at the minimum, CTL requests that one of the threads shutdown.

In this way, the number of threads running changes dynamically with the workload that is coming from the Continuous LogMiner.

Disabling Dynamic Adjustments to the Threads

Setting the minimum threads equal to the maximum threads disables dynamic thread adjustment.

---

3. If you started using the Loader with Version 2.0 or later, this structure for the highwater table is automatic. If you started with a version prior to 2.0, before using multi-threading, use the procedure

   ```
   convert_loader_hw_1_2_to_2_0.sql
   ```

1. The minimum and maximum number of threads are set with the Parallel keyword in the Control File.
2. The interval for startup (and for shutdown) is set with the Thread keyword in the Control File.

Manually Starting and Stopping Threads

Two procedures are supplied which allow the Database Administrator to manually adjust the number of threads that are running. These are JCC_CLML_START_-THREAD and JCC_CLML_STOP_THREAD. These procedures should be invoked with one parameter which is the name of the Loader family for which the thread should be started or stopped. One thread is started or stopped with each procedure invocation. You may issue these commands repeatedly without waiting for the requested actions to complete.

You cannot use these commands to reduce the number of threads below the minimum specified for the family or increase the number of threads beyond the maximum specified for the family.

Manual adjustments can be useful in cases where the startup and shutdown intervals may have been mis-estimated or in special circumstances, such as an unexpected load.

With Version 3.1.0, manually stopping threads became more immediate. With that and later versions, stop thread requests are honored regardless of the workload.

Altering Minimums and Maximums On-Line

The parallel keyword enables specification of a minimum number of threads to run and of a maximum number of threads to run. Keywords are set in the Control File.

On occasion, Administrators using the Loader have wanted to be able to modify the minimum and maximum defined in the Control File *while the Loader is running.* If the parallel keyword is defined in the Control File, the minimum and maximum can be changed on-line with

```
JCC_CLML_minimum_threads <LoaderName> <new minimum>
JCC_CLML_maximum_threads <LoaderName> <new maximum>
```

If the parallel keyword has not been defined in the Control File, thread counts cannot be changed this way.

The Thread Log Files

The Control Process (CTL) of the Loader acts as a logging sink for Loader activity and for Rdb Continuous LogMiner activity. Every thread startup will either start a new log file or append to an existing log file.[1] The CTL maintains these. The

Loader and LogMiner logs are stored in the directory JCC_TOOL_LOGS and incorporate the Loader name as part of the file name.

The procedure JCC_CLML_REOPEN_LOG causes the control process to close all existing logs and to re-open them. This procedure requires one parameter which is the Loader family name.

Statistics for Tuning

There are statistics options available to aid your tuning.

The best way to set the available parameters for multi-threading the Loader will vary with the application. You will probably need to try a set of values and, then, tune as you get experience.

For example, for an application that trickles updates to the database for most of the day, but has an important peak of intense writing, you may want to set the minimum to one and the maximum to thirty-two and try a startup interval of five seconds and a shutdown interval of fifteen seconds. For performance reasons, you might also increase the commit interval with the checkpoint keyword and use the input timeout keyword to ensure that, during low activity, updates are transmitted at satisfactory rates.

For less volatile databases, you will want a smaller maximum, say eight threads.

System resources may also constrain you. Process quotas, maximum processes on the system, the type of license you have for the target, CPU capability, or other things may suggest less exuberant use of threads. Further, choices for minimum and maximum threads must be balanced with other settings. See “I/O Management” on page 393.

A good measure of Loader throughput is the reported value of trailing time in all statistics reports. However, note that, if you are using a large commit interval and a large timeout,[1] then the trailing value can be artificially inflated.

1. The choice of a log file per thread or appending to an existing log file is made with the logical name JCC_CLML_logging_style. See also “The Log Files” on page 356.
1. See “Keyword: Checkpoint” on page 231 and “Keyword: Output_failure” on page 278.

Consistency Modes Used with Parallel Threads

An optional parameter to the parallel keyword enables specification of the "mode." Options for consistency modes are constrained, unconstrained, and automatic. Constrained is the default. Constrained consistency mode serializes updates to individual rows so that more recent changes to the source are not overwritten by earlier changes. Unconstrained consistency mode is available to allow the target application to control the consistency issues. Automatic consistency mode uses consistency mode unconstrained for tables that are set for insert only and uses consistency mode constrained for all others.[1]

The sections, on additional performance considerations for multi-threading, all relate to constrained consistency mode (or to constrained use in automatic consistency mode).

Pseudo-Parallelism and Separate Loader Families

Before the Loader supported true parallelism, it was possible to achieve a semblance of parallelism. The approach provided looser transaction control.

There is less reason to use this approach with true parallelism available. It is included here, in case you have a need for it or have an existing use of it. Under certain circumstances, JCC Loader support may recommend it. For example, it can be useful in certain Data Pump scenarios.

If you can afford to break the strict conformance of transactions in the source database mapping to transactions in the target database, throughput to the target can often be improved by creating multiple Loader families.[2]

Each Loader family will maintain its own distinct set of tables. The result is pseudo-parallelism. The multiple families maintaining the target are each specialized through multiple Control Files to define different Loader families. The net result is the ability to deploy more CPUs and to multi-thread networks and I/O

1. For additional material on consistency mode, see "<consistency mode> optional" on page 281.
2. The use of the term Loader Family refers to the group of processes - parent (CTL), CLM, and LML threads - for a single Loadername.

operations, but without the transactional consistency provided by the true multi-threaded Loader.

It is also possible to use multiple Loader families and to have some or all of those Loader families use the Loader's parallelism.

Commit Interval

The Loader, with the checkpoint keyword, permits you to set a commit interval greater than one. With a commit interval greater than one, the Loader collects several transactions before passing them on. This can be a performance enhancing feature.

If Loader commit intervals are too small, excessive I/O can result. If they become too large, excessive numbers of locks will be required.

page 409There are several circumstances, however, in which checkpointing does not give the desired results. For these circumstances further refinements are provided. In the following subsections, examine the complications that can arise and the tools for addressing them:

- Input Read Timeout - prevent stalls in systems in uneven loads
- "No Work" Transactions - prevent counting irrelevant transactions
- Input Buffer Threshold - automatic adjustment for varying workloads
- Stale Timer - switches automatically between approaches designed for best runtime and best restart, depending on load
- Interaction with other tuning options (See "I/O Management" on page 393.)

Input Read Timeout for Checkpointing

With a checkpoint interval greater than one, if there is a point at which the updates to the application become infrequent, a stall can occur. Since there are no transactions to fill the commit interval, no commits occur. Therefore, in systems with uneven loads, using CLML and a longer commit interval can precipitate a significant and artificial delay for some transactions.

You can specify an input timeout. The input timeout should be set to the amount of time that it is acceptable for the Loader to wait for input before it checkpoints information that is already buffered. This value is set in the Control File with the keyword Input_Failure. If that keyword is not defined *and* the logical name JCC_LOGMINER_LOADER_STALE_INTERVAL is defined, the value defined for the logical name will be used as the number of seconds to wait. [1]

See also "Keyword: Input_failure" on page 248.

No Work Transactions and Checkpoint Intervals

For Loader purposes, "no work" transactions are those that contain a commit record, but no rows that meet the criteria for loading. "No work" transactions can interfere with the normal operation of checkpointing.

By default, the Loader counts "no work" transactions toward a checkpoint interval. These "no work" transactions may cause the Loader to checkpoint more frequently than necessary. For some configurations of the Loader, this would mean that the checkpoint interval had to be set artificially high in order to avoid checkpointing too frequently due to the large volume of these "no work" transactions.

The Loader has a mechanism for excluding these "no work" transactions from the checkpoint count. This uses the same keyword that resolves the issue of infrequent commits. If you set the input_failure[2] keyword in the Control File to a positive value, it not only enables input timeouts, it also excludes "no work" transactions from the count.

A value of zero for the input_failure keyword (or leaving the keyword out) continues the default of counting "no work" transactions toward the commit interval.

See also "Keyword: Input_failure" on page 248 and "Input Read Timeout for Checkpointing" on page 388.[3]

1. Note that the Loader always commits or writes to the checkpoint file on transaction boundaries. The Loader, even with the input read timeout, will never commit less than a full transaction. The LogMiner does not provide information to the Loader until the commit and the Loader does not segment transactions.
2. The syntax is Input_failure~<seconds>.

Input Buffer Threshold for Checkpointing

Initially, the Loader would only checkpoint when it reached the declared checkpoint interval. For some systems, setting the commit interval appropriately was difficult because the number of rows included in a transaction could vary dramatically with the time of day.

A threshold concept has been introduced. If the number of records in the input buffer exceeds the threshold, the Loader will commit at the next transaction boundary. Under these circumstances, the Loader may be committing earlier than the specified commit interval.

This threshold cannot be set by the user. Instead, the Loader dynamically tunes it to the workload. The input buffer threshold is initially set to 50 * the checkpoint interval. That is, if the checkpoint interval is 10, then the input buffer threshold will initially be set to 500 records (10 * 50 = 500). After each checkpoint, the Loader will calculate a new value for the input buffer threshold as 150% of the average records per output transaction.[1]

If the calculated input buffer threshold is less than 10 * the checkpoint interval, the input buffer threshold is set to 10 * the checkpoint interval. In the example, that minimum setting would be 100. If, instead, the first output transaction[2] includes 200 record insert/update/deletes, then the Loader input buffer threshold will be set to 300. [3] If the second transaction includes 100 record insert/update/deletes, then the Loader input buffer threshold will be modified to 225. [4]

---

3. The operation of the input_failure keyword is tuned to address environments with bursts of transactions with work to pass to the target, followed by no work transactions. A timer is started when the Loader receives the first commit and the time is tested after each commit to determine whether the timeout threshold is exceeded. When the threshold is exceeded, the Loader performs a timeout checkpoint with all currently buffered data. Timeouts are approximate. Since they are checked only on the commit, the total time between checkpoints may vary from just over the threshold to nearly twice the threshold.

1. The input buffer threshold is not recalculated after a checkpoint that is caused by the input read timeout feature. However, checkpoints caused by the input buffer threshold are included in the calculations.
2. Note that an output transaction may represent multiple input transactions. How many is controlled with the Commit keyword.
3. (200/1) * 1.5 = 300
4. ((200+100)/2) * 1.5 = 225

As can be seen, this algorithm will, gradually, settle on a threshold of about 150% of the number of rows in an output transaction. If the profile of transactions changes, then this number gradually adjusts.

For a parallel Loader thread that is re-using a slot that was previously used by a thread which was shutdown, the new thread will start using the same values with which the previous thread shutdown. When the parent process is shutdown[1], the thread loses all context of the records/transaction history. It will re-start with the initial default of 50 * the checkpoint interval.

An example of an environment for which this enhancement is valuable is one that includes routine processing for over two thirds of the day, but for a few hours in the middle of the night has updates to the source database occur in large batches.

Note: If the Loader checkpoints early due to this input timeout, it will still checkpoint at the declared interval. Also, the Loader will never commit less than a full source database transaction.[2]

Stale Timer

Concern for performance includes concern for the best runtime performance and concern for the best restart performance. The optimistic runtime setting requires the least number of checkpoints to the checkpoint target, but requires that more transactions be resent on restart. The optimistic restart setting requires more checkpoints to the checkpoint target, but requires fewer transactions be resent on restart. The Loader switches between the two because the longer a thread must wait to read more data from the LogMiner mailbox, the more 'stale' the data previously checkpointed becomes.

If a thread has waited for the LogMiner mailbox lock (read lock) longer than the 'stale' timer interval, the Loader will switch from optimizing the runtime to optimizing the restart and will write its checkpoint. When the thread does start to read more data, it will switch back. To switch back requires modifying the checkpoint, again, while still holding the mailbox lock. This ensures that the Loader family can

1. Shutting down the parent process (or control process or CTL) shuts down the full Loader session.
2. The LogMiner does not provide information to the Loader until the commit and the Loader does not segment transactions.

restart from the transaction which the thread just read. The thread can then release the mailbox lock and continue.

The default value for the 'stale' timer is 0.5 seconds. The Administrator can set it to another value with the logical name JCC_LOGMINER_LOADER_STALE_INTERVAL

See also "Input Read Timeout for Checkpointing" on page 388 and "I/O Management" on page 393.

Tuning the Checkpoint

As in any tuning effort, sometimes increasing one parameter requires adjustment in others and options designed to enhance performance can, in combination, achieve the opposite. This documentation reflects as many of these as possible. However, which choices for options and settings will provide the best performance is dependent on the choices made for application architecture and the environment in which it is implemented.

For example, if the minimum number of threads is set too high for the workload, triggering the 'stale' timer is virtually guaranteed. Triggering the timer necessitates additional writes of the checkpoint data which, for this scenario, decreases performance. The effect is increased by any of several factors. If the workload is single row transactions, the read lock is required more frequently for the amount of work done and, thus, increases the likelihood of triggering the 'stale' timer. If the target database is remote and the network time approaches or exceeds the time necessary to trigger the 'stale' checkpoint, additional work to update the checkpoint is effectively required.

To improve performance in this situation there are several alternatives.

- Reduce the minimum number of threads for the Loader family, such that they are created as needed. For this, it may also be preferable to reduce the thread startup.[1]
- Increase the 'stale' timer to exceed the amount of time necessary to update the target. The logical name JCC_LOGMINER_LOADER_STALE_INTERVAL changes the 'stale' timer interval.[2]

1. See "Keyword: Parallel" on page 279.
2. See "Interpretation of Lock Conflicts" on page 403.

- Reduce the time necessary to checkpoint. If modifying the workload or the network is not possible, changing the checkpoint target may help.[1] However, there are disadvantages to doing so.

Commit Interval and Batch Options for Some Loader Targets

Some options for the Loader target support batching transactions. The use and tuning is similar to that for the commit interval. Loader commit intervals and Loader target batch intervals can be used together.

I/O Management

I/O Management presents different issues with different Loader choices.

Reducing Buffer Count

The logical name RDM$BIND_BUFFERS can be used to set the Rdb database buffer count. The JCC LogMiner Loader, beginning with version 3.6, sets this logical name to limit the number of buffers allocated to five for the source database attach. Reducing the buffer count to five for the source attach is appropriate because the Loader CTL (control) process does not require many database buffers to perform its work. A lower buffer count is sufficient for the source attach and the work associated with heartbeat.

Rdb Targets and Buffers

When Rdb is the Loader target, I/O control can be achieved by using large numbers of database buffers. Large here should be construed as thousands or tens of thousands of buffers. Large numbers of global buffers are likely to be most beneficial.

If your installation uses large commit intervals, then there is a good likelihood that I/O tuning parameters such as clean buffer counts and asynchronous batch write blocks will also be important.

1. See “<checkpoint stream type> optional” on page 232.

The buffer count can be redefined with RDM$BIND_BUFFERS by the Loader Administrator as appropriate to the Loader session. The use is specific to Rdb Loader targets that are local to the machine on which the Loader session is running.

It is important to recognize the trade-off involved in increasing buffers, as too large a number can cause a process to page fault. Extensive page faulting can cause a process to slow down, sometimes significantly.

It is also important to note that, at this time, there is only one place to define the buffer count. Reducing the count for the source and increasing it for the target is not possible.

Checkpointing to a File

Targets other than Rdb and Oracle require checkpointing to a file. For these targets, it is important to recognize that the disk hosting the checkpoint file is likely to see frequent activity.

Process Quotas

Large commit intervals and large numbers of global buffers suggest that the Log-Miner Loader process will require large process quotas. Quotas to examine are:

- ENQLM
- WSQUOTA
- WSEXTENT
- PAGFILQUO
- BYTLM

Not all of these are discussed in this document.

Page File Quota

In order to support recovery from failure, the Loader retains all input rows in dynamic memory for a particular transaction in the target database. Since many source database transactions may be packaged into a single target database transaction, this may result in the local caching of a considerable number of rows. The size

of this cache can be exacerbated by the execution of large complex transactions in the source database.

The Loader can use significant amounts of memory in some cases. For instance, if it encounters a large transaction, changing many rows, those rows are read into memory and, optionally, sorted before being sent to the target. This can use significant amounts of memory.

If the commit interval is set high, many rows can end up buffered in Loader memory.

It is appropriate, therefore to ensure that processes have enough page file quota.

If the page file quota is strained, it is also appropriate to consider reducing the Loader commit interval through use of the CHECKPOINT~<n> parameter in the Control File. See “Keyword: Checkpoint” on page 231.

CPU Requirements

Each thread of the JCC LogMiner Loader is, itself, an inherently single threaded process. With sufficient process and I/O tuning this process is likely to become CPU bound. Some CPU performance opportunities include:

- Process quota tuning
- Buffering objects
- Rdb buffer management tools such as clean buffer count

Using 64-bit Memory

Some circumstances need to take full advantage of 64 bit processing. Here is an example.

```
Input file time interval from: 3-OCT-2014 12:17:39.27
                           to:  3-OCT-2014 12:17:39.27
Total records read:             115771
Total TSNs read:                6
```



```
Total records processed:       0
Total output transactions:     0
Ave records/transaction:       0.00
```

These numbers show that there is a very large transaction that the Loader is processing 115,771 rows so far. The Loader accumulates updates in virtual memory and having more memory can improve performance.

Instead of waiting to discover an unexpectedly large transaction in an application mix, the Administrator can set the Loader to use 64-bit memory from the beginning.

To use 64-bit memory for the Loader data buffers, define the logical name

```
$ define JCC_COMC_VA_MEMORY_MODEL 1
```

Sorting and Performance

Sorting is an important topic when tuning performance. ALso see the documentation for OpenVMS.

Sort Avoidance Optimization

The Loader will avoid sorting buffered records that are already in sorted order. This optimization is automatically in effect unless the sort is disabled.

Currently, the Loader only enables the sort bypass when all of the rows in the buffered rows are for the same table. Therefore, for applications that modify a large number of rows in a single table in a single transaction the performance will show the most benefit.

If NOSORT output logging is enabled, the log will show the benefits.

```
1-OCT-2003 12:19:31.44 208116A9 LML EXMP logging~output~NoSort
 o
 o
 o
1-OCT-2003 14:01:29.27 208116A9 LML EXMP Sort bypassed (200 rows) ...
```

Sort~Disable and Delete Rows

To avoid surprises when sorting is disabled, the Loader always emits the delete rows before the modify rows within a transaction.

This corrects a scenario in which the Administrator would disable the sort and attempt to replicate changes made with an application that does modifies as a combination of delete and insert. Since the Loader is only presented with M or D for the actions, the Loader must interpret the M through an “upsert” (an attempt to modify, followed by an insert if the row does not exist). For the applications described, this could result in the insert being quickly deleted. Since that is not the intended result, the change to always present deletes first avoids an obscure and unintended effect.

Sort and the Map Keywords

See “MapTable and Sort Order” on page 264.

Sortwork File Control

The Rdb LogMiner relies on OpenVMS sort. OpenVMS sort sorts some number of rows and writes them to a work file. It continues in this fashion until there is only one empty file and, then, instead of filling that, the sort does a merge of what is in memory with all the individual sorted runs. If there are a large number of rows to sort, a larger number of sortwork files results in handling rows fewer times.

Beginning with Version 2.0, it is possible to use fewer or more sortwork files. The logical name, JCC_ADD_CLM_SORTWORK_FILES, allows the Administrator to set the number of sortwork files used by the Rdb Continuous LogMiner subprocess of the JCC Continuous LogMiner Loader. The default is six.[1] Integer values between two and nine, inclusive, are recognized. If the logical name is not set to one of these values, the default will be used.

Sort Work Files and Row Sizes

Rdb sortwork files use fixed-length records. Because it is impossible to predict what record types will be sorted within any particular transaction, sort has to be able to handle the largest possible record of all the tables being extracted. So if most of the records are 100 bytes but even one table has a record up to 5000 bytes, sort has to use space for 5000 byte records always. For this reason, surprisingly large sort work spaces can be required.

See also “Space in the CLM Logging Mailbox” on page 467.

1. Originally, six was the only value used.

Tuning the Target

The Loader will eventually slow down if the target cannot accept the input fast enough to keep up.

Target Specific Tuning

Some tuning options will be dependent on understanding the target. See the vendor's documentation and the Loader documentation specific to the target that you have chosen.

The most frequent performance enhancement that is dependent on the target is getting the indices correct. There should be an index on the primary key of any table that is to include updates and deletes. Preferably, that should be the only index on the table.

Materialized Data

There are several columns of materialized data that can be used in ways that may influence performance in the target.

- To partition the target(s)
- To filter out some rows

See "Keyword: VirtualColumn" on page 291.

SQL Interface and Rdb Targets

Beginning with Version 2.0 of the Loader, the SQL interface for an Rdb database was significantly revised. The new interface is much faster. If the old interface is required, use the logical name JCC_LogMiner_Loader_Rdb_version and set the version to less than version 7. For example

```
$ define JCC_LogMiner_Loader_Rdb_version "60"
```


Synchronization and the VMS Lock Manager

The JCC LogMiner Loader uses the OpenVMS Lock Manager, when appropriate, to synchronize updates to the target. The Loader does not use the OpenVMS Lock Manager for single-threaded Loader families or for unconstrained modes. The Loader does use the lock manager with multi-threaded, *constrained*[1] mode Loader families.

The Loader use of the OpenVMS lock manager includes tuning improvements that are automatic.

- The Loader chooses a default locking mode. “Locking and Locking Control Modes” on page 402
- The Loader limits use of ASTs. “Reducing ASTs” on page 401
- The Loader avoids simplifies the locking when a single thread is in use in a multi-threaded loader. “Optimization for a Single Thread” on page 401
- The Loader shifts the locking model when a very large transaction is encountered. “Synchronization for Extremely Large Transactions” on page 403

Most use of the VMS lock manager is tuned by the Loader itself. The Loader Administrator has two control options.

- threshold for defining how large a transaction uses the locking model for very large transactions (“Locking Threshold for Large Transactions” on page 404)
- choice of the locking mode (“Locking Modes” on page 402)

Basic Synchronization for Constrained Parallel Mode

One of the consistency modes of operation for the Loader running in parallel is “constrained mode.” Constrained consistency mode serializes updates to individual rows so that more recent changes to the source are not overwritten by earlier changes.

In constrained mode, the Loader obtains an OpenVMS lock to represent each row read from the Continuous LogMiner. To do this, each Loader takes out a Loader-specific lock on each dbkey, as it reads the record from the Continuous LogMiner. The Loader maintains a list of dbkey locks that it owns, in order to ensure that it

1. See “Consistency Modes Used with Parallel Threads” on page 387.

only attempts to get a lock on a dbkey once. The Loader uses a sequential search algorithm to search the list of currently queued dbkeys prior to attempting to queue a lock request.

Since the Continuous LogMiner will send only the *last* version of a dbkey in any single transaction, a duplicate dbkey cannot occur within a single transaction. Because of this, the Loader is able to search the list of dbkeys *only* up to the beginning of the current transaction.

The Loader stalls while writing to the target until the lock is granted. By this mechanism, updates to individual rows are serialized without blocking rows that are not updated in consecutive transactions.[1]

The algorithms work well for small and medium transactions.

In Loader configurations with a checkpoint interval of one, the search is eliminated completely.

Optimization for a Single Thread

When run in parallel mode, the Loader takes out an appropriate lock (based on the configured locking level) for *each* row read that needs to be written to the target. If a single thread is running, this would cause unnecessary processing. Therefore, when a single thread is running, the thread obtains the target write lock. The effect is as if the lock threshold[2] is set to 1 and protects the ordering should a second thread start before the existing thread has completed its update to the target.

Reducing ASTs

The Loader requests locks[3] with the noqueue option and only requests queuing with AST notification, if the lock isn't immediately granted. This reduces the total num-

1. Additional performance boosts are discussed in "Reducing ASTs" on page 401 and "In Loader configurations with a checkpoint interval of one, the search is eliminated completely." on page 401.
2. See "Locking Threshold for Large Transactions" on page 404.
3. The OpenVMS Lock Manager is used to coordinate constrained consistency mode for multi-threaded uses of the Loader.

ber of Asynchronous System Traps (ASTs) required while locking for constrained consistency mode.[1]

Locking and Locking Control Modes

The JCC LogMiner Loader uses the VMS lock manager to synchronize updates to the target data store. (This use of the lock manager occurs prior to any locking that the target data store may or may not do.) A lock is created for each row to be written to the target and the value of the originating dbkey is used for the resource name.

By default, the Loader uses row level locking. Not all targets implement the same locking model. Additionally, some work loads may cause locking between threads on objects other than rows (pages, indices, etc.) Because locking in the target can reduce the throughput of the Loader, it is necessary to examine finer control.

The logical name JCC_LML_LOCKING_LEVEL enables user control of Loader locking levels. Proper use can improve thread synchronization and performance by reducing locking within the target.

Locking Modes

The logical name may be defined to any of these locking modes:[2]

PAGE. The Loader threads will use only the logical area and page number portions of the originating dbkey for synchronization. This is also known as PAGE(0). The lock name will be displayed as '<logical area>:<page>:-1' in jcc_lml_show_locks. For example, 79:791:-1

PAGE(1). The Loader threads will use only the logical area and page number portions of the originating dbkey for synchronization, *with the extension that the lock will be on the highest page number that is evenly divisible by sixteen*. (That is, the

1. This approach was introduced in Version 2.2.4 of the Loader.The change reduced locking costs by as much as 10%.
2. The Locking Diagnostic Tool, "Example of the Activation Log" on page 369, will reflect the locking mode through the lock names shown in this list.

low order nibble of the page number is masked.) The lock name will be displayed as '<logical area>:<calculated page>/-1:-1' in jcc_lml_show_locks. For example, 80:16/-1:-1[1]

PAGE(2). The Loader threads will use only the logical area and page number portions of the originating dbkey for synchronization, *with the extension that the lock will be on the highest page number that is evenly divisible by 256*. (That is, the low order *byte* of the page number is masked.) The lock name will be displayed as '<logical area>:<calculated page>/-2:-1' in jcc_lml_show_locks. For example, 79:768/-2:-1[2]

LAREA. The Loader threads will use *only the logical area number portion* of the originating dbkey for synchronization. The lock name will be displayed as '<logical area>:-1:-1' in jcc_lml_show_locks. For example, 81:-1:-1[3]

Interpretation of Lock Conflicts

Early versions of the Loader interpreted any Rdb lock conflict as an error and aborted. Beginning with Version 3.1, the Loader interprets an Rdb lock conflict as a form of Rdb deadlock and retries the transaction.

Synchronization for Extremely Large Transactions

To enhance performance for extremely large transactions, the Loader employs an alternate synchronization technique which focuses on getting the extremely large transaction out of the way so that normal processing can resume. The performance benefit is significant.

The Loader Administrator's only involvement is, optionally, to set the threshold for when this happens. The default will be appropriate to most circumstances.

---

1. The notation with the '/-1' is consistent with Rdb notation which is explained this way: Locking level of PAGE(1) masks-out the low order nibble of the page longword so that we get the same page value for any of the pages in the 16 page range. In this case, page 16 is the surrogate lock page for pages 16 - 31.
2. Similarly, to the notation for PAGE(1), this notation indicates the masking of the low order *byte* and page 768 is the surrogate lock page for pages 768 - 1023.
3. Again, this is consistent with Rdb representations.

differs from the standard locking technique in the way the OpenVMS Lock Manager is used. The alternate technique reduces the number of locks obtained for very large transactions.

There are some concomitant changes in the way stall states are reflected in the statistics displays.[1]

The goal of the change is to reduce the cost of using locks by detecting when a Loader thread is using large numbers of locks and switching to an alternate synchronization model. The benefit will be particularly noticeable in systems where the lock manager is poorly tuned.

Locking Threshold for Large Transactions

The logical name JCC_LOGMINER_LOADER_LOCK_THRESHOLD enables the Administrator to define a threshold number of row representation locks that any given Loader thread should obtain. If undefined, the default value for the lock threshold is 100,000.[2] Valid values are 1 to 1,000,000. If an invalid value is specified, the default will be used. There is no way to disable the feature, but the maximum value of 1,000,000 will effectively disable it.

The lock threshold defines the maximum number of row representation locks that the Loader will attempt to obtain before shifting to an alternate approach. The alternate approach is to abandon acquiring row locks and, instead, acquire a single surrogate lock.

The lock used is called the WriteLock. All parallel, constrained mode, Loader threads must obtain the WriteLock to be able to write to the target. If the threshold lock number is not reached by any of the Loader threads, all threads writing to the target will maintain the WriteLock in concurrent write.

When a Loader thread, reading from the Continuous LogMiner mailbox, reaches the lock threshold value, it abandons acquiring locks for each row buffered and, instead, asynchronously requests the WriteLock in protected write mode. When the thread receives the commit record for the transaction, it will release the mailbox

1. See “The Monitor and Loader Threads” on page 320 and “Stall States and Statistics Display” on page 406.
2. This was introduced with version 2.2.4 and represents a change in Loader behavior if the special circumstances are met.

lock and allow other threads to read from the Continuous LogMiner mailbox. All Loader threads that read data subsequent to that will stall behind the thread that met or exceeded the lock threshold. When the process that exceeded the lock threshold releases the WriteLock, processing will proceed in the default way unless and until another thread reaches the threshold.

Example of Locking and Extremely Large Transactions

The alternate synchronization technique can significantly enhance throughput for extremely large transactions. What do we mean by extremely large transactions? In one example, the Loader was worrisomely long handling a transaction of 500,000 rows. With the lock threshold set at the default of 100,000, the improvement was significant.

In the example, instead of six hours and five minutes (required prior to implementation of the alternate technique), the code followed *approximately* this path:

Time	State
1:32 AM	Transaction commits on the source
1:36 AM	CLM starts passing data to LML
1:46 AM	Lock threshold reached, 100,000 rows buffered by LML
1:48 AM	Remaining 400,000 rows buffered by LML, sort and start writing data
2:02 AM	All data written to Oracle target and checkpointed

Note: *This is an example. The times reported are approximate, but illustrative. With different databases, different applications, different hardware, different system tuning, and/or under different circumstances the results may be quite different.*

So, how does more stringent locking lead to net greater throughput? By focusing on getting the extremely large transaction(s) out of the way, significant expenditure of resources spent with locks and ASTs is eliminated.

Supporting Stall States

The synchronization technique for extremely large transactions[1] is reflected in the monitor with two stall states.[2] These are described here.

1. "Synchronization for Extremely Large Transactions" on page 403.

WriteLock Stall state ("|"). The thread that reached the threshold, will finish reading from the mailbox and do the required sorting. If this activity is completed before the WriteLock is granted, that thread will be in the WriteLock Stall state. It will remain in this state until all Loader threads holding the lock in concurrent write have released it.

WriteLock Block state ("]"). If, while a thread is waiting for the WriteLock in protected write mode, other threads that have not reached the lock threshold, reach the write phase, they will stall waiting for the WriteLock in concurrent write mode.

Stall States and Statistics Display

The stall states related to synchronization of extremely large transactions are reflected in the statistics display as shown in an example.[1]

Writers. In the example, three threads are in the ">" state. This means that they have the WriteLock in concurrent write mode, as well as all the dbkey locks that they need and they are currently writing to the target.

Waiters. In the example, there are twenty threads in the "W" state. This means that they have the WriteLock in concurrent write mode, but do not have all of the dbkey locks that they require. They are waiting on those locks.

Writelock Stalled. In the example, there is one thread, thread u, in the "|" state. This means that this thread has reached the lock threshold and is waiting for the writers and the waiters to release the WriteLock so that the thread can get the WriteLock in protected write mode. Note that there could be more than one thread in this state.

Writelock Blocked. In the example, there are seven threads in the "]" state. This means that these processes have requested the WriteLock in concurrent mode and been blocked by the request of the "writelock stalled" process or by processes with a prior request to get the WriteLock in protected write mode. These processes will continue to be blocked until all the processes with prior requests for the WriteLock in protected write mode have gotten and released the lock and those processes cannot get the lock until the "writers" and the "waiters" have released it. When these

---

2. See "The Monitor and Loader Threads" on page 320.

1. Note that you will not see the Writelock Stalled and Writelock Blocked stall states except in architectures involving extremely large transactions.

blocked processes do get the WriteLock in concurrent write mode, some of them will become active writers immediately and some will be stalled further while waiting for dbkey locks.

```
Rate:   6.00                        REGTESTRDB            22-FEB-2005 15:11:03.00
================================================================================
   Input: 22-FEB-2005 15:00:25.93               Output: 22-FEB-2005 15:00:01.39
--------------------------------               --------------------------------
Transactions                 3849              Checkpoints                   520
Records                     90508                 Timeout                      0
   Modify                   86041                 BufferLimit(   262)          89
   Delete                     618                 NoWork                       0
   Commit                    3849              Records(   3)                82583
 Discarded                                         Messages(  N/A )            N/A
   Filtered                    0                   Filtered                     0
   Excluded                    0              Failure                           0
   Unknown                     0              Timeout                           0
   Restart                     0         - Current ---------------- Ave/Second -
   NoWork                    475         Checkpoints                2         0.34
   Heartbeat                   0         Records                  584        98.67
Timeout                        0         Rate                  115.30%
--- Restart Context -----                 Latency(sec)           94.65
```

FIGURE 1. Detail Statistics - Showing Stall States

Performance Improvements in the Loader

Every release of the JCC LogMiner Loader has included one or more ways of tuning Loader performance without any intervention. For example, with version 3.6, the attaches to the database used to process FilterMap and MapResult are reduced, logging performance is improved through how the process priority is set, and batch updates with the JDBC interface are improved.

Therefore, one way to tune performance is to ensure that the latest release is the one being used.

Analyzing Performance

Analysis of performance for any complex system requires finding ways to see what is happening and following the leads provided.

If the performance of an application seems slow, begin your analysis by checking the ping time between the source and the target. If you are not already runnning the Loader statistics monitor, do so. See “Monitoring an Ongoing Loader Operation” on page 313.

Particularly if writes to the target have gotten slower, as it has acquired more data, review whether you have indexes on the primary keys.

This chapter includes approaches to tuning the Loader and the overall architecture. If the Loader monitor shows difficulty with the target, use some of the same approaches that might be used with tuning the source. If the target is as noticeably the problem as in the following image from a running system, no amount of tuning the Loader is going to address the problem.

Summary

Achieving good performance is a balancing act. No answer is correct for all situations. Some experimentation will be required.

CHAPTER 16

Aids for the Administrator

When this chapter was first developed, it was called Aids for the Database Administrator. Then, the support group for the JCC LogMiner Loader realized that it wasn't always the DBA who contacted us with the interesting questions. In addition, the questions are not limited to the database.

In this chapter, you will find a collection of materials that are particularly important to the person charged with "making it all work."

You need tools to monitor and tune performance and to answer questions and control what is happening. The JCC LogMiner Loader includes tools that meet these needs.

You will find relevant material in many chapters, including "Control File" on page 217, "Monitoring an Ongoing Loader Operation" on page 313, "Performance Considerations" on page 381, and others.

Note that the chapters on specific targets also include important information for your success.

Additional material is available in the blogs

http://www.jcc.com/lml-blog

Topics

This chapter includes sections intended to aid the Administrator charged with maintaining and operating the architecture that includes the JCC LogMiner Loader. Note that the architecture also includes software that is not a product of JCC Consulting and that the documentation for those products may also be needed.

The topics included here are, of necessity, are only loosely related. Some sections are very long and some quite short. Many of the sections are clarifications suggested by support desk traffic.

The list to follow is loosely grouped by overall topic and may help you find the portions that you need.

- Rdb Issues and Loader Options for Addressing those Issues
  - "Dangerous Interaction Between RMU Backup and LogMiner" on page 412
  - "Loader Heartbeat and AIJ Backup" on page 475
  - "Repairing Invalid AERCP Values" on page 434
  - "Oracle SR 3-12002172341" on page 413
- Oracle and Oracle targets "Oracle Issues" on page 415
- After Image Journal Tools and concepts
  - "The After Image Journal - AIJ" on page 416
  - "Knowing Whether the AIJ Is Processed" on page 417
  - "Searchlist for AIJ Backup Files" on page 417
  - "Understanding the AIJ Switches and Micro Quiet Points" on page 418
  - "AIJ Backup Stalls" on page 418
  - "Dangerous Behavior" on page 418
- Starting, Shutting Down, and Restoring the Loader
  - Normal restart "Restart, Recovery, and Shutdown" on page 420
  - Normal shutdown "Shutdown" on page 425

- The full run command“Overrides in the Run Command” on page 427
- Starting the Loader for the first time “Setting the Restart Context for the First Time” on page 433 and “Starting for the First Time in the Backed Up AIJs” on page 430.

- Unusual Restart Conditions
  - “Overrides in the Run Command” on page 427.
  - “Starting for the First Time in the Backed Up AIJs” on page 430.
  - “Restarting in the Live AIJ after DB Reorganization” on page 430.
- “Special Restart - Skipping Updates on Purpose” on page 435.
- Upgrading and changing
  - Version Changes for Rdb “Upgrades and Changes” on page 449.
  - Version Changes for the Loader “Upgrading the Loader” on page 450.
  - Metadata Changes for the Source Database “Metadata Changes and Mapping the Source to the Target” on page 452.
- OpenVMS and related topics
  - “OpenVMS and the Loader” on page 454.
  - “Finding Sessions in the Cluster” on page 455.
  - “Directory Security” on page 455.
- “Operator Classes and Tardiness Messages” on page 457.
- Naming and Placing Things
  - “Naming and Placing the Log Files” on page 460.
  - “Controls for the Filter Database” on page 461.
  - “Logical Name Controls for Loader Procedures” on page 463.
- Enhancing Control
  - “Generating the Control File” on page 463.
  - “Logical Name Controls for Loader Procedures” on page 463.
  - “Tuning Considerations” on page 465.
  - Review the sections in “Extended Examples and Tools” on page 547.
- Exception Handling, Analysis, and Tuning
  - Previous chapters, including “Monitoring an Ongoing Loader Operation” on page 313 and “Performance Considerations” on page 381.
  - “Exception Messages” on page 464.

Rdb Issues

From time to time, issues will be discovered with any sophisticated software product. Oracle Rdb issues of concern to the Database Administrator when running the JCC LogMiner Loader, that have not been resolved by the publication of this documentation, are reflected in the following.

Dangerous Interaction Between RMU Backup and LogMiner

The difficulty reported here is recorded as Oracle Rdb bug #18601419.

The Oracle Rdb documentation indicates that RMU LogMiner and RMU AIJ backup can interact in a rare and unpleasant way. The result is the potential loss of multiple transactions to the LogMiner. Although this problem is rare, it has been encountered in at least two JCC customer installations.

The problem can occur if the LogMiner starts processing in backup AIJ files and then switches to the Live files while an AIJ backup is progressing. Since the LogMiner is behind and part way through the journal, it is possible that the backup process can finish and erase the AIJ. Since the AIJ architecture allows for empty pages, when the LogMiner finds empty space, it skips to the end without an exception.

All of this is done silently with no hint of an anomalous situation. Neither JCC's Loader software nor the customer has a clue that the process can loose transactions in the stream going from the database to the Loader target.

The situation is compounded by the fact that the Loader "remembers" where it is in the journal. This memory is contained in the checkpoint record for the Loader family session. Once the checkpoint is advanced, it is advanced.

The only solution is to "rewind" the Loader session. It can be forced to start at the beginning of a backed up AIJ file set. (Recall that the earliest of these backed up AIJs must be at a quiet point boundary.) Alternately, the JCC LML Checkpoint Cache Server can be used to identify a point in time much closer to the actual missing data.

Until Oracle Rdb Engineering addresses this issue, customers are advised to avoid running an AIJ backup while LogMiner is processing AIJ backup files. If the LogMiner is currently processing the live AIJs, RMU Backup/After and the RMU LogMiner work correctly together.

See also "Knowing Whether the AIJ Is Processed" on page 417, "Automated AIJ Backups" on page 483, and the blog www.jcc.com/lml-abs-bad. The blog will be updated as appropriate and may have more recent information on the Rdb bug.

Oracle SR 3-12002172341

Bug reference number 22561645 impacts use of the Loader Data Pump with Rdb Versions 7.3.1.0 to 7.3.1.3. The issue is addressed in Rdb Version 7.3.2.

Rdb Version	Rdb Characteristic	Loader Result
prior to 7.3.1.0	LARGE MEMORY IS ENABLED is invalid.	Loader's Data Pump VLM feature will fail if use is attempted..
7.3.1.0 to 7.3.1.3	New syntax works, but the feature doesn't work.	VLM feature will not fail immediately, but VLM will not actually be used.
7.3.2	Syntax and feature work.	The Loader's Data Pump can use VLM and handle larger data sets.

The problem is that the "LARGE MEMORY IS ENABLED" syntax is either invalid or ignored, for Rdb versions prior to 7.3.2, for the type of temporary table

that the Data Pump uses, such that inserting more than 22,036,168 rows into the temporary table will fail.

MQP Repair

See also “Repairing Invalid AERCP Values” on page 434.

Asynchronous Writes to the VMS Mailbox

Changes to RMU/UNLOAD/AFTER have occurred in Versions 7.3.1.3 and 7.3.2.1 of Rdb. The following chart summarizes the changes and the impact on Loader operations.

Rdb Version	Behavior	Loader Results
Prior to 7.3.1.3	No option for asynchronous writes to the VMS mailbox	Loader running well
7.3.1.3 until 7.3.2.1	Rdb introduces asynchronous writes to the VMS mailbox to improve performance	Loader users discover that the async writes *can* saturate the mailbox causing the entire process to hang causing AIJ Backup Server (ABS) to stall and block other database users.
7.3.2.1	Rdb introduces a new qualifier for RMU/UNLOAD/AFTER that is /[NO]MBX_ASYNCH	Loader introduces the logical name JCC_ADD_CLM_MBX_ASYNCH to indicate when the new Rdb qualifier should be used.

More information is available from the Rdb release notes or from Oracle Bug 24514893.

Note that the logical name should not be defined when using an Rdb version prior to 7.3.2.1, as the /[NO]MBX_ASYNCH qualifier is not defined for these versions and the LogMiner will exit with the exception

```
%DCL-W-IVQUAL, unrecognized qualifier - check validity, spelling, and placement
```

If the logical name is not defined, the MBX_ASYNCH qualifier is not specified. That is, asynchronous writes are *not* used and the issues introduced with asynchronous writes are not present, as was true prior to 7.3.1.3.

To use asynchronous writes define the logical name to true (with '1', 'T', or 't' as the first character). This causes the Loader to use the asynchronous qualifier RMU/UNLOAD/AFTER/MBX_ASYNCH which causes Rdb to use asynchronous writes to the VMS mailbox.

Defining the logical name to false (with '0' or anything other than '1', 'T', or 't') causes the Loader to use the qualifier that turns off asynchrous writes, RMU/UNLOAD.AFTER/NOMBX_ASYNCH. In this case, Rdb will use only synchronous writes to the VMS mailbox, as was true prior to Version 7.3.1.3.

Versions of Rdb prior to 7.3.1.3 will fail if the qualifier is supplied. Version 3.5 of the Loader enables requesting asynchronous behavior for Rdb versions which support it. Care is recommended.

Oracle Issues

Those users of the JCC LogMiner Loader who are using Oracle targets need to be aware of the following issues:

Behavior Change in New Version of Oracle

While regression testing for new Oracle versions,[1] JCC noticed that for the Oracle 11.2 interface for OpenVMS, when specifying the target, it is necessary to include the full TNS name specification in the Control File and in the dump checkpoint procedures. For example, "regtest.jcc.com" is required where "regtest" was adequate for former versions of Oracle on OpenVMS[2].

1. For the most up-to-date chart of which versions are supported, please reference the information given on the JCC web pages in the Resources area in the JCC LogMiner Loader Hints blogs, specifically the blog on Product Compatibility. http://www.jcc.com/lml_prod_compat
2. To support Oracle as a target via the OCI interface requires that Oracle SQL*net and an appropriate client version of Oracle be running on OpenVMS. Check with Oracle for which target database versions require which client versions on OpenVMS. As of this writing, the latest client version for OpenVMS is Oracle 11.2. The testing mentioned was using Oracle 12.1 on the target.

This may require attention on upgrades, as the Oracle 10.2 interface on OpenVMS did not require the full TNS name specification.

JDBC Batching and Oracle End Targets

If you are using the JDBC target to write to an Oracle database of a recent version, you need to be aware of a change in behavior.

Previously, Oracle JDBC driver support for standard batches was insufficient and the Loader, consequently disabled batch support if the driver was Oracle. Oracle's 12c JDBC documentation includes the following:

> Deprecated. As of 12.1 all APIs related to oracle-style statement batching are deprecated in favor of standard JDBC batching. We recommend using the standard model going forward as it is spec compliant and provides more information and control to the application.

None of that invalidates earlier versions of the Loader, as the Loader merely disables batching when using the Oracle JDBC driver. However, the Loader can now take advantage of the improvement in Oracle handling of JDBC batches.

For Version 3.5 of the JCC LogMiner Loader, standard JDBC batching will be available to Oracle targets of Version 12.1.0.1.0 or later. Having this batching available is expected to offer performance options.

The After Image Journal - AIJ

Managing the AIJs is a critical component of a smoothly running JCC LogMiner Loader architecture. See "AIJ Files" on page 28 for a general introduction. Because the AIJ is critical to successful use of the JCC LogMiner Loader, the kit includes some tools.

Knowing Whether the AIJ Is Processed

In certain circumstances, you will need to know whether all of the processing of the existing AIJs has occurred. These circumstances include

- Before you do a database reorganization
- Before you modify the metadata

To know that the LogMiner and Loader have completed work with the AIJ,

1. Build the safety test described in "Safety Test for AIJ Backup" on page 418 into the backup procedure.
2. Check the LogMiner with RMU/SHOW STATISTIC <source database>. When the statistics screen appears, press the letter "M" to bring up the menu. Choose "Journaling Information", then "LogMiner Information". On this screen, find
   - At the top, "CurrSeqNum" (the current AIJ sequence number that the database is writing) and "CurrEOF" (the current AIJ end of file). These two numbers will not change unless an update transaction is committed.
   - Data under the headings "Process.ID State ..... SeqNum CurrVBN". The LogMiner has finished processing all of the data in the AIJ files when it shows a state of "Hibernating".
3. Check the Loader with "$ jcc_lml_statistics <LoaderName> 6 d". In the lower lefthand corner, verify that all running Loader threads are in either the "z" or "R" states and are not changing.

Searchlist for AIJ Backup Files

Having AIJ backup files in multiple directories on multiple devices is not handled smoothly by the Continuous LogMiner. To address this, the JCC LogMiner Loader uses a logical name, JCC_AIJ_BACKUP_SPEC, to specify the location of the backup files. Processing of this logical name will dissect multiple definitions of AIJ backup file specifications and supply them to the Continuous LogMiner as a comma separated list of values, which the LogMiner can handle correctly.

Examples. The first example shows two specifications in a list.

```
$ define jcc_aij_backup_spec <disk1>:[<dir1>]*.aij_backup, <disk2>:[<dir2>]*.aij_backup
```

The second example is more generic and handles situations that need to include backed up AIJs with varying filenames.

```
$ define jcc_aij_backup_spec AIJ_DIR:*.AIJ;*
```


Understanding the AIJ Switches and Micro Quiet Points

See “Tracking AIJ Switches” on page 432

AIJ Backup Stalls

See “Loader Heartbeat and AIJ Backup” on page 475.

Dangerous Behavior

See “Dangerous Interaction Between RMU Backup and LogMiner” on page 412 and “Automated AIJ Backups” on page 483.

Safety Test for AIJ Backup

Rdb documentation indicates that RMU AIJ backup and RMU LogMiner (RMU/Unload/After_Journal) can interact in a rare and unpleasant way. The result is the potential loss of multiple transactions to the LogMiner. This problem has been experienced by at least two JCC Customers.

For more on this difficulty, read the blog entry

http://www.jcc.com/lml-abs-bad

To avoid the rare, but significant difficulties that can result from this interaction, include JCC_CLML_AIJ_BACKUP_SAFE in the procedures that run backup and use it to ensure that backup does not run while the LogMiner is still catching up. JCC_CLML_AIJ_BACKUP_SAFE is a new routine in the Loader kit. The routine tests whether it is safe to run backup by ensuring that the LogMiner is processing data in the LIVE AIJ files.

```
$ JCC_CLML_AIJ_BACKUP_SAFE      -
      <database name>           -
      <Loader name>             -
      <Checkpoint name>         -
      [<checkpoint type>]
```

Parameters

<Database Name>. Database name is the source database.

<LoaderName>. LoaderName is the LoaderName for the checkpoint.

<checkpoint name>. Checkpoint name is the checkpoint filename or the target (of the checkpoint) database or OCI service. (If an OCI service is specified, the routine will prompt for proper credentials to access the target database.)

<checkpoint type> ***optional.*** Checkpoint type is the optional type of the checkpoint stream: LML_INTERNAL, OCI, or RDB. (The default is LML_INTERNAL.)

Output

The procedure will set the DCL symbol JCCLML$AIJ_BACKUP_SAFE to either “True” or “False”.

It is important not to proceed if

```
JCCLML$AIJ_BACKUP_SAFE == "False"
```

Use

A good practice would be to call this procedure as part of the AIJ backup procedure and to continue only if JCCLML$AIJ_BACKUP_SAFE is returned as “True”. If it is returned as “False”, backup should wait a few minutes and try again. The procedure should include a counter and alert someone if it has tried some set number of times without receiving “True”.

Restart, Recovery, and Shutdown

JCC's LogMiner Loader, working with the Rdb LogMiner, provides robust support for restarts. The Loader maintains its own context and, by default, will restart where stopped, whether it was stopped by a failure or by a person.

This section includes explanation of how the Loader is able to restart normally when running as the Continuous LogMiner Loader and includes special notes on restart for each of the other modes. The section also includes notes on shutdown, on starting the Loader for the first time, and on the seldom used override parameters for the full run command.

High-Water Table and Checkpoint File

To protect the ability to restart, the Loader requires a special reference for the restart information. With Oracle or Rdb database targets, this is, generally, a table in the target database. The Loader uses this *high water* table to maintain the context each time the Loader session commits. Various information is kept in this table including the AERCP and TSN of the last successful commit. [1]

The *checkpoint file* plays a similar role for those sessions for which the database table is not an option. Both the table and the file are referred to in this discussion as checkpoint and writing to them as checkpointing. [2]

Each separate Loader thread[3] that updates the target database has a named record in the high-water table (or checkpoint file). This named record is read during restart and the LogMiner output is properly positioned to begin the loading session at the point of failure.

The Loader Restart Context

The Loader restart context has several elements.

1. See "The Loader Restart Context" on page 420.
2. Checkpointing to the highwater table or checkpoint file should be distinguished from the keyword checkpoint which establishes the commit interval.
3. See "Parallelism and Loader Threads" on page 383, "The Monitor and Loader Threads" on page 320, and "Keyword: Thread" on page 284.

- The Rdb AERCP tells the LogMiner the AIJ sequence number and relative block offset of the last quiet point (or micro-quiet point). [1] This is the point where the LogMiner can restart.
- The TSN of the last transaction committed to the target tells the Loader where to begin again. The Loader will ignore transactions from LogMiner until this TSN appears.
- The Loader Sequence Number is used by some applications. It is a value that is incremented by one each time the Loader sees a commit record from LogMiner (provided there are rows of interest within the transaction)[2]. It may be materialized as part of the output to the target, using the virtual column keyword. (See "Keyword: VirtualColumn" on page 291.) It may be used by the application to indicate the relative age of each update and is particularly important in applications that may not maintain strict transactional consistency. In these cases, the Loader sequence number is sometimes used to prevent overwriting data with data from an older transaction.

Completeness

The Loader is designed to ensure that all records received from the LogMiner are passed on to the target at least once. Success is not assumed until the Loader has recorded the event.

In the case of database targets with a highwater table, the Loader's transaction commits the highwater row as part of the commit of the other rows. The information on which is the last transaction processed is, therefore, completely accurate ... for single threaded Loaders.

However, in the case of multiple threads, the Loader must begin with the transaction immediately past the earliest thread checkpoint. Since some threads may have gotten further than others, some transactions may be re-sent. Each transaction will be sent at least once.

1. A quiet point is a moment in time, an epoch, when all transactions that have been started have also been committed or rolled back. A micro-quiet point is a quiet point that occurs naturally, without a request for a quiet point backup. A quiet point can also be achieved by requesting a quiet point backup. In this case, the quiet point will be at the beginning of the AIJ just *after* the quiet point backup. A quiet point is a necessary starting point for LogMiner. See also "Quiet Points" on page 97.
2. See "No Work Transactions and Checkpoint Intervals" on page 389.

With a checkpoint file and a non-database target, the Loader may pass on the source rows and not receive an acknowledgment before there is an interruption. In this case, the target process may have recorded the rows, but the Loader has not yet updated the checkpoint. On re-start the Loader will send the questionable rows again. The target interfaces are designed to minimize the number of rows that may be sent twice. Each transaction will be sent at least once. No transactions are lost.

Restart for Continuous LogMiner and Loading

The Continuous LogMiner[1] and Loader process has been designed to be interruptible and restartable with no data loss. The nature of the restart will depend on what happened to the AIJ while LML was down. If active AIJs have not been backed up, while the Loader was stopped, it will be possible to just restart everything in place. If processing has continued and has been sufficient that an AIJ backup has occurred, the Continuous LogMiner will start with the backed up AIJ and will proceed operating with the backed up AIJ files, in the appropriate order, before switching to the active AIJ.

To support starting in the backed up AIJ files requires that the logical name JCC_AIJ_backup_spec be defined. The LogMiner will use this logical name to identify where to find the backed up AIJ files.

The logical name can use wildcards and, in some cases, must. The definition of the logical name must include all versions of the AIJ files. In rare situations that include multiple versions of the filename or filenames that vary based on which AIJ file was being backed up, wildcards will be needed to collect all of the backup files needed. For example:

```
$define JCC_AIJ_BACKUP_SPEC AIJ_DIR:*.AIJ;*
```

Restart and Other Modes

The concern for management of the AIJs also applies to static and copy modes.

Static LogMiner. The static LogMiner only processes backed up AIJs. The changes represented in the AIJs, however, must be processed in the same order that they occurred in the source database. With the static LogMiner, therefore, care is required in ordering any backed up AIJs. If AIJs are applied out of sequence, unpre-

1. You may also wish to read “Shutting Down Continuous LogMiner” on page 75.

dictable data results. Older data may overwrite newer data. If mining multiple AIJs at once, cause the LogMiner to order them by journal sequence number.

Static Mode Loader. The Loader Control Process can be instructed to run in Static mode. For this, the process uses the Static LogMiner and the Continuous LogMiner Loader. The Loader's Control Process manages the operation and relies on the JCC_AIJ_backup_spec logical name to ensure that the expected AIJ files are processed. Restart is as it is for Continuous mode. See "Restart for Continuous LogMiner and Loading" on page 422.

Copy Mode. Copy mode uses the Static LogMiner and the Continuous LogMiner Loader. However, unlike Static mode, Copy mode does not run the LogMiner under the control of the Loader's Control Process. Therefore, care must be taken with the AIJ sequencing.

Because Copy mode has some aspects of continuous operation, but has a limited input stream, handling restart requires a bit of understanding. See "Checkpointing and Discontinuities" on page 88 for a discussion of how to repeatedly apply the Copy Procedure output drawn from the static LogMiner.

The Control Process

The control process for running the Continuous LogMiner and Loader starts both the Continuous LogMiner (CLM) and the Continuous LogMiner Loader (CLML).[1] If the CTL (control) process detects a restart condition, [2] then it sends two extra parameters to the CLM subprocess command procedure. Those two extra parameters are the formatted AERCP and the logical name JCC_AIJ_backup_spec.

The formatted AERCP is specified on the CLM command line via the /restart qualifier. The AIJ backup file is specified on the CLM command line in addition to the /order_aij/files qualifier. Note that the logical name JCC_AIJ_backup_spec must be defined before running the Loader.

1. The Control Process also runs the LogMiner and Loader for non-continuous operation. See "Modes of Operation" on page 73.
2. The highwater record has complete_flag = 'N'.

Restart and Backup

The CLML begins with backed up AIJs, if there are any that have not been processed, and switches to the active AIJ only when finished with the backed up AIJs.

Note that the LogMiner and the Loader cannot process AIJ files that are unavailable. If any AIJ files are missing — or are not found, using the logical name JCC_AIJ_backup_spec — the Loader family will shut down.

Backing Up and Catching Up

During the catch-up phase for CLML, once the Loader is pointed at the backed up AIJs, no further AIJ backups should occur prior to the LogMiner switching to the active journal. If a backup does occur while the LogMiner is catching up, you will see an error log something like this.

In particular, note the messages for 09:57:57

.

```
17-MAY-2002 09:57:55.83 21EDA8A3 CLM TFY_DCA_CL   %RMU-I-LOGRECSTAT,
transaction with TSN 0:1010245292 committed
17-MAY-2002 09:57:55.83 21EDA8A3 CLM TFY_DCA_CL   %RMU-I-LOGRECSTAT,
transaction with TSN 0:1010245293 committed
17-MAY-2002 09:57:55.83 21EDA8A3 CLM TFY_DCA_CL   %RMU-I-LOGRECSTAT,
transaction with TSN 0:1010245295 committed
17-MAY-2002 09:57:55.83 21EDA8A3 CLM TFY_DCA_CL   %RMU-I-AIJMODSEQ, next AIJ
file sequence number will be 5449
17-MAY-2002 09:57:57.79 21EDA8A3 CLM TFY_DCA_CL   %RMU-F-AIJSEQAFT,
incorrect AIJ file sequence 5450 when 5449 was expected
17-MAY-2002 09:57:57.79 21EDA8A3 CLM TFY_DCA_CL   %RMU-F-FTL_RMU, Fatal
error for RMU operation at 17-MAY-2002 09:57:57.79
17-MAY-2002 09:57:59.84 21EDA8A3 CLM TFY_DCA_CL   09:57:59 $error_exit:
17-MAY-2002 09:57:59.84 21EDA8A3 CLM TFY_DCA_CL   09:57:59 $!
```

FIGURE 1. Error Log When Backed Up AIJ Cannot Be Found

If the problem is that someone kicked off an AIJ back up while CLML was still catching up, getting the exceptions messages is an awkward interruption. However, recovery is accomplished by restarting again.

Shutdown

It may help to review the shutdown procedure. To shut down the Continuous LogMiner and Loader, use

```
$ JCC_CLML_SHUTDOWN <loadername>
```

A Continuous LogMiner and Loader session consists of at least:

- The Control process
  - Created when you submit the batch procedure
  - Starts and stops the other processes
  - Manages the log files
- LogMiner process
  - Started by the Control process
  - Is specific to a Loader session
  - Extracts the data from the AIJ and feeds it to the Loader process
- Loader process
  - Started by the Control process
  - Receives data from the LogMiner process and applies it to the target
  - May be a single process (the default and most frequent occurrence) or multiple processes, if the Loader is running in parallel mode (See "Parallelism and Loader Threads" on page 383.)

The JCC_CLML_SHUTDOWN procedure gracefully shuts down all these processes. It also drains and closes the mailbox used for communication between the LogMiner and the Loader.

Normal Shutdown Example

An example of the output from a normal shutdown is shown here. Note that it identifies who and what processes required the shutdown. (The wrap between the first and second lines is added to improve the readability of the example.)


```
21-JUL-2005 22:30:09.32: Received 'SHUTDOWN' from
process 2361D623 [ATLAS::JEFF           (2361A203 BATCH_1191]
Shutting down CLM process

21-JUL-2005 22:30:09.34:  Waiting for process to exit.

21-JUL-2005 22:30:09.47:  LML thread 0 was shutdown normally.
%dba_clm_ast:  CLM process was shutdown.
CLM has exited.
LML has exited.
Exiting Continuous LogMiner Loader

%jcc_continuous_logminer_loader:  exit status
   %DBA-S-SUCCESS, Routine completed successfully.
```

FIGURE 2. Normal Shutdown

Automatic Shutdown and Failure

The Control (CTL) process also manages the shutdown when a process fails. It can be important to know what caused the shutdown. Therefore, the message[1] will be either:

```
%DBA-I-CLML_FAIL, CLML failed, please see CTL log for more information.
%DBA-I-LML_FAIL, Loader failed, please see CTL or LML log for more information.
```

FIGURE 3. Automatic Shutdown Following a Failure

Each message identifies the correct process and log file for diagnosing issues.

If the Continuous LogMiner executable exits for some reason other than being shut-down by the Control (CTL) process, the exit status of the Continuous LogMiner (CLM) process will be the exit status of the Continuous LogMiner executable.

1. These two messages were introduced in Version 2.2.8 to replace the more generic %DBA-I-SHUTDOWN, Shutdown requested.

Default Restart

By default the Loader, on restart, will read its highwater data and attempt to restart at the transaction following the last transaction successfully sent to the target. It processes through backup AIJs, if necessary, and continues into the live AIJ.

For the default restart, see “Running Continuous LogMiner and the Loader” on page 72.

Overrides in the Run Command

Under certain circumstances, the JCC LogMiner Loader may be run with overrides. These overrides can cause the Loader to alter its default processing of the Loader checkpoint information.

In the following circumstances, you may need to use the optional override parameters but, even then, do so with caution. In fact, before you use the optional override parameters, you may want to contact Loader support to ensure that the override is the best approach.

1. If the DBA is doing a big database reorganization that is not journaled, the Administrator must ensure that all LogMiner activity is completed before starting the reorganization. The Administrator can, then, shut down the Loader and LogMiner, backup the journal, and restart (in the live journal) using the overrides.
2. If an environment has gotten thoroughly confused, it may be necessary to use the override. Hopefully, this will never happen in production.[1]
3. If you are running the LogMiner and Loader for the first time on the source and need to start in the backed up AIJs.[2]
4. For special applications that do not need “old” data there is an additional option.[3]

1. See “Data Pump” on page 505 for information on how to re-establish a correct starting point and discuss with JCC LogMiner Loader Support how best to use the run command to get started again.
2. See “Starting for the First Time in the Backed Up AIJs” on page 430.
3. See “Thus, the work to create realistic test environments for down stream applications is reduced, as are the surprises from not having tested with realistic data.” on page 483.

In all other circumstances, JCC advises ignoring the override parameters.

The Full Run Command

See "Running Continuous LogMiner and the Loader" on page 72 for the normal start and the normal restart. *The following is for exceptions only.*

```
$ JCC_RUN_CLM_LML                    -
<source database name>               -
<LogMiner options file>              -
<LogMiner Loader Control File>       -
[<restart override tag>              -
<LogMiner restart context>           -
<Loader sequence number>             ]
```
[1]

The last three parameters are optional. *They can be used during restart, but generally are not.* Do NOT use them unless you fully understand the implications and have a situation which warrants this unusual action.

<Source Database Name>. The source database name specifies the name of the source database that the continuous LogMiner is running against.

<LogMiner Options File>. The LogMiner options file is the name of the LogMiner options file that describes the tables to be mined. See "<LogMiner Options File>" on page 73.

<LogMiner Loader Control File>. The LogMiner Loader Control File must specify the name of the target database. It also specifies many additional aspects of how the Loader session should operate. See "<LogMiner Loader Control File>" on page 73 and "Control File" on page 209.

<Restart Override Tag> ***Optional*****.** The restart override tag causes the Loader to alter its default processing of the Loader checkpoint information. This parameter can have values FILE or CHECKPOINT or can be left out. *Note that this parameter overrides the normal, default behavior. In almost all circumstances, you do not want to use this.*

1. Each of the optional parameters are independent, but each of the last two place requirements on how the first is used. See the details on each parameter for further explanation.

- If not used (the default), the Loader, on restart, will read its highwater data and attempt to restart at the transaction following the last transaction successfully sent to the target.
- If FILE, the Loader will use all of the highwater information *except the name of the input file.* Instead of using the input file that it had been using, it will get the input filename from the logical name jcc_logminer_loader_input. Note that this applies *only* to static mode.
- If CHECKPOINT, the Loader will discard all of the highwater information — except the Loader Sequence Number (LSN) — and start as if from scratch.
- If the null string, the default behavior will occur. Note, that this parameter is required if one of the other optional parameters is used. Therefore, using this parameter with the null string maintains the default behavior.

<LogMiner Restart Context> ***Optional.*** This parameter controls whether the Continuous LogMiner starts in the active AIJ or starts with backed up AIJs. Values are LIVE, RESTART, BACKUP, or a timestamp. *Note that this parameter — if the restart override tag is specified — overrides the normal, default behavior. In most circumstances, you do not want to use this.*

- If LIVE, start at the beginning of the live (active) AIJ.
- If BACKUP, start at the beginning of the backed up AIJs.
- If RESTART, start where processing stopped. This is the default.
- If a timestamp, the Special Restart that relies on the Cache Checkpoint Server will be used. (See "Cache Checkpoint Server" on page 437.)

Calling it <LogMiner restart context> may be confusing since there is something in the Rdb LogMiner called the AERCP which is often referred to as the LogMiner restart context. Also, this parameter applies to starting the *first time*, as well as restarting. When starting for the first time in the backed up AIJs, see "Starting for the First Time in the Backed Up AIJs" on page 430.

The <restart override tag> and the <logminer restart context> are independent of each other and each are individually optional. However, if you wish to specify a <logminer restart context> and not a <restart override tag>, then the <restart override tag> must be specified as an empty string.

<Loader Sequence Number> ***Optional.*** This parameter provides a way to override the Loader sequence number. Values are RESTORE or a valid unsigned quadword integer value. *Note that this parameter — if restart override tag is specified —*

overrides the normal, default behavior. In most circumstances, you do not want to use this.

- If RESTORE, start with the saved context. This is the default.
- If an integer, override the saved context and start with the value given for the Loader sequence number.

Note that the Loader Sequence Number is sometimes used to establish the relative age of data in the target. Modifying this value can have significant consequences.

The <restart override tag> and the <loader_sequence_number> are independent of each other and each are individually optional. However, if you wish to specify a <logminer restart context> and not a <restart override tag>, then the <restart override tag> must be specified as an empty string.

Starting for the First Time in the Backed Up AIJs

To start for the *first time in the backed up AIJs,* instead of in the live AIJ, use the run command with the restart override tag as the empty string and the logminer restart context as 'BACKUP'. The full command will be

```
$ JCC_RUN_CLM_LML                  -
<source database name>             -
<LogMiner options file>            -
<LogMiner Loader Control File>     -
""
BACKUP
```

FIGURE 4. Starting the Loader in the Backed Up AIJs

Define the logical name JCC_AIJ_backup_spec. See "Restart for Continuous LogMiner and Loading" on page 422.

Restarting in the Live AIJ after DB Reorganization

If you do a database reorganization, you will want to shutdown Loader operations and journaling during the reorganization.

After an unjournaled database reorganization, use the run command with the restart override tag as the empty string and the logminer restart context as 'BACKUP'. The full command will be

```
$ JCC_RUN_CLM_LML                    -
<source database name>               -
<LogMiner options file>              -
<LogMiner Loader Control File>       -
""
LIVE
```

FIGURE 5. Restarting the Loader in the Active AIJ

To avoid losing changes that occur in the source, but will not be replicated to the target, you must drain all AIJs before the reorganization. That is,

1. Shutdown applications that access the source database, specifically those that write to the source.
2. Allow the Loader to completely process the AIJs.[1]
3. Shutdown the Loader.[2]

If you did not follow this set of actions, you can (before re-starting the Loader) capture the missing data changes by backing up the AIJs with the /quietpoint qualifier, process the AIJ files with the static LogMiner, and then provide the LogMiner output file to the Continuous LogMiner Loader with JCC_LOGMINER_MODE = COPY.

Once your reorganization is complete, use the command with the override as shown to resume work.[3]

1. See "Upgrades and Changes" on page 449.
2. Execute "`$ jcc_clml_shutdown <Loadername>`" and wait for the batch job to complete.
3. Use the override only immediately after the reorganization. For later restarts, you will want to use the defaults and supply none of the override parameters to the run command.

Other Overrides

Any other override suggests a situation that is already confused. JCC recommends contacting JCC LogMiner Loader Support for assistance.

Unusual Restart Conditions

The Administrator may find occasion to worry about unusual restart conditions.

Processing Early in the Backed Up AIJs

If you are surprised by how far back in the AIJs the Loader restarts, there are two possible explanations.

1. You may have a long running transaction in the backed up AIJ files that has not yet committed in the target. The rmu/unload/after will have to start before that transaction.
2. You are running an older version of Rdb. In older versions of Rdb, the LogMiner read through all of the backed up AIJs whether it needed to or not. Recent versions of Rdb have been optimized to skip unneeded backed up AIJs.
3. You have experienced a documented interaction between AIJ backup and the Rdb LogMiner. See “Automated AIJ Backups” on page 483 and further references there.

Note that when you shut down a LogMiner Loader family, the Loader process waits for the current transaction write to complete before shutting down. If you have a large transaction with a performance issue on the target, this can take a long time.

You can discover more about the restart point using the command:

```
JCC_LML_DUMP_CHECKPOINT <Loader name> <target database> Rdb
```

Tracking AIJ Switches

Some Loader Administrators have found the log file entries confusing when the last micro quiet point (MQP) trails the current processing significantly. The Oracle Rdb LogMiner does not provide the AIJ sequence number for the journal that is currently being processed. Instead, the Loader reveals the AIJ sequence number from the AERCP in the current commit row. This AIJ sequence number is the last MQP.

Some update activity causes the AIJ sequence number (for the last MQP) that is found in the AERCP to significantly trail the current AIJ.

In attempting to clarify this in the log, the Loader offers the following style messages.

```
MQP in AIJ sequence number 1
MQP switched from AIJ sequence number 1 to 2
```

When the Loader is restarted from a checkpoint, an AIJ sequence number transition is detected, and the restart point has not been encountered, the Loader logs this additional information. For example, if the TSN is 15, the message would be

```
*** Restart TSN (15) not yet found.
```

Recovering from Exceptions

If an AIJ file is missing and you can't locate it, theoretically, there is no recovery.

If you can determine which data might have changed, you may be able to use the Data Pump to resolve the difficulty. This approach is likely to involve either some guessing or a great deal of data transfer. Losing an AIJ file before it is processed by the LogMiner is, still, a situation to avoid, if at all possible. (See "Data Pump" on page 505.)

If the problem is that someone started an AIJ back up while CLML was still catching up, getting the exception messages is an awkward interruption. However, recovery is accomplished by restarting again.

Setting the Restart Context for the First Time

When the Loader finds no highwater data, an opcom message is issued.[1] The opcom message requires a response. This is a strength, if the failure is a surprise. It is a nuisance on starting for the first time or starting from scratch, repeatedly, while developing an application.

In applications for which it makes sense to automate the opcom response and continue without interruption, a logical name can be set to create or quit as in:

```
DEFINE JCC_LOGMINER_LOADER_HW_RESPONSE CREATE|QUIT
```

1. See "Keyword: Operator" on page 265.

See also “Running the Loader for the First Time” on page 102.

Network Errors Using Rdb Remote

The JCC LogMiner Loader responds to network errors reported by rdb$remote. When the Loader detects a network error, it will disconnect, re-attach, retry, and show the following exception message in the log.

```
5-AUG-2003 06:15:38.35 20203231 IML CNTRL_METE   %dba_set_trans_rdb: Fatal Exception
5-AUG-2003 06:15:38.35 20203231 IML CNTRL_METE   %RDB-F-IO_ERROR, input or output errord
5-AUG-2003 06:15:38.36 20203231 IML CNTRL_METE   -SYSTEM-F-PATHLOST, path to network partner node lost
```

FIGURE 6. Network Exception

Retry Delay

The logical name JCC_LogMiner_Loader_retry_delay, when defined as a positive real value between 0.0 and 100,000 (exclusive), specifies the number of seconds that a Loader thread should wait after receiving an exception before trying again to send the data to the target. If the target is transaction-based,[1] the Loader will abort the current transaction and, potentially, disconnect from the target before waiting the retry delay number of seconds.

An example is

```
$define JCC_LOGMINER_LOADER_RETRY_DELAY 2.0
```

The display state for this is ‘d’ in the monitor. That is the screen will show (in the lower right):

```
- Loaders - 0 ------------------------------
- States  - d
```

Repairing Invalid AERCP Values

The AERCP (AIJ Extract Recovery Control Point) is generated by the Rdb Continuous LogMiner and is stored with other contextual information by the JCC Log-

1. The database targets support transaction-based processing. XML, JDBC, and file targets do not.

Miner Loader in the checkpoint data store. The checkpoint information is used by the JCC LogMiner Loader to ensure data integrity on restart. Therefore, on restart, the AERCP is passed, by the Loader, to the Rdb Continuous LogMiner to indicate a starting position.

On rare, but significant, occasions the AERCP has been reported to contain an invalid value of negative one (-1) for the AIJ sequence number field of the MQP (Micro-Quiet Point). To protect against this invalid value, the Loader saves the most recent valid AERCP. Then, if, while processing a commit record, the Loader detects an invalid value in the AIJ sequence number portion of the AERCP, the Loader replaces the invalid data with the last known valid MQP. This results in a valid AERCP for restart.

When this “MQP fixup” code is triggered, a message is generated in the Loader log file of the following format:

```
MQP fixup; invalid AIJ sequence number 4294967295 encountered
(TSN 33888 MQP 4294967295-0-33888 set to 2-0-33601)
```

FIGURE 7. AERCP FixUp

Special Restart - Skipping Updates on Purpose

This style of restart - intentionally - does NOT support complete replication from Source to Target. JCC recommends contacting support before using this option.

The company using the JCC LogMiner Loader that requested this option uses the Loader for a variety of applications. One application is responsible for updating an electronic sign with current information. For this application, there is NO interest in “old” data.

The application may be down due to network or other issues. When the application comes back up, there is an immediate need for “current” data. The special restart permits skipping data changes that are no longer needed.

Restart Options

The Loader will normally start at the point at which it last shut down. The Loader checkpoint record includes the information that tells the LogMiner where to resume. In order to ensure a prompt and accurate restart, the JCC LogMiner Loader stores the LogMiner generated AERCP in its checkpoint information.[1] At runtime, the Loader checkpoint values are transient, as new checkpoint information replaces older values when the Loader commits newer transactions to the target.

The Special Restart Option supports increased choices in restart points. It does not require that all restart points are dependent on quietpoint boundaries or checkpoints the Loader has recorded to mark the end of processing. Additionally, the new restart points may represent time intervals when the Loader was not running.

The following diagram "Timeline of Process Interruption and Restart" on page 448 illustrates the differences in the default restart and the special restart discussed in this section.

Support for the Special Restart Option

The Special Restart Option enables the Administrator running the Loader to cache valid AERCP restart points and the timestamp for when they occurred. The frequency of updating the cache with AERCP and timestamp is at the Administrator's directive and can be as frequent as once a minute.

The extra checkpoint information can, then, be used to request that the Loader restart processing at a given timestamp. Doing so may avoid processing records that are no longer current.

There are three utilities to support this functionality:

- jcc_lml_cache_checkpoint
  ("Cache Checkpoint Server" on page 437 and following)
- jcc_lml_cache_checkpoint_shutdown
  ("Cache Checkpoint Server Shutdown" on page 441)

1. See "Keyword: Checkpoint" on page 222.

- jcc_lml_dump_checkpoint_cache
  (“Dump Checkpoint Cache” on page 442)

Cache Checkpoint Server

The Cache Checkpoint Server, jcc_lml_cache_checkpoint, captures a specified number of Continuous LogMiner AERCP checkpoints for a Source database. These checkpoints consist of the AERCP and the timestamp. The timestamp can be used to choose where to resume processing on restart.

The Cache Checkpoint Server is a separate utility from the JCC LogMiner Loader. It is designed this way so that checkpoint information can be cached regardless of the state of any Loader operations.

The finite number of cache entries is implemented in a single, circular, overwriting cache. The oldest entries are replaced as new entries are written.

The interval specified is the *minimum* number of minutes between entries. Since the Cache Checkpoint Server is using commit timestamps and these do not necessarily occur at regular intervals, the exact interval between cache entries is uncertain. On restart, the cached entry chosen will be the latest one possible without going beyond the timestamp specified in the restart command. If there is one that was committed exactly at the chosen time, it will be used; if not, the last one before the chosen time will be used.

Usage of the Cache Checkpoint Server

```
$ jcc_lml_cache_checkpoint <source db> [interval][entries][override]
```

<source db>. required
The name of the source database

[interval]. optional
The minimum number of minutes between cached entries. If not specified, the value stored in the cache file is used. If a new cache file is being created, the default value of 10 (minutes) is used. The minimum is 1; the maximum is 1440. Invalid values are ignored and processing continues as if no value was specified.

[entries]. optional
The maximum number of entries to cache. If not specified, the value stored in the cache file is used. If a new cache file is being created, the default value of 144 is used. The minimum is 16; the maximum is 43,200. Invalid values are ignored and processing continues as if no value was specified.

[override]. optional
Allows the user to initialize the checkpoint file and start collecting the checkpoint cache data from either the AIJ backup files or the live AIJ files. Valid values are BACKUP and LIVE. If no value is specified, the cache file is read and the most recent cached value is used to position the restart. If a new cache file is being created and this parameter is not specified, the cache file is initialized and processing starts at the beginning of the LIVE AIJ files.

Example

This example shows the Cache Checkpoint Server being executed for the database that is defined by the logical name source_db. It will capture checkpoints at approximately one (1) minute intervals. The cache will maintain at most 1440 checkpoint entries. These settings will maintain a cache entry approximately every minute for a single day.

```
$ JCC_LML_CACHE_CHECKPOINT source_db 1 1440
```

Restrictions on the Cache Checkpoint Server

Currently, there is a restriction that once the cache file is created, the interval and entries values cannot change. If a different number of entries or interval is required, the existing file must be removed prior to starting with new values in a new file.

Since the Cache Checkpoint Server uses the Continuous LogMiner to capture the AERCP and commit timestamp data for the cache, there must be commit records available. For there to be commit records available, there must be active update transactions committed in the source database. The Continuous LogMiner requires that a table be specified in order to access commit records for all transactions. Any table can be specified.

However, since the LogMiner must process the data changes to this table, the table selected should have few or no updates. (As the update rate to the selected table

increases, so does the amount of unproductive work for this process.) The default chosen is JCCLML$HEARTBEAT because it conveniently meets the two criteria. It exists (if the heartbeat function is used) and it has a very low update rate. To override this default use the logical name JCC_LML_CACHE_USE_TABLE.[1]

There should not be more than one concurrent process running the Cache Checkpoint Server utility for a given source database at a time. The utility generates a unique name for a given database based on the database root file location on a given disk volume. An OpenVMS lock on a resource using this unique name ensures that only one version of the utility can run at a time for a given source database.

The generated unique name is also used as part of the default filename generated to store the cache of checkpoint data. The default cache filename is

```
jcc_tool_data:$<name unique to database>$.jcclml_cache
```

A logical name JCC_LML_-CHECKPOINT_CACHE_OVERRIDE can be used to specify a value for this file that is other than the generated default. This can prove useful when a database is restored and recovered, as restore and recovery will change the unique name while the data within the cache file is still valid.

Validity of the Cache Entries

The entries in the cache file are valid so long as:

- The AIJ backup files referenced remain available to be accessed during restart.
- If the database has been restored, all of the AIJ data has been recovered.
- After image journaling has not been disabled. (This applies regardless of whether it was subsequently re-enabled.)
- There are no metadata changes to the selected table that cause the internal record version to be changed.

Flexibility for the Cache Checkpoint Server

The Cache Checkpoint Serve can be run continuously. Alternately, it can be run

1. See “Loader Heartbeat and AIJ Backup” on page 475 for an explanation of heartbeat and “JCC_LML_CACHE_USE_TABLE.” on page 440 for additional restrictions.

only when there is an anticipated need. It is up to the Administrator running the Loader to determine when the Cache Checkpoint Server should be run and for what interval. One valid approach is to wait until there is an interruption and then start the cache checkpoint server.

Logical Names for the Cache Checkpoint Server

For compatibility, the Cache Checkpoint Server uses many of the same logical names as the Continuous LogMiner Loader. This section comments on the logical names that are specific to this utility, the logical names that are used in the same fashion for both, and one logical name that is not explicitly used by this utility, but must be understood.

JCC_LML_CHECKPOINT_CACHE_OVERRIDE. This logical name enables the user to determine the filename for the checkpoint cache. If it is used, the same logical name and value must be provided to both the JCC LogMiner Loader family and to the Dump Checkpoint Cache utility, in order to make the data available. See also “Restrictions on the Cache Checkpoint Server” on page 438.

JCC_LML_CACHE_USE_TABLE. This logical name enables the user to specify a table name that is different from the default. The table specified must not be an Rdb system table or a temporary table, but a physical table that is defined by the user. See also “Restrictions on the Cache Checkpoint Server” on page 438.

JCC_LOGMINER_LOADER_DB. This logical name is used by the Cache Checkpoint Server to provide the database name to the Continuous LogMiner callback routine. It is defined by this utility as the name of the database that is passed as the first parameter.

JCC_ADD_CML_SHARED_READ. This logical name is not explicitly used by this utility. However, if this utility is run simultaneously with a JCC LogMiner Loader processing the same source database, this logical name must be defined to avoid file access exceptions in the Continuous LogMiner sub-process. See “Finding AIJ Backups” on page 71.

Shared Logical Names. Logical names used for the same purposes in this utility as for the Continuous LogMiner are shown here with cross reference to other

discussions.

TABLE 1. Logical Names Used for the Special Restart and Elsewhere

Logical Name	Reference
JCC_AIJ_BACKUP_SPEC	"Review and Tips — Directories and Files" on page 66
JCC_ADD_CLM_TRACE	"The Log Files" on page 356
JCC_ADD_CLM_DEBUG	"The Log Files" on page 356
JCC_ADD_CLM_LOG	"The Log Files" on page 356
JCC_ADD_CLM_IGNORE_OLD_VERSION_TABLES	"A Rare Exception: Old Table Versions" on page 452
JCC_ADD_SORTWORK_FILES	
JCC_ADD_QUICK_SORT	
JCC_ADD_CLM_STATISTICS	"Modifying CLM Statistics Output" on page 355

Cache Checkpoint Server Shutdown

The cache checkpoint shutdown command sends a shutdown request to a Cache Checkpoint Server that is running for a specified database.

At any time, there will be only one concurrent process running the Cache Checkpoint Server for a given database.[1] The Cache Checkpoint Server Shutdown can be run on the same system that is running the Cache Checkpoint Server. Alternately, it can be used on any system in the same cluster, provided that system has the disk mounted where the database root file resides and uses the same physical device name.

There are no additional logical names introduced for Cache Checkpoint Server Shutdown.

Usage of Cache Checkpoint Server Shutdown

```
$ jcc_lml_cache_checkpoint_shutdown <source db>
```

1. See "Restrictions on the Cache Checkpoint Server" on page 438.

<source db>. required
The name of the source database.

Examples of Cache Checkpoint Server Shutdown

When there is no cache checkpoint server running for the database defined by the logical name source_db, the output will be

```
$ JCC_LML_CACHE_CHECKPOINT_SHUTDOWN source_db
No JCC LogMiner Loader Checkpoint Cache server running on this
database.
```

When there is a cache checkpoint server running for the database defined by the logical name source_db, the output will include the process ID of the Cache Checkpoint Server that was requested to shutdown.

```
$ JCC_LML_CACHE_CHECKPOINT_SHUTDOWN source_db
JCC LogMiner Loader Checkpoint Cache server shutdown (pid:31400523)
```

Dump Checkpoint Cache

The Dump Checkpoint Cache utility is used to examine the contents of the checkpoint cache.

At any time, there will be only one concurrent process running the Cache Checkpoint Server for a given database.[1] The Dump Checkpoint Cache can be run on the same system that is running the Cache Checkpoint Server. Alternately, it can be used on any system in the same cluster, provided that system has the disk mounted where the database root file resides and uses the same physical device name and can directly access the checkpoint cache file.

See the logical name JCC_LML_CHECKPOINT_CACHE_OVERRIDE on page 430.

Usage of Dump Checkpoint Cache

```
$ jcc_lml_dump_checkpoint_cache <source db> -
[timestamp|ALL|SUMMARY|CURRENT]
```

1. See “Restrictions on the Cache Checkpoint Server” on page 438.

<source db>. required
The name of the source database.

[timestamp|ALL|SUMMARY|CURRENT]. optional
CURRENT is the default. If CURRENT is specified or the parameter is not set, the utility dumps the cache header and the most recent record written to the cache. The DCL symbols are set for the CURRENT record.

If SUMMARY is specified, the utility dumps the cache header, the most recent record written to the cache and the least recent record written to the cache. The DCL symbols[1] are set for the CURRENT record.

If ALL is specified, the utility dumps the cache header, including all the records within the cache, most recent first. The DCL symbols are set for the CURRENT record.

If a valid OpenVMS timestamp is specified (in double quotes), the utility dumps the cache header, the most recent record written to the cache and the record in the cache that would be used for restart for the given time. The DCL symbols are set for the RESTART record.

Example of Dump Checkpoint Cache

This is an example of leaving out the optional parameter. Using the CURRENT parameter would produce the same results.

```
$ JCC_LML_DUMP_CHECKPOINT_CACHE source_db

JCC LML Dump Checkpoint Cache D03.04.00 (built 2-JAN-2012 11:00:41.87
Cache: JCC_TOOL_DATA:$$1$DGQ500_41B_3A3_0$.JCCLML_CACHE;1
Size: 00145 Rqst: 00144 Intv: 00001 Last: 00047 @ 2-JAN-2012 11:09:27.35
[00047] 23-DEC-201115:46:42.92 AERCP: 1-28-7-1944064-211282-211282
```

The symbols are set as follows:

```
$ show sym jcclml$*
JCCLML$AERCP == "1-28-7-1944064-211282-211282"
JCCLML$CACHE_ENTRY_NUMBER == "47"
JCCLML$CACHE_INTERVAL == "1"
```

1. See “DCL Symbols for Loader Exit Statuses” on page 465.


```
JCCLML$CACHE_REQUESTED == "144"
JCCLML$CACHE_SIZE == "145"
JCCLML$COMMIT_TAD == "23-DEC-2011 15:46:42.92"
JCCLML$WRITE_TAD == " 2-JAN-2012 11:09:27.35"
```

The second example shows the results of using the SUMMARY parameter.

```
$ JCC_LML_DUMP_CHECKPOINT_CACHE source_db summary

JCC LML Dump Checkpoint Cache D03.04.00 (built 2-JAN-2012 11:00:41.87)
Cache: JCC_TOOL_DATA:$$1$DGA500_41B_3A3_0$.JCCLML_CACHE;1
Size: 00145 Rqst: 00144 Intv: 00001 Last: 00047 @ 2-JAN-2012 11:09:27.35
[00047] 23-DEC-2011 15:46:42.92 AERCP: 1-28-7-1944064-211282-211282
[00000] 23-DEC-2011 14:55:19.88 AERCP: 1-28-6-256-36705-36705
```

The third example shows part of what is generated when the ALL parameter is used.

```
$ JCC_LML_DUMP_CHECKPOINT_CACHE source_db all

JCC LML Dump Checkpoint Cache D03.04.00 (built 2-JAN-2012 11:00:41.87)
Cache: JCC_TOOL_DATA:$$1$DGA500_41B_3A3_0$.JCCLML_CACHE;1
Size: 00145 Rqst: 00144 Intv: 00001 Last: 00047 @ 2-JAN-2012 11:09:27.35
[00047] 23-DEC-2011 15:46:42.92 AERCP: 1-28-7-1944064-211282-211282
[00046] 23-DEC-2011 15:45:42.92 AERCP: 1-28-7-1632512-206355-206355
[00045] 23-DEC-2011 15:44:43.72 AERCP: 1-28-7-1392896-202499-202499
[00044] 23-DEC-2011 15:43:43.19 AERCP: 1-28-7-1159168-198688-198688
[00043] 23-DEC-2011 15:42:43.14 AERCP: 1-28-7-962048-194040-194040
[00042] 23-DEC-2011 15:41:43.55 AERCP: 1-28-7-737536-190738-189021
[00041] 23-DEC-2011 15:40:42.93 AERCP: 1-28-7-482304-185490-185490
[00040] 23-DEC-2011 15:39:42.91 AERCP: 1-28-7-279808-182347-183139
[00039] 23-DEC-2011 15:38:43.79 AERCP: 1-28-7-101888-176308-176308
[00038] 23-DEC-2011 15:37:42.91 AERCP: 1-28-6-2923264-174620-173645
[00037] 23-DEC-2011 15:36:42.93 AERCP: 1-28-6-2706432-169559-169559
[00036] 23-DEC-2011 15:35:42.93 AERCP: 1-28-6-2472448-163280-163280
[00035] 23-DEC-2011 15:34:43.00 AERCP: 1-28-6-2264576-152306-159126
[00034] 23-DEC-2011 15:33:42.92 AERCP: 1-28-6-2098944-152001-158727
[00033] 23-DEC-2011 15:32:43.07 AERCP: 1-28-6-1940992-153104-154953
[00032] 23-DEC-2011 15:31:42.95 AERCP: 1-28-6-1786112-149106-152540
[00031] 23-DEC-2011 15:30:42.96 AERCP: 1-28-6-4096-45935-44524
[00030] 23-DEC-2011 15:29:43.89 AERCP: 1-28-6-4096-45935-41752
[00029] 23-DEC-2011 15:28:57.96 AERCP: 1-28-6-4096-45935-138081
[00028] 23-DEC-2011 15:27:52.21 AERCP: 1-28-6-4096-45935-105313
[00027] 23-DEC-2011 15:27:02.85 AERCP: 1-28-6-4096-45935-81761
[00026] 23-DEC-2011 15:25:42.92 AERCP: 1-28-6-4096-45935-55086
[00025] 23-DEC-2011 15:24:52.95 AERCP: 1-28-6-4096-45935-70497
[00024] 23-DEC-2011 15:23:43.87 AERCP: 1-28-6-4096-45935-41556
```

This example shows the dump utility with a timestamp and shows the record in the cache that would be used for restart, given the specified timestamp.

```
$ JCC_LML_DUMP_CHECKPOINT_CACHE source_db "23-DEC-2011 15:19"
```

```
JCC LML Dump Checkpoint Cache D03.04.00 (built 2-JAN-2012 11:00:41.87)
Cache: JCC_TOOL_DATA:$$1$DGA500_41B_3A3_0$.JCCLML_CACHE;1
Size: 00145 Rqst: 00144 Intv: 00001 Last: 00047 @ 2-JAN-2012 11:09:27.35
[00047] 23-DEC-2011 15:46:42.92 AERCP: 1-28-7-1944064-211282-211282
[00020] 23-DEC-2011 15:16:24.70 AERCP: 1-28-6-4096-45935-137083
```

For this example, the symbols are set as follows:

```
$ show sym jcclml$*
JCCLML$AERCP == "1-28-6-4096-45935-137083"
JCCLML$CACHE_ENTRY_NUMBER == "20"
JCCLML$CACHE_INTERVAL == "1"
JCCLML$CACHE_REQUESTED == "144"
JCCLML$CACHE_SIZE == "145"
JCCLML$COMMIT_TAD == "23-DEC-2011 15:16:24.70"
JCCLML$WRITE_TAD == " 2-JAN-2012 11:09:27.35"
```

Using the Cached Checkpoint Information for Restart

The addition of the Cache Checkpoint Server, and the information that it maintains, expands the set of restart points. A new special restart, using the cached checkpoint information, is included in Version 3.4 and later versions of the JCC LogMiner Loader.

In order to add this capability without disrupting existing installations, restart capability is added to the JCC_RUN_CLM_LML command[1] without changing the number of parameters. The command remains as follows, but with a new interpretation for the fifth parameter.

```
$ JCC_RUN_CLM_LML                   -
<source database name>              -
<LogMiner options file>             -
<LogMiner Loader Control File>      -
[<restart override tag>             -
<LogMiner restart context>          -
<Loader sequence number>            ]
```

The possible values for the fifth parameter ([LogMiner restart context]) are LIVE,

1. See also “The Full Run Command” on page 428.

BACKUP, RESTART, and the special restart option which is a timestamp for use with the Cached Checkpoint data.[1]

The timestamp must be specified in double quotes and must be in the past. The timestamp may be an absolute, delta, or combination date.[2] The passed date will be converted to an absolute date for processing.

As with the BACKUP and LIVE values, specifying a timestamp for the fifth parameter will override the usual stored checkpoint information that the Loader wrote in previous executions. As with the BACKUP and LIVE values, the Loader Sequence Number (LSN)[3] from the checkpoint file *will* be used unless otherwise overridden.

When the timestamp specified does not exactly match a timestamp in the checkpoint cache, the Loader will discard the records prior to the specified timestamp.

The Loader will exit with an exception when a timestamp is specified for the fifth ([aij option]) parameter of the jcc_run_clm_lml command and any of the following cases apply:

- The checkpoint cache file does not exist in the expected location
- The checkpoint cache file security does not permit the Loader to access it
- The timestamp specified is before the first cached timestamp in the cache file
- The timestamp specified is in the future
- The selected cache record for restart requires an AIJ backup file that is not accessible.

Before using the timestamp with the run command, also review the logical names discussed in “JCC_LML_CHECKPOINT_CACHE_OVERRIDE.” on page 440 and “JCC_ADD_CML_SHARED_READ.” on page 440.

1. The JCC_RUN_CLM_LML command is explained in “The Full Run Command” on page 428.
2. See “Finer Control of the Start Time” on page 91 for a discussion of absolute, delta, and combination dates in start times.
3. See “Recovery” on page 36 for a discussion of LSN and restart.

Examples

This section offers an example of the run command. The example should serve as a summary of the Cache Checkpoint Server and its related technologies.

The example assumes that the logical name Source_db points to the source database, lm_options points to the options file for the LogMiner, and lml_cf points to the Control File or initialization file for the Loader.

```
$ jcc_run_clm_lml source_db                        -
                  lm_options                       -
                  lml_cf                           -
                  ""                               -
                  "1-Feb-2012 01:23.45.67"
```

For this example, the optional parameters are:

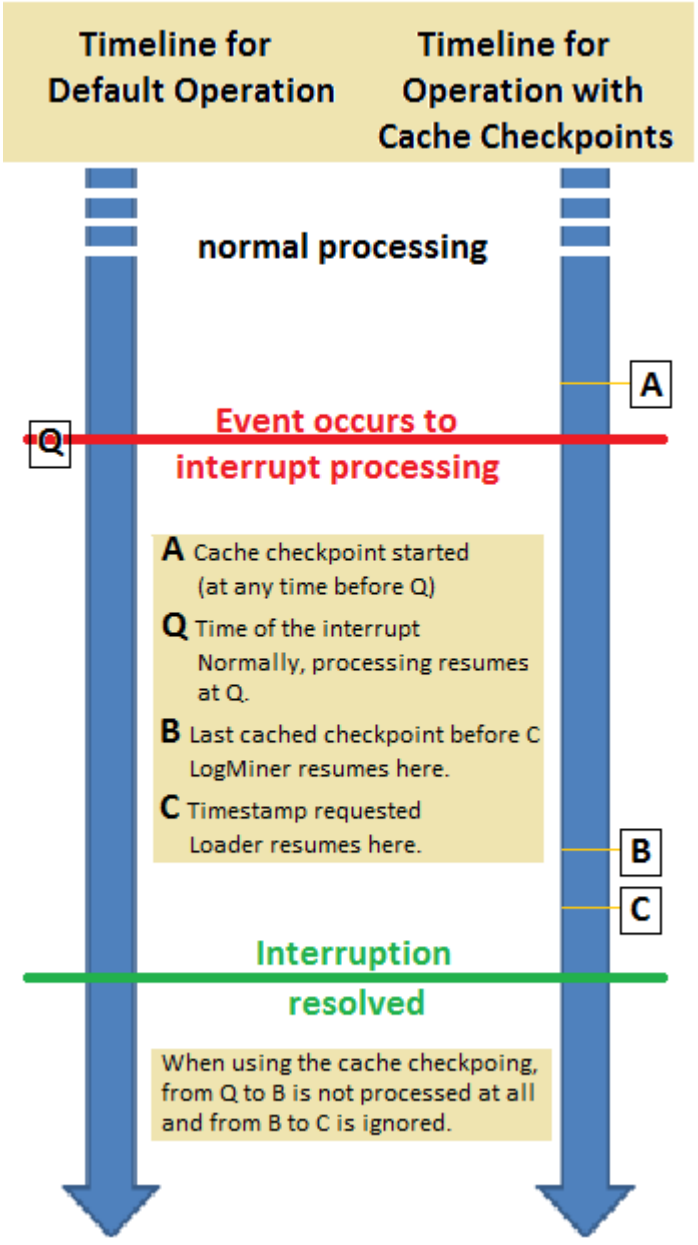
- "" (Alternately, the override parameter could be CHECKPOINT.)
- a missing optional parameter for LSN (Loader Sequence Number)
- an absolute date to match to the cache checkpoint file to choose the restart point

These optional parameters override the usual restart. The usual restart would be at the point that Loader processing stopped and this is at the point determined by the timestamp parameter.

Other valid run commands that utilize the cache checkpoint Server could include the empty string shown or CHECKPOINT for the override option parameter and any of the date formats supported for the aij option parameter. Other examples of valid values for the aij option parameter (the fifth parameter) are any absolute, delta, or combination date. (See “Finer Control of the Start Time” on page 95 for a discussion of date formats.)

The diagram to follow illustrates the timeline for both the default recovery operation and for a use of the cache checkpoint approach.

FIGURE 8. Timeline of Process Interruption and Restart



Default operation results in processing being resumed where it was interrupted (at Q). See “Rdb Issues” on page 412.

The cache checkpoint is started at A in this diagram. It could be started anywhere prior to the cache checkpoint to be used at restart. C is the timestamp requested with the restart command and B is the cached checkpoint that is the latest in time prior to C. The transactions between the interruption and B are skipped. LogMiner processing begins at B and the Loader ignores the transactions until the timestamp C is reached.

Note that processing between Q and C cannot be safely recovered later. Use cache checkpoint only when you do not need those transactions. This option is not suitable for most replication, auditing, or archiving and is only suitable for special cases.

Upgrades and Changes

A smoothly running JCC LogMiner Loader architecture implies a stable source, a stable target, and consistent software. When one of these must be changed, care is required. Each is covered separately here, but there are similarities in how to successfully make changes with minimal disruption.

Basically, the steps are:

1. Fully process all AIJs.
2. Stop all work.
3. Do the special steps for upgrade or change.
4. Start everything up again.

The details do vary and care is recommended.

Upgrading Rdb

Since the AIJ format can change between versions of Rdb, care is recommended when upgrading.

1. Make certain that you have a good, recent database backup.
2. Shut down the application(s).

3. Let the Loader families mine all of the data that has accumulated so that none is left to process under the new version.
4. Shutdown the Loader sessions.
5. Backup the AIJs and rename the backup files so that they will not be examined with a new version of Rdb after the re-start by the LogMiner.
6. Upgrade the database(s).
7. Verify that all of the active AIJs are visible.
8. Start an online database backup.
9. Modify any command procedures that explicitly set their Rdb version to the old version.
10. Start the Loaders in the *live* journals. Skip backup journals completely. [1]
11. Start the application(s).

Exception. If the re-naming of the AIJs does not occur, it is possible to fall into a very interesting exception. If a Loader shuts down due to a problem with the target or for any other reason and there is a backup of the AIJ before it is restarted, the Control process will start looking for the correct starting point (correct AERCP) in the backup journals. That is as it should be, unless there is a journal in that path that was written with a format that is not supported by the new version of Rdb. Then, you will see this message:

```
%RMU-F-FILACCERR, error reading journal file DISK$B4_AIJBK_01:[SUB-
RDB04_AIJ_BACKUP]SUBRDB04_130165.AIJ;1
-RMU-F-BADAIJVER, after-image journal version is incompatible with
the runtime system
%RMU-F-FTL_RMU, Fatal error for RMU operation at 28-MAY-2003
07:43:38.40
```

FIGURE 9. Loader Finds AIJ with Old Metadata

Upgrading the Loader

New versions of the JCC LogMiner Loader are released whenever sufficient new features or corrections justify it. The Loader endures extreme regression testing before a new release. Support will often recommend an upgrade to the latest version. The steps to upgrade are straightforward. As always, careful AIJ management is important.

1. See step 2 which discusses “emptying” the journals before the upgrade.

The steps are:

1. Read the release notes.
2. Restore the backup save set to the appropriate directory structure. This can be a parallel directory structure to the current installation. See “Multi-Version Support” on page 56.
3. Verify that the license key is edited into the local license startup procedure in the new directory structure.
4. Start the new version of the JCC LogMiner Loader.
5. Shutdown existing Loader sessions.
6. Do any edits needed to use the new version.[1]
7. Start the Loader sessions.

The steps should take less than an hour per system and do not require down-time for the source database.

Note also that JCC has generally been able to keep the Loader backwardly compatible and any exceptions have been called out in the release notes.

Source Database Reorganization

Successful interruption of the Loader operation for a database reorganization has requires a similar approach. See Figure , “Restarting in the Live AIJ after DB Reorganization,” on page 430.

Metadata Changes in the Source Database

When the metadata in the source database must be changed, either the change will be immaterial to the Loader or care will be required.

If the metadata change does not touch — directly or indirectly — any column or table that is represented in the LogMiner options file or in the Loader Control File, there is no change to make.

1. Adjust any settings that tie the operation to a specific version. For example, a phrase such as “$@device_name:[JCCLML_V344.COM]JCC_LML_USER.COM 3.4” will need to be edited to use version 3.5.

If the metadata change does touch a column that is represented in the LogMiner options file or in the Loader Control File, it will be necessary to

1. Shut down the application
2. Process all existing AIJs
3. Backup AIJs
4. Shut down the Loader
5. Make the metadata changes
6. Update the Control File and/or LogMiner options file
7. If the target is a replication, it may be necessary to update the target metadata.
8. Move or rename the backup AIJs so they will not be visible via the logical name JCC_AIJ_BACKUP_SPEC.
9. Restart CLML in the live journal.
10. Restart the application.

Metadata Changes in the Target Database

If the target is a database, it may be desirable to add indexes to improve query performance. Indexes can be added without disrupting Loader operation.

Remember that triggers that have already fired in the source should not be added to the target and that constraints can cause failures if rows are updated in unexpected order.

Metadata Changes and Mapping the Source to the Target

On the subject of metadata changes, it may also help to consult the chapter on variance between the source and the target, “Schema and Data Transforms” on page 489.

A Rare Exception: Old Table Versions

Correct definition of the source metadata, including the version number for the metadata, is required. However, on *rare* occasion, a verb rollback of a modification of a row may result in Rdb’s writing an old version of a row to the AIJ, even though the transaction did not actually complete a successful modification. In this rare instance, the user of CLM and the Loader needs some way to get past the issue.

CLM was modified as is reflected here in RMU help:

> Ignore=Old_Version[=table-list]
>
> Specifies optional conditions or items to ignore
>
> The RMU Unload After_Journal command treats non-current record versions in the AIJ file as a fatal error condition. That is, attempting to extract a record that has a record version not the same as the table's current maximum version results in a fatal error.
>
> There are, however, some very rare cases where a verb rollback of a modification of a record may result in an old version of a record being written to the after-image journal even though the transaction did not actually complete a successful modification to the record. The RMU Unload After_Journal command detects the old record version and aborts with a fatal error in this unlikely case.
>
> When the Ignore=Old_Version qualifier is present, the RMU Unload After_-Journal command displays a warning message for each record that has a non-current record version and the record is not written to the output stream. The Old_Version qualifier accepts an optional list of table names to indicate that only the specified tables are permitted to have non-current record version errors ignored.

The JCC LogMiner Loader supports this notation with a logical name to specify the list of tables. The logical name is JCC_ADD_CLM_IGNORE_OLD_VERSION_TABLES.

If the logical name is defined, the CLM command line will include 'ign=old=(<the value of the logical name>)'. Since the DCL command line is limited, it may not be possible to specify many tables in the comma separated list. If there is more than one table, enclosing the logical value in quotes is important, as only the first translation is used. It is also possible to set the logical name to '*' to include all tables.

For example

```
$ define JCC_ADD_CLM_IGNORE_OLD_VERSION_TABLES "t1,t2"
```

or

```
$ define JCC_ADD_CLM_IGNORE_OLD_VERSION_TABLES *
```

Testing for a Difficulty with Older Versions

For most Administrators, the following pertains to difficulties that do not apply.

When a version of the generation procedure that pre-dates Loader Version 2.2.8 has been used *and* the source metadata has been modified with an SQL statement that uses BEFORE or AFTER in adding or altering a column, there may be errors in the data written to the target. To test whether your source database metadata has this column reordering, use the analysis tool jcc_order_analysis.sql.[1]

This procedure is an example of the upward compatibility of the JCC LogMiner Loader. *Note that, if the initial generation was with the procedure in Loader Version 2.2.8 or later, there is no Loader issue for reordered columns.*

OpenVMS and the Loader

If you are using the JCC LogMiner Loader, your source database is Rdb and it and the Loader are running on OpenVMS. Included in this section are some of the topics that are related to OpenVMS.

Use of OpenVMS Clusters

Should you need to make control of the Loader session possible from a cluster node different from the one on which it is running, you will need the command procedure JCC_CLML_COMMUNICATE. This procedure supports sending LogMiner Loader commands across nodes in a cluster.

This command procedure is used by:

JCC_CLML_START_THREAD
JCC_CLML_STOP_THREAD
JCC_CLML_SHUTDOWN
JCC_CLML_MAXIMUM_THREADS
JCC_CLML_MINIMUM_THREADS
JCC_CLML_REOPEN_LOG

Note that if DECnet proxy logins are not enabled, these commands will work only if the Loader session is running on the current node.

1. This procedure is found in the directory jcc_tool_sql in the kit for version 2.2.8 or later. Before running the procedure, you will need to define the logical name target_db to be the database to test.

Finding Sessions in the Cluster

Because some systems can become complex, the Loader provides tools for locating portions of your application. The JCC_FIND_LML_PROCESSES procedure shows which processes are running. With the addition of the parameter 'SESSIONS', the Loader will output one line per Loader family running in the cluster and will show which system is running the family.

Example. This example shows ten Loader sessions.

```
$ jcc_find_lml_processes sessions
Node   Username     Loader Name
NODE1  REG          OCI_NGATE
NODE1  REGF         REGTESTSRDB
NODE1  REGF         REGTESTAPI
NODE1  REGF         REGTESTFLTR
NODE1  REGF         REGTESTORA
NODE1  REGF         REGTESTRDB
NODE1  REG          SQS_INET
NODE2  REGF2        REGTESTAPIJ
NODE2  REGF2        REGTESTORAJ
NODE3  REGF1        REGTESTPRDB
10 running Loader sessions
```

FIGURE 10. Finding Sessions in the Cluster

Directory Security

Some installations have higher security requirements for the files in the Loader directory trees than what is provided by default. With release 3.1 of the Loader

- The default security is increased, to eliminate world write capability to most of the files and directories in the Loader directory tree.
- Control of the security model is provided through inclusion of the procedure jcc_tool_com:jcc_tool_security.com.

If the default security model for the Loader is not acceptable, then jcc_tool_com:jcc_tool_security.com should be copied to jcc_tool_local:jcc_tool_security.com and modified to provide the required security. If jcc_tool_local:jcc_tool_security.com exists, it will be used during startup instead of the default procedure jcc_tool_com:jcc_tool_security.com.

If you do not copy the file to jcc_tool_local before making your changes, your changes may be overwritten the next time you upgrade.

Controlling Generated OpenVMS Process Names

Some systems management tools don't handle embedded blanks in process names. The Loader supports a logical name that resolves the issue by eliminating the embedded blanks. Declare the process name separator for CLML with the logical name, JCC_CLML_PROCESS_NAME_SEPARATOR.

The Process Name Separator support is provided for your convenience. If you do not have a problem with embedded blanks in process names, don't bother with this support.

The example shows a directory with blanks in the process names and three different uses of the logical name to replace the blanks.

```
$ show sys/proc=*SUBRDB05
OpenVMS V7.3  on node ATLAS   6-NOV-2002 12:21:50.44  Uptime  7 18:54:58
  Pid    Process Name    State  Pri      I/O       CPU       Page flts  Pages
20201F5F ||0 SUBRDB05    LEF      5     1115   0 00:00:00.09       345    276  S
20201F61 CLM SUBRDB05    HIB      6     1069   0 00:00:00.30      1224   1436  S
20201664 ||1 SUBRDB05    LEF      6      892   0 00:00:00.09       340    271  S
$ jcc_clml_shutdown SUBRDB05
$ define/system JCC_CLML_PROCESS_NAME_SEPARATOR "+"
$ show sys/proc=*SUBRDB05
OpenVMS V7.3  on node ATLAS   6-NOV-2002 12:22:14.57  Uptime  7 18:55:22
  Pid    Process Name    State  Pri      I/O       CPU       Page flts  Pages
20201D74 ||0+SUBRDB05    LEF      5     1100   0 00:00:00.10       345    276  S
20201F75 CLM+SUBRDB05    LEF      6     1071   0 00:00:00.29      1248   1426  S
20201D76 ||1+SUBRDB05    LEF      6      893   0 00:00:00.04       340    271  S
$
$ jcc_clml_shutdown SUBRDB05
$ define/system JCC_CLML_PROCESS_NAME_SEPARATOR "-"
%DCL-I-SUPERSEDE, previous value of JCC_CLML_PROCESS_NAME_SEPARATOR has been superseded
$ show sys/proc=*SUBRDB05
OpenVMS V7.3  on node ATLAS   6-NOV-2002 12:30:47.70  Uptime  7 19:03:56
  Pid    Process Name    State  Pri      I/O       CPU       Page flts  Pages
20201C82 ||0-SUBRDB05    LEF      5     1151   0 00:00:00.09       345    276  S
20201E83 CLM-SUBRDB05    HIB      6     1069   0 00:00:00.37      1198   1426  S
20201D84 ||1-SUBRDB05    LEF      6      892   0 00:00:00.12       340    271  S
$ jcc_clml_shutdown SUBRDB05
$ define/system JCC_CLML_PROCESS_NAME_SEPARATOR ":"
%DCL-I-SUPERSEDE, previous value of JCC_CLML_PROCESS_NAME_SEPARATOR has been superseded
$ show sys/proc=*SUBRDB05
OpenVMS V7.3  on node ATLAS   6-NOV-2002 12:31:11.11  Uptime  7 19:04:19
  Pid    Process Name    State  Pri      I/O       CPU       Page flts  Pages
20201F88 ||0:SUBRDB05    LEF      4     1097   0 00:00:00.13       345    276  S
20201D89 CLM:SUBRDB05    LEF      6     1070   0 00:00:00.31      1357   1294  S
20201D8A ||1:SUBRDB05    LEF      6      892   0 00:00:00.11       340    271  S
$ jcc_clml_shutdown SUBRDB05
```

FIGURE 11. Process Name Separator

Tuning OpenVMS for JDBC

See “Systems Tuning Using JDBC as the Loader Target” on page 148.

Operator Classes and Tardiness Messages

The Loader provides control for which operator classes receive opcom messages. The operator classes supported are

- cards
- central
- cluster
- devices
- disks
- license
- network
- security
- tapes
- oper1, oper2, ..., oper12

Setting the Operator Class

The keyword OPERATOR can be used to set one or more operator classes to receive messages. The default is central. Any number of classes can be specified in a comma separated list or ALL may be specified.

OPCOM messages generated by the license and command line validation routines are generated before the Control File is processed. Therefore, these messages will be sent to ALL operator classes.

See also “Keyword: Operator” on page 274.

Tardiness Messages

The Loader statistics program generates messages if a set threshold is reached for “tardiness.” Tardiness is defined as being more than the number (specified as the

tardiness threshold) of seconds lag between the update on the source and passing the changes to the target.

```
jcc_lml_statistics <LoaderName>                    -
                   [refresh seconds]               -
                   [brief|full|detail|csv]         -
                   [tardy threshold[operator class]]
```

A message is also generated when the Loader catches up again. See “Tardiness Threshold optional” on page 315.

By default, tardiness messages are sent to the CENTRAL operator class. However, a parameter enables setting the operator class to any desired collection or to set the operator class to ALL. The same set of operator classes are available as for failure messages. See also “Keyword: Operator” on page 265 and “Get the Current AIJ Sequence Number” on page 378.

Choosing the Tardiness Indicator

The Loader’s ability to post messages when processing falls behind is enhanced with control over what will trigger the tardy alert. The logical name JCC_LOGMINER_LOADER_STAT_TARDY_FIELD can be set to any of several values to cause the tardy OPCOM messages to be generated for the interval that is most important to the goals of the specific environment.

Valid values for JCC_LOGMINER_LOADER_STAT_TARDY_FIELD, along with the messages that they trigger, are shown in the chart to follow. TrailOutput is the default and is the value used in previous releases.

TABLE 2. Values for the Tardiness Logical Name

Value	Tardy Message	Caught-up Message
TrailOutput	JCCSTAT: JCC Loader ‘<LoaderName>’ output is trailing realtime by <TrailOutput> seconds	JCCSTAT: JCC Loader ‘<Loader-name>’ is below tardy interval of <tardy> second; output trailing real-time by <TrailOutput> seconds
TrailInput	JCCSTAT: JCC Loader ‘<LoaderName>’ input is trailing realtime by <TrailInput> seconds	JCCSTAT: JCC Loader ‘<Loader-name>’ is below tardy interval of <tardy> second; input trailing real-time by <TrailInput> seconds

TABLE 2. Values for the Tardiness Logical Name

Value	Tardy Message	Caught-up Message
Latency	JCCSTAT: JCC Loader '<LoaderName>' latency is <Latency> seconds	JCCSTAT: JCC Loader '<Loader-name>' is below tardy interval of <tardy> second; latency is <Latency> seconds
LatencyInput	JCCSTAT: JCC Loader '<LoaderName>' input latency is <LatencyInput> seconds	JCCSTAT: JCC Loader '<Loader-name>' is below tardy interval of <tardy> second; input latency is <LatencyInput> seconds
LatencyOutput	JCCSTAT: JCC Loader '<LoaderName>' output latency is <LatencyOutput> seconds	JCCSTAT: JCC Loader '<Loader-name>' is below tardy interval of <tardy> second; output latency is <LatencyOutput> seconds

For example, the following shows definition of the logical name, an exception message on the definition, and a tardy message and a caught-up message.

```
$ define JCC_LOGMINER_LOADER_STAT_TARDY_FIELD LatencyOutpu
$ JCC_LML_STATISTICS regtestrdb 6 d

JCC LogMiner Loader Statistics D02.01.01 (built  7-APR-2004 15:08:54.08)

%jcc_lml_statistics: Logical JCC_LOGMINER_LOADER_STAT_TARDY_FIELD = LATENCYOUTPU.
%jcc_lml_statistics: Invalid value found for logical.
%jcc_lml_statistics: Valid values are TrailOutput, TrailInput, Latency, LatencyInput
and LatencyOutput.
%jcc_lml_statistics: Aborting.
%DBA-E-INVALID_DATA, Invalid input data.
$ define JCC_LOGMINER_LOADER_STAT_TARDY_FIELD latency
%DCL-I-SUPERSEDE, previous value of JCC_LOGMINER_LOADER_STAT_TARDY_FIELD has been
superseded
$ JCC_LML_STATISTICS regtestrdb 6 d

JCC LogMiner Loader Statistics D02.01.01 (built  7-APR-2004 15:08:54.08)

%jcc_lml_statistics: JCC_LOGMINER_LOADER_STAT_TARDY_FIELD = "LATENCY".

Waiting  3.0 seconds JCC LogMiner Loader process 'REGTESTRDB' to start.

%jcc_lml_statistics: JCC LogMiner Loader process 'REGTESTRDB' is not currently
running.
Exiting...
```

FIGURE 12. Example of Tardiness Messages

Naming and Placing the Log Files

The Control process acts as a logging sink for the LogMiner and for the Loader threads. In turn, it writes the logging information into files, one per client. As data is written, the Control process appends a time stamp to each line.

Re-directing the Log Files

The files are written to the directory JCC_TOOL_LOGS. The logical name JCC_TOOL_LOGS is defined in the JCC_DBA_STARTUP procedure. This logical name may be redefined as necessary, but must be defined in process context.[1]

Closing and Re-Opening Log Files

You can direct the Control Process to close existing log files and open new ones with the command

```
$ jcc_clml_reopen_log <loader name>
```

Providing Sufficient Disk Space for the Log Files

The JCC LogMiner Loader files are not large. However, the Loader has numerous logging options.[2] Depending on which of these options you find useful, the Loader may write voluminous log files. The disk you choose should have sufficient space to accommodate those files.

How much space? It will depend significantly on the logging options as well as the dynamic nature of the threads. More threads require more logs. Starting and stopping threads also requires more disk space for logs. How much more depends on the load of the particular database and the start and stop thresholds. Echoing DCL commands and Control Files in the logs (something strongly recommended by JCC) also requires some space. The impact of the Control Files being echoed is directly related to the number and complexity of tables defined in the Control Files. Disk space for the logs also increases when CLML is started and stopped on a frequent basis.

1. For a discussion of process context, see “Prerequisites to Logical Name Maintenance” on page 553.
2. See “Keyword: Logging” on page 247.

Controls for the Filter Database

To support the advanced features of filtering[1] and data transforms,[2] the JCC Log-Miner Loader utilizes a small Rdb database.

Use of FilterMap and MapResult provides advanced features. Neither is required. The filter map database is not utilized until a row is received that must be processed using it.

The name and directory placement of the database can be controlled with logical names.

Directory Placement of the FilterMap Database

Directory placement is controlled with the logical name JCC_LOGMINER_LOADER_FILTER_DIR.

The name specified must be a valid device and directory. The Loader will append "<Loadername>.rdb" to create the full file specification.

Naming the FilterMap Database

The logical name JCC_LOGMINER_LOADER_FILTER_NAME enables the user to select the name of the filter database. The value of this logical name will override the default value. [3]

This feature is useful in environments that include multiple Loader families on the same system/cluster, as all of the Loader families (or a subset) can use the same filter database. As usual, if this feature is not important, the default should be used.

An example of naming the filter database is shown in the following.

1. See "Keyword: FilterMap" on page 233.
2. See "Keyword: MapResult" on page 257
3. By default, the Loader creates a filter database for each Loader family and names the database using the LoaderName.


```
$ define jcc_logminer_loader_filter_dir jcc_tool_data:
$ define jcc_logminer_loader_filter_name ivp_fxml
$ jcc_logminer_loader

JCC LogMiner Loader D02.02.00 (built  7-JUL-2004 14:56:42.59)

This application is licensed to JCC.
Start time: Wed Jul  7 15:07:07 2004

jcc_logminer_loader_filter_dir is set to JCC_TOOL_DATA:
jcc_logminer_loader_filter_name is set to IVP_FXML
o
o
o
```

FIGURE 13. Naming the Filter Database

The database that will be used by the loader in the example will be “JCC_TOOL_-DATA:IVP_FXML.RDB”. If the database does not exist, it will be created.[1]

The “.RDB” is appended to the value specified by the logical name. If the value of this logical name contains a period (“.”), the string will be truncated to that position and “.RDB” will be added to the truncated value.

The database so created contains parameters set by the Loader. You can modify these default parameters. If, for instance, if you are going to use the database for multiple filtering sessions, you may need to modify the filter database to add more user slots. Reference the Rdb documentation on "ALTER DATABASE" for how this is done.

For additional material, see “Keyword: FilterMap” on page 233.

Transaction Control for the FilterMap Database

Two logical names will affect the transaction model used for the FilterMap database.

- JCC_LML_RESULT_TXN_TYPE sets the transaction type. The default is "READ ONLY". Possible values include "READ WRITE ISOLATION LEVEL READ COMMITTED" and any other supported Rdb transaction model.

1. See also “Keyword: FilterMap” on page 233.

If MapResult is to store data in the FilterMap database, READ WRITE will be required for that.

- JCC_LML_RESULT_TXN_PRESTART sets whether to prestart transactions. To prestart transactions, set it to 1, t[rue] or T[rue]. The default or anything other than 1, t, or T disables the prestart.

These settings only apply when using the MapResult and FilterMap database. The defaults are good for most cases. In addition to enabling writes, the settings are available to provide performance options when performance demands are high.

Controlling the Loader and the LogMiner

You have many options for how the JCC LogMiner Loader behaves and the kit includes many tools. See also "Extended Examples and Tools" on page 547.

Generating the Control File

The Loader kit offers assistance in generating the Control File. See "Building the Metadata Control File" on page 222.

Logical Name Controls for Loader Procedures

Logical names are used in many ways with the Loader. For a summary list, see the appendix, "Logical Names" on page 585.

The Loader kit includes a facility to aid your maintenance of logical names. The purpose of this facility is to allow you to control various Rdb run-time parameters on a Loader by Loader basis and to distinguish performance characteristics of Loader threads and LogMiner threads.

Understanding this tool is not required for most installations. However, for complex environments, it may prove helpful. See "Logical Name Controls for Loader Procedures" on page 553.

Exception Messages

For a list of exception messages, their meaning, and what actions can be taken to rectify the problem, see the file

```
jcc_tool_source:jcclml_msg.doc
```

Starting a Loader with the Same Name

When a Loader is started with a Loadername that is already in use, the underlying problem is reported so that the Administrator can correct the issue.

```
%jcc_continuous_logminer_loader: Unable to create shared memory for statistics
%DBA-I-SECTION_REMAP, Existing section mapped for write
```

FIGURE 14. Message for Second Loader with a Name Already Used

Re-tries and Exceptions

Some failed actions should be retried. For example, if the Loader attempts to open a file and another process has it open, re-trying may resolve the issue. Some failed actions will not resolve. For example, if an action cannot complete because the account used lacks sufficient privilege, re-trying will not succeed.

The logical name JCC_LML_ACTIVATION_LOG_ATTEMPTS (with a default of 200) can be used to adjust the number of re-tries. See “Activation Log” on page 368.

Creating a Bugcheck Dump

For diagnostic purposes, a Loader bugcheck may be required. To force a bugcheck, use

```
JCC_CLML_BUGCHECK
```

When this procedure is used, the bugcheck information will be written to the LML thread log file(s) and then the process will exit.

Generally, this will only be used at the direction of JCC LML support. To stop the Loader gracefully, see “Shutting Down Continuous LogMiner” on page 75.

DCL Symbols for Loader Exit Statuses

The procedure JCC_TOOL_SOURCE:DBA_CONDITIONS.COM contains DCL symbol definitions for every possible Loader exception status. With these definitions, you may symbolically test Loader exit statuses in DCL and make decisions as required. This procedure eliminates the need for hard-coding exception message values into the calling DCL.

Determining LogMiner Status

You can determine whether Rdb has the LogMiner enabled with the command

```
$ pipe rmu/dump/header=journal <your db name>      -
    | search sys$pipe logminer
```

You should see:

```
      - LogMiner is enabled
      - Continuous LogMiner is enabled
```

If you get any other status reported, it may be that you failed to perform a full database backup immediately after issuing an RMU set LogMiner command.

Tuning Considerations

There are aspects of running the Loader that require an understanding of OpenVMS and Rdb. Procedures in the kit and logical names are provided to minimize requirements on the Administrator. Some specific tips are assembled in this section.

Concurrent Reads

The logical name jcc_add_clm_shared_read must be defined to ‘TRUE’ to support the multiple processes reading concurrently from the same backup AIJ files. If you have only one Loader reading from the mailbox, the logical name is not required.

Buffered I/O

Buffered i/o is one of the resources consumed when logging Loader activity. Truly excessive logging will consume more buffered i/o than is desirable in a production

environment. See “Keyword: Logging” on page 247 for a discussion of which logging settings are inappropriate except when debugging.

Tuning for XML and JDBC Targets

To publish transactions to the target, XML and JDBC targets require sufficient resources for processing multiple copies of the messages generated. The OpenVMS account used to run the Loader must be configured with generous amounts of page file quota and appropriate working set quotas. How much is enough will depend on the size of the transactions to be moved.

Tuning for Other Targets

Additional tuning tips are provided in the sections specific to a given target.

Space for Sorting

Sorting is also discussed in “Performance Considerations” on page 381.

Rdb sortwork files use fixed-length records. Because it is impossible to predict what record types will be sorted within any particular transaction, sort has to be able to handle the largest possible record of all the tables being extracted. So if most of the records are 100 bytes but even one table has a record up to 5000 bytes, sort has to use space for 5000 byte records always. For this reason, surprisingly large sort work spaces can be required.

Locking and the Mailbox

The Continuous LogMiner (CLM) process reads from the AIJ and writes the data for a transaction to an OpenVMS mailbox, when it reads a transaction commit record. The Loader (CLML) process reads from the mailbox and, when it gets the transaction commit, writes it to the target.

Generally, locking and this mailbox is not a problem. However, there are Administrator choices that can make it an issue.

- ALS, which is a requirement for Hot Standby, is not a requirement for using LogMiner; but it can help. The ALS takes over the task of writing to the AIJ files so individual database attaches do not have to do the writes. This speeds up writes to the AIJ and can reduce contention around AIJ related locks.

- Using the source database as the target such that the results of CLML processing is written back to the same database can cause locking. If the CLM is waiting to write to the mailbox because the mailbox is full and the CLML is updating the database and needs to write to the AIJ, the CLML will not be able to get the lock to write to the AIJ and a circular situation similar to a deadlock can result. If an AIJ backup also starts, it will feed the gridlock because it also needs to get an AIJ lock to force an AIJ switch.
- An unusual increase in transactions or in the size of transactions can fill the mailbox and contribute to locking issues.
- Database tuning choices that open the database on more than one node, improperly balance page size and block size, create awkward query solutions, and others can add to locking difficulties.

Space in the CLM Logging Mailbox

The JCC Continuous LogMiner Loader uses OpenVMS mailboxes to off-load the overhead of writing log files from the children processes (CLM and LML) to the parent (CTL) process. This method supports user control of several aspects of logging, including combining log files, reusing log files, and reopening log files.

However, this logging model is implicated in a hang involving CLML (Continuous LogMiner Loader), CLM (Continuous LogMiner), and the source database when both the heartbeat function and CLM statistics output are enabled. The hang is due to an AST conflict between the logger and the database caused by an insufficient size for the CLM logging mailbox in the parent (CTL) process.

The mailbox size describes the number of 2K byte records that the mailbox can buffer. In Release 3.3.0, the size of the mailbox was increased. The default value is set as the greater of 1024 or the smallest power of 2 that is greater than (the number of defined tables plus twenty).

In addition, the logical name JCC_CLM_LOG_MAILBOX_SIZE can be used to make the mailbox size even greater than the default. If the value of the logical name is less than the calculated default, the default will be used.

Exception Information when Virtual Memory Exceeded

The JCC LogMiner Loader achieves performance gains through the use of memory to buffer data. When virtual memory is exceeded, the Loader process exits, report-

ing that memory has been exceeded. Under normal operation, resources will have been properly configured to support the transactions.

However, an unusual workload that includes a significantly larger transaction can occur without the Administrator responsible for the Loader anticipating the change. In this case, it is useful to determine the source table involved so that the size of the transaction can be known.

Beginning with Release 3.3.0, the JCC LogMiner Loader provides information to help identify the source table and transaction size which exceeded the configured resources. In addition, the Loader reports the number of rows currently buffered in the virtual array and a brief dump of the record most recently read. An example is:

```
o
o
o
%dba_buffer_input: unable to write VA for modify buffer.
%dba_buffer_input: Buffer size is 439271
Commit TAD: 17-DEC-2009 08:14:41.80 Read TAD: 17-DEC-2009 08:20:00.06
tsn: 14397 LSN: 8149 action: M table: DETAILS dbkey: 85:11809:4
o
o
o
```

LogMiner Quick Sort

The Rdb release notes for 7.2.1.4 include:

> "The RMU/UNLOAD/AFTER_JOURNAL performs a sort operation to eliminate duplicate record modifications for each transaction being extracted. For smaller sort cardinalities, an internal in memory "quick sort" algorithm is used, otherwise the SORT32 algorithm is used. Previously, the limit for using the quick sort routine was a fixed value of 5000 records.
>
> This restriction of a fixed value for the threshold has been relaxed in **Oracle Rdb Release 7.2.1.4**. A new qualifier / QUICK_SORT_LIMIT=n has been provided to allow explicitly controlling the maximum number of records that will be sorted with the in memory algorithm. The default value is 5000. The minimum value is 10 and the maximum value is 100,000.

> Larger values specified for the /QUICK_SORT_LIMIT qualifier may reduce sort work file IO at the expense of additional CPU time and/or memory consumption. A too small value may result in additional disk file IO.
>
> Oracle believes that, in general, the default value should be accepted."

The LogMiner includes the option of controlling the CLM quick sort. The logical name JCC_ADD_CLM_QUICK_SORT can be used to set the LogMiner "/quick_sort" qualifier. The logical name can be defined to an integer between 10 and 100,000.

Note that both Oracle and JCC recommend using the default, in most circumstances.

Other Tuning

See also the chapter "Performance Considerations" on page 381 and, in particular, the discussion of interactions among the tuning tools in "I/O Management" on page 393.

Interpreting Complex Scenarios

This section includes explanations from JCC support that have seemed particularly pertinent. For additional material of this kind consider the FAQ and check whether there is a blog http://www.jcc.com/lml-blog that addresses your issue. Note that the blogs may be updated more frequently than this documentation.

Interpreting the Order in Which Things Happen

Applications should not be written (nor processing analyzed) with an assumption that statements, *within a transaction,* will be executed in the same order on the source and the target. This is true even in cases of full replication. If you have not made this assumption, you do not necessarily need to analyze the following points.

- The order in which AIJ records are written to disk is not necessarily the order in which updates occur. Consider the following sequence for a transaction in which the AIJ records for the change to page 20 will be written to the AIJ in step 4 and the change to page 10 will be written to the AIJ at the commit.

- Update line 1 on page 10
- Update line 1 on page 20
- Update line 2 on page 10
- Read a new buffer, forcing the buffer containing page 20 to be written back to disk
- Commit

- CLM (Oracle's Continuous LogMiner) outputs the last value for each dbkey within a transaction. The output is likely to be in dbkey order. Since Rdb will not reuse dbkeys within a transaction, a delete and insert of a row with the same primary key will result in two records output from CLM, the delete and the modify.
- When the JCC LogMiner Loader reads the data output from CLM for transactions, it puts delete records on one queue and modify records on another queue. When the Loader applies the transaction to a target, it first applies all delete records, then all modify records. (See "Keyword: Sort" on page 274.) This ordering produces the correct results in all cases.

Due to these points, not only is there no way to force the statements to be executed in the same order between source and target, there is no way to identify the original statement order.

Examining the Checkpoint Rows

The Loader stores the checkpoint data in a table in the database, when possible, and in a file at other times. The table in the database will be named "logminer_highwater". See also "Keyword: Checkpoint" on page 222.

The successful completion will always show as 'N' for a running Loader family because it is not complete while running. The complete set of statuses for the checkpoint rows are

code	meaning
N	Not complete, that is, running
R	Not complete, at checkpoint time, data was read and not yet committed.
Y	Loader shutdown and work completed

code	meaning
S	Loader thread not active and checkpoint information stale
I	Checkpoint record initialized, but not yet used

The checkpoint state completion flag 'R' is a variant of 'N'. Specifically, it indicates a slightly different starting point to the Loader than the 'N' flag. While the 'N' flag indicates that the AECP in the checkpoint record represents the last transaction that was committed to the target, the 'R' indicates that the AERCP in the checkpoint record represents the first transaction that was read from the source and that it has not yet been written to the target.

On restart from a checkpoint with the completion flag of 'N', the Loader will ignore the first transaction sent from the CLM because it has already been written to the target. On restart from a checkpoint with the completion flag of 'R', the Loader will process and replicate the first transaction sent from the CLM because it has not yet been written to the target.

Addressing Data Issues

A wide range of data transforms are available through the Loader. See "Keyword: MapResult" on page 257 and "Data Transforms" on page 497 for the flexibility introduced with the JCC LogMiner Loader Version 3.5.

This section focuses on specific data issues and solutions. Many of these focus on dates.

Null Dates

Rdb's default date of 17-NOV-1858 may be an unwelcome value. This is interpreted as a zero date, not a null value. For true null dates, it is possible to use the NULL value setting provided with the MapColumn keyword to define how the (null) date should appear in the target.

Unexpectedly Large Dates from the Source

If the Loader encounters a date greater than 9999-12-31 23:59.9999999 [1], it will stop processing. To avoid this and to ignore such dates, define the logical name JCC_LML_NULL_BAD_DATE to “1” or to “T” or “t”.

The result of encountering such a date, if the logical name is set, is that the date will be processed as if NULL. This means that the Bad Date logical name can be combined with setting ifnull to assign an artificial date to any extremely large date. This may or may not be consistent with your company policy.

Note that the row containing the bad date will still be sent to the target.

Date Filter

Date filter is a parameter for the JCC_LML_CREATE_CONTROL_FILE procedure. When this parameter is included, the procedure generates a Control File that contains a FilterMap statement. The FilterMap statement added ensures that all date columns are between 17-Nov-1858 00:00:00.00 and 31-Dec-9999 00:00:00.00, inclusive.

Either of the following will create the <database root>_DATEFILTER.INI Control File.

```
$ JCC_LML_CREATE_CONTROL_FILE <db> DateFilter
```

or

```
$ JCC_LML_CREATE_CONTROL_FILE <db> ALL
```

Note that the entire row is filtered out if the Control File is generated with this switch and the Loader encounters a date outside the “valid” range.

1. That is, anything in the year 10,000 or beyond.

Date Formatting

Note that date formatting can also be achieved with “Keyword: Date_format” on page 229.

Delta Dates Represented as NULL

OpenVMS represents delta dates as negative values. Setting the logical name JCC_LML_NULL_DELTA_DATE to ‘1’ or ‘t’ or ‘T’ will set any date column that is negative to NULL.

Delta Dates and Filters

In situations that include date columns to store delta dates, a problem remains and the filter must be hand modified as shown in the example.

Assume a table called T1. In T1, there are three columns of type DATE VMS (or other Date and Time data types) called X, Y, and Z. Assume that columns X and Y contain date and time data, but column Z contains delta date data (of the format dddd HH:MM:SS.hh).

If the DateFilter parameter of JCC_LML_CREATE_CONTROL_FILE is used on the database containing table T1, the FilterMap generated for the T1 will be:

```
FilterMap~T1~where \
  COALESCE(X,date vms'17-NOV-1858') \
    between date vms'17-NOV-1858' and date vms'31-DEC-9999' \
  and COALESCE(Y,date vms'17-NOV-1858') \
    between date vms'17-NOV-1858' and date vms'31-DEC-9999' \
  and COALESCE(Z,date vms'17-NOV-1858') \
    between date vms'17-NOV-1858' and date vms'31-DEC-9999'
```

If column Z contained regular date and time data, this filter would yield the desired result of excluding non-VMS dates. Since column Z contains delta date data, this FilterMap directive will filter every row that has a value for column Z. (Rows where Z is Null will not be filtered.) This is probably not the intended result.

If the goal is NOT to filter every row where Z has a value, the generated control file must be hand edited to exclude the test on the value of the Z column so that the filter becomes:

```
FilterMap~T1~where \
  COALESCE(X,date vms'17-NOV-1858') \
    between date vms'17-NOV-1858' and date vms'31-DEC-9999' \
```



```
and COALESCE(Y,date vms'17-NOV-1858') \
  between date vms'17-NOV-1858' and date vms'31-DEC-9999'
```

Unsigned Values for Materialized Data

The derived data that is materialized for VirtualColumn has been treated as signed data for publication to the target. Some targets support unsigned data or data that exceeds the signed values that Rdb supports. With the JCC LogMiner Loader Version 3.5, it is possible to set the Loader to attempt to write unsigned values to specific materialized columns in the target database.[1]

This feature is enabled by defining the logical name JCC_LML_STORE_UNSIGNED to “1” or to “T” or “t”.

An unsigned 8 byte integer requires a target column that can store between 0 and 18,446,744,073,709,551,615. The materialized values that may be stored as unsigned 8 byte integers are the VirtualColumns

- LOADER_SEQUENCE_NUMBER
- TSN
- ORIGINATING_DBKEY

An unsigned 4 byte integer requires a target column that can store between 0 and 4,294,967,295. The materialized values that may be stored as unsigned 4 byte integers are the VirtualColumns

- TID
- PID
- JCCLML_TXN_RECORDS

1. This does not work for targets, such as Rdb, that do not store unsigned data in columns.

JCC_DBkey_to_quad and JCC_quad_to_DBkey

Rdb's representation of the DBkey may be needed as a 64 bit integer, also referred to as BIGINT or quadword. The Loader kit includes utilities to translate a DBkey to a quad word or a quad word to DBkey format. For example:

```
$ JCC_DBkey_to_quad 8:462:0
 dbkey: 8:462:0 = quad: 2251799843962880
```

and

```
$ JCC_quad_to_DBkey 2251799843962880

 quad: 2251799843962880 = dbkey: 8:462:0
```

Loader Heartbeat and AIJ Backup

If a database becomes quiescent with no update transactions for an extended period of time, the AIJ backup process will stall behind the LogMiner.[1] This is because the LogMiner (RMU/UNLOAD/AFTER/CONTINUOUS) maintains a lock on the last location it read in an AIJ. When a journal switch occurs, it does not move the lock to a location in the next journal until the next update transaction commits. While the LogMiner maintains its lock on the journal location, it prevents an RMU/BACKUP/AFTER command from backing up the journal. Further, if the AIJ backup process has a timeout embedded in it, then the backup process can fail.

The Loader provides a way to resolve this impasse by turning on a "heartbeat." The "heartbeat" commits a transaction to the source database on a regular basis. Because the commit of the heartbeat transaction advances the logical end of file in the after image journals, each heartbeat transaction provides the LogMiner an opportunity to respond to a database journal switch request.

1. The same can be true if there are transactions, but none that are defined such as to be passed on by the LogMiner and Loader. These "inactive" sessions do not checkpoint as active sessions would. Therefore, the Loader is modified to checkpoint the current information whenever an AIJ switch is encountered on a "no work" transaction. ("No work" transactions are those that contain no rows that cause the Loader - with the definitions in the Control File - to publish anything to the Loader target.

The “heartbeat” requires a table called JCCLML$HEARTBEAT. The table has one row. The continuous Loader parent (CTL) process attaches to the source database and updates the row once per interval.

To start the “heartbeat” define (in process context) the logical name for heartbeat, JCC_CLML_HEARTBEAT_ENABLE, to 1.

By default the heartbeat interval is 300 seconds (5 minutes). To change the interval, set the logical name JCC_CLML_HEARTBEAT_INTERVAL to the number of seconds in the interval desired. Setting the interval to zero is an alternate way of disabling the heartbeat feature.

By default or if there is an exception, the “heartbeat” is turned off.

If the Loader does not find the necessary row, the heartbeat code will insert the row.

Exceptions

If there are any exceptions in processing the heartbeat, the parent process will disable the feature and generate an OPCOM message of the format

```
<LOADERNAME> failed to generate heartbeat. Disabling
feature; <exception message>
```

Using the Heartbeat

Set the two logical names

- JCC_CLML_HEARTBEAT_ENABLE to 1
- JCC_CLML_HEARTBEAT_INTERVAL to the number of seconds in the interval between heartbeats (anything except zero)

Run the Loader. The parent (CTL) process will establish the JCCLML$HEARTBEAT table and its row and will write the following to the log:

```
Heartbeat processing is enabled
Heartbeat interval is set to <interval> seconds
```

If you wish to define the storage map for the new table, shutdown the Loader to create the storage map and to move the table to the storage area. Then, re-start the Loader.

One is Enough

If you are running multiple Loaders against the same source database, note that activating the heartbeat for one of the Loaders is enough.

For the additional Loaders using the source, you need to enable the heartbeat, but set the interval to zero so that those processes do not attempt to manage the heartbeat or duplicate its activity. Do this by setting

- JCC_CLML_HEARTBEAT_ENABLE to 1
- JCC_CLML_HEARTBEAT_INTERVAL to 0

Activating heartbeat for more than one Loader per source consumes resources inappropriately.

Logging the Heartbeat

In order to understand the complete operation of your system, you may want to have heartbeat activity recorded in the log. This is not likely to be something that you want to do in production, but it can be included in production. How verbose the log becomes will depend on the heartbeat interval that you have set.

To include the heartbeat activity in the log, include in the Control File

```
logging~heartbeat
```

Example output is shown here.

For the CTL log (when a heartbeat transaction is committed)

```
11-JUN-2007 09:37:21.29: Heartbeat update committed
```

For the LML logs (when a thread receives the jcclml$heartbeat record update)

```
11-JUN-2007 09:37:45.71 202434A9 LML MAPTABLE Heartbeat AERCP: 1-
28-4-3-39104-39104 Commit TAD: 11-JUN-2007 09:37:21.28
```

Beginning with Version 3.5 of the JCC LogMiner Loader, the logging records when heartbeat begins and when it ends.

When heartbeat begins the log will include the time stamp and

```
Heartbeat update start
```

When heartbeat ends, one of two messages will occur in the log. Which one occurs depends on the version of Rdb, since earlier versions do not support returning the

TSN (transaction sequence number). Either message begins with the time stamp. The messages are:

```
Heartbeat update committed
Heartbeat update (TSN <tsn>)committed
```

In the second case, '<tsn>' will be replaced with the TSN of the heartbeat transaction.

The Heartbeat and CLM Statistics

The heartbeat mechanism was introduced to provide an answer for a conflict between Rdb backup and the LogMiner. However, under rare circumstances the heartbeat mechanism and the CLM statistics, if both running, can result in lock conflict. Therefore, when the heartbeat logical is defined, the statistics qualifier is removed from the Oracle Rdb Continuous LogMiner command.

See "Modifying CLM Statistics Output" on page 355 for more information on overriding or modifying the LogMiner statistics interval.

Setting the Heartbeat Interval

An interval of 300 to 600 seconds (five to ten minutes) is generally a good choice.

In an active database, any update transaction is sufficient to cause data to be written to the new AIJ, after an AIJ switch. Therefore, for consistently active databases, the hearbeat mechanism provides no additional functionality.

In an inactive database (or during an inactive time period), the heartbeat mechanism provides the essential transaction that causes the LogMiner to release its lock on the previous journal. To meet this need, the heartbeat interval must be configured such that the heartbeat transaction occurs between the time that the AIJ switch occurs and the time at which the RMU/BACKUP/AFTER receives a time-out exception. If your AIJ backup command includes the /WAIT=<seconds> qualifier, the heartbeat interval should be set less than the seconds for that qualifier. If you perform manual AIJ switches and then backup specific AIJ sequence numbers, then the hearbeat interval should be less than the duration between the switch and backup commands.

In addition to setting the heartbeat interval low enough to meet your other settings and practices, JCC recommends that (except in very unusual circumstances) the heartbeat interval not be set less than twenty seconds.

Side Effects of the Originating DBKey Approach

When there is no primary key that is a combination of columns in the source, the dbkey approach provides an alternative.[1] However, there are limits to the approach. These limits derive from the necessity that the dbkey does not change in the source, if it is to be used to identify rows in the target. The problematical situations are discussed here.

Export/Import

An Export-Import completely changes the dbkeys on the source data. The dbkeys on the source will not reflect information that is meaningful in the target. Export/import and dbkeys are not an appropriate combination, therefore, for replicate or for rollup (nodelete).

Truncate Table

For a truncate, the AIJ shows only the fact of a table being truncated, and nothing about the individual rows. The dbkeys on the source, after the truncate, will not reflect information that is meaningful in the target. There is nothing to replicate to the target. Records loaded with RMU/Load are journaled, but the dbkeys are likely to be entirely different. Truncate and dbkeys are not an appropriate combination, therefore, for replicate or for rollup (nodelete).

Rdb Alter Storage Map

An Rdb Alter Storage Map Reorganize command shows up in the AIJ as a series of deletes and inserts. An update to the source database that is done by doing a delete then an insert will have different Dbkeys for the deleted row and the inserted row. Rdb can eventually re-use Dbkeys, so it is possible over time to have two completely different rows that have the same originating dbkey.

If you are using originating dbkey and have configured a table for Replicate, an alter storage map reorganize and an Update using a delete - insert are not problems.

1. Alternately, an adding an identity attribute column to the source database can address the issue. See “Identifying Rows in the Target” on page 39.

If you have configured a table for Rollup or Nodelete, an alter storage map reorganize and an Update using a delete/insert are not appropriate.

Although Rdb may reuse dbkeys across transactions, it will not reuse dbkeys within a transaction. To deal with the case of a delete then an insert of the same logical record in a transaction, the Loader always processes deletes first. This works because the RMU/UNLOAD/AFTER command always provides the last value for a particular Dbkey.

Throttling the Loader

There is a great deal of attention that has been given to making the Loader fast. Should you wish to slow it down or regulate its speed, you can use the logical name JCC_LogMiner_Loader_Throttle.

Options for the throttle are:

- Realtime
- Fixed
- Percentage of realtime

Realtime

The Loader can approximate, for the target, the update rate of the source. The goal for this tool is to apply changes at a rate that is less than 1/100th of a second different than the rate between source transactions.

The realtime throttle is set with

```
$define JCC_LOGMINER_LOADER_THROTTLE realtime
```

(Case does not matter.)

Setting the throttle to realtime causes the log to include a message of the following type

```
26-SEP-2003 12:18:17.04 208004C4 LML EXMP Output delay throttle set
to REALTIME
```

The Loader will then compare the number of seconds (and/or fractions thereof) between each two successive source transactions to the amount of time since the last commit. The Loader will further adjust by an estimate of the amount of time required to apply records to the target. If the Loader detects that it is missing its throttle target, the estimate of the amount of time required to apply records to the target is adjusted and a message generated. For example,

```
26-SEP-2003 12:18:26.06 208004C4 LML EXMP Throttling too much by
0.118321 seconds. Adjusting ...
```

There are some limits:

- Realtime throttling is disabled if the data from the Rdb LogMiner does not include commit records, as is true for static mode.
- This feature is intended to provide transactions at the same rate as the source database. However, the Loader only receives information on the commit and there will be some unavoidable variance. To most closely reproduce the source rate, set the CHECKPOINT commit interval to 1.
- Any delay between two source transactions of more than 100,000 seconds (about 27.77 hours) will be ignored and a message will be logged.
- This is a throttle. It can only reduce the rate of target updates to approximate the source rate. The Loader must be tuned with sufficient threads and other performance considerations such that there is no issue with keeping up.

A thread waiting for the realtime throttle will display on the monitor screens with a state of ‘T’. The log will also show the throttling activity.

Fixed

Instead of realtime, the throttle feature can, alternately, be set to a FIXED interval pause of a specific number of seconds. For example

```
$define JCC_LOGMINER_LOADER_THROTTLE 5
```

This causes a five second delay between commits to the target.

A thread waiting for the fixed throttle will display on the monitor screens with a state of ‘t’.

Realtime Throttle Percentage

The realtime throttle percentage supports running the Loader at a faster speed than realtime. This is useful for testing scalability.

The realtime throttle is set with

```
$define JCC_LOGMINER_LOADER_THROTTLE realtime
```

The realtime throttle percentage is set with

```
$define JCC_LOGMINER_LOADER_THROTTLE_REALTIME_PERCENTAGE <value>
```

JCC_LOGMINER_LOADER_THROTTLE_REALTIME_PERCENTAGE is a logical name that can be set to a value between 0.0 and 100.00. A setting of 100 for this logical name models the original workload. A setting of zero for this logical name disables the realtime throttle *percentage*, but not the realtime throttle.

Setting JCC_LOGMINER_LOADER_THROTTLE_REALTIME_PERCENTAGE to fifty provides half as much throttle which means that the Loader sends data to the target at twice the speed of the original workload. Setting the logical name to 25 provides 25% of the throttle which sends the data at four times the original workload.

All of the restrictions cited for the realtime throttle apply.

Loader Tools for Testing

The Loader can provide powerful aids to testing applications that include the Loader. Testing may be significantly enhanced with a combination of

- Copy mode (See “Copy Mode” on page 78.)
- Loader monitoring tools (See “Monitoring an Ongoing Loader Operation” on page 313.)
- Realtime Throttle Percentage (See “Realtime Throttle Percentage” on page 482.)
- Materialized values that show time stamps (See “Keyword: VirtualColumn” on page 291.)

Using these Loader features, it is possible to see bottlenecks from the source to the target. It is possible to run the same real data in real data volumes over and over as tuning occurs in all parts of the combined application. It is also possible to run at accelerated data volumes to test scalability.

Thus, the work to create realistic test environments for down stream applications is reduced, as are the surprises from not having tested with realistic data.

Automated AIJ Backups

ABS, Automatic Backup Server, and command procedures that behave similarly set off an AIJ backup whenever a journal fills. Often ABS or other automated approaches set off backup when the database is at its busiest. Of course, performance issues can result. JCC Consultants and many other Rdb experts recommend against using ABS or similar approaches.

When using the LogMiner, it becomes even more dangerous to employ automated procedures for AIJ backup. Managing the AIJs is an important part of running a successful installation of JCC LogMiner Loader. Unplanned backups can provide inappropriate interactions between an RMU AIJ backup and RMU LogMiner. These interactions can even lose data!

There are two possible scenarios that produce difficulty.

1. When the LogMiner is begun at essentially the same time as the backup, neither has yet achieved the locks that would normally coordinate their claims on the journals. This is a rare circumstance, but has been identified as the issue for users of the Loader who contacted the support desk. The issue is addressed in the blog www.jcc.com/lml-abs-bad.
2. When the LogMiner is processing in a backed up journal when RMU/backup/after is started, the LogMiner can finish with the backup journal it was processing and be unable to move on, because the next AIJ to be processed is no longer in the live database and is not in the list of AIJs to process. Basically, the picture changed while it wasn't watching. The exception message will resemble the following.

   ```
   ..., incorrect AIJ file sequence ... when ... was expected
   ```


In the second case, the LogMiner can - with attention to AIJ backups - be restarted. In the first case, data can be lost without detection or warning.

Not using ABS or other automated backup is the best solution.

In complex interactions, care is required. When JCC and Rdb developers discovered that, in an inactive database, the LogMiner could interfere with backup, JCC added the heartbeat[1] feature to the Loader to prevent the problem. There is nothing that can be added to the Loader to prevent unplanned results from automated backup. Automated backup is uncoordinated backup. The way to prevent issues is to *manage AIJ backup by plan.*.

Note that the difficulty with ABS is exacerbated by the bug described in “Dangerous Interaction Between RMU Backup and LogMiner” on page 412.

The Loader does include a tool that enables the Administrator to determine the status of the AIJs. See “Knowing Whether the AIJ Is Processed” on page 417.

1. See “Logical Name Controls for Loader Procedures” on page 463.

Reminders

This section briefly lists the points that are the most frequent source of difficulty.

No Quiet Point Backup

The LogMiner processes committed transactions. The LogMiner and the Loader protect transactional consistency. The LogMiner cannot work on a partial transaction. Before starting the LogMiner, establish a "quiet point." A quiet point marks an epoch (point in time) at which there are no transactions started that aren't committed or rolled back. Starting at a quiet point means that the LogMiner starts at a known state.[1]

JCC recommends that you include at least one quiet point backup in each day's processing.

Missing AIJs

Rdb writes the After Image Journal. LogMiner processes the AIJ and passes the information on to the Loader. If an AIJ file that has not been processed by LogMiner is not available, neither the LogMiner nor the Loader can make up the data.

Using LogMiner and the Loader may require some changes in policies about keeping AIJs on-line.

Backing Up AIJs While Catching Up

The Continuous LogMiner and the Loader can start after a significant interruption and process through backed up AIJs until it is appropriate to switch to the live AIJ. However, during this catch up, the live AIJ should not be backed up. The catch up can appear so seamless that people can forget and trip the backup process.[2]

1. See also "Quiet Points and AIJs" on page 43, "Quiet Points" on page 97, and "The Loader Restart Context" on page 420.
2. Automated backup, such as with ABS, can cause unexpected backups and interrupt seamless processing. For that and other reasons, many Rdb experts recommend against using ABS. See also "Automated AIJ Backups" on page 483.

If a backup does happen during a catch up, the LogMiner will fail with an AIJ sequence error and the Loader session will shut down. The proper response is to restart the Loader session.

No Reliable Primary Key

Except in a case that is only inserts with no deletes and no updates (e.g., an audit), the Loader must be able to identify rows in the target. In the following circumstances, there is no combination of columns in the source table sufficient for a primary key:

- If any column in the candidate primary key may change
- If any column in the candidate primary key is null
- If every column in the table is also in the key[1]

If any of these are true, the Loader cannot identify the row for update. Using the originating_dbkey approach is an alternative in these circumstances, but some valid primary key must exist. [2]

Overall Architecture Consistency

Constraints on the target database may cause exceptions because the order in which the DML statements from a single transaction are applied to the target is not guaranteed. Further, since constraints that exist on the source have already been checked, adding the same constraints to the target is unnecessary and adds processing, as well as potential exceptions.

Triggers that have fired on the source should not also be included in the target.

These are two examples of the need to analyze the overall architecture of your source applications, source database, use of the LogMiner and Loader and your target database and dependent applications.

1. If every column in the table is also in the key, the Loader assumes that one of the columns in the key can change. Therefore, replication is not supported for such tables without use of the originating_dbkey mechanism.
2. An alternative for *adding* a column which meets the criteria is discussed in “Identity Attribute” on page 40.

Using JCC LogMiner Loader Support

The JCC support desk can provide answers more rapidly, if all the necessary information is available. It is generally best to send all logs.

In the following example, the subprocesses have fed some information to the main process log, but the detail of the exception is in the subprocess log.

The lines in this log that are informative begin with “See the LML logfile ...”

```
%dba_lml_ast: CTL active bit clear for Loader thread 0
%dba_lml_ast: See the LML logfile (jcc_tool_logs:jcc_run_lml-REG-
TESTSQL_0.log) for more detailed information about this
exception.
 8-AUG-2014 11:00:34.45: LML process exited with an unexpected sta-
tus %DBA-E-MAX_OUT_RETRIES, Maximum message output failures
received.
8-AUG-2014 11:00:34.45: See the LML logfile
(jcc_tool_logs:jcc_run_lml-REGTESTSQL_0.log) for more detailed
information about this
exception.
```


CHAPTER 17

Schema and Data Transforms

The JCC LogMiner Loader can be used to achieve serious schema changes between source and target. The Loader may also be used to transform data between source and target.

The concepts related to schema changes and data transforms are reviewed here so that the Application Architect can have a complete compendium of the options and limitations.

It should be noted that design choices are even more likely to result in surprises as you add complexity. The Loader cannot repair a confused design. JCC recommends caution in any use of the Loader to support serious schema changes or data transforms without a careful design review and an alertness to possible complexities.

Why Databases Change

You may have already encountered a need for a schema change or for a reinterpretation of the data and be looking for how much support the Loader offers. The need for a change does not necessarily result from an existing database design being "bad" when it was created. To put the topic in context, consider:

- Do you remember when applications were designed with minimal characters to describe a category and some backroom person could recite what 4F stood for? Do you need to translate that data into something that supports decision making without having to memorize obscurities?
- Do you need to merge information from two or more systems? Did the developers of the diverse systems represent the world in different ways or emphasize different aspects? Are you now trying to draw information from more than one system into a coherent whole?
- Do you have information in your data that management or a regulatory agency insists must be masked or otherwise hidden? Do you have a plan for doing so, but can't update all of your programs and reports overnight?
- Do you want to add a data warehouse to your resources?
- Do you have a healthy and reliable OLTP system, but requests from decision makers for a different view of the data?
- Do you need to combine your data with a larger pool of data resources to support Big Data analyses?

There are two sorts of fundamental changes that may be desired in the architecture that you design to meet your goals:

- Schema changes
- Data transforms

The next section discusses existing databases and changing times more thoroughly. It also specifies an example to use in discussion of schema changing options that you may want.

Data transforms are discussed in "Data Transforms" on page 497.

Schema Designs and Alternatives

We, in computing, learned to normalize our data to provide an appropriate foundation for the flexibility and efficiency that relational databases bring to transaction processing. We learned that unnormalized data limited our options and that programming snarls suggested an error in getting to third normal form.

Meanwhile, we began delivering tools that would let people analyze the data for reports or decision support. Unfortunately, the very normalization that makes OLTP work is based on understanding the relationships among data, relationships that may not be obvious to end-users. Also, reassembling normalized data may prove tedious and computationally costly for day to day analysis. The example shown in the following illustrates some of the problems.

There are additional discrepancies between OLTP systems and data warehouses or other summations. The most critical are needs for summarization and for time based processing.

Normalization Example

To address the issues and options, consider for an example a CSP (Customer Service Provider) system. CSP systems have customers, accounts, transactions against account, outstanding balances, payments, etc. For purposes of the OLTP or relational database source, each of those except the outstanding balance will be represented as a table. The outstanding balance, while theoretically derivable from the other information, is generally denormalized to a column for the account.

Now, what's the problem? Those are all simple concepts. They are simple concepts, unless the end-user expects to enter either the customer name or the account number and see all the data that is relevant. If an account can have many names attached to it and/or a customer can have many accounts, how is the data represented for the end-user. For that matter, if a customer can have many phone numbers, they may be normalized into their own table in the source database, but the end-user is likely to want a list of numbers to appear with the account or customer information.

When we design forms to present the data, we attempt to learn what the end-user will need and, then, may find it necessary to denormalize the data for the presentation. When we design an Operational Data Store (ODS) or data warehouse, the issues are more complex because there may be many end-users and/or the needs may change over time.

This section will use the CSP example to illustrate concepts and options.

Target Design and the Loader

JCC's LogMiner Loader is agile about placing data into your target. Although JCC Consultants enjoy database design and application architecture, *designing the target to meet the need is beyond the scope of this document.*

As you design your target, you want to know which things are transparently supported by the Loader. This chapter seeks to explain that.

Options for the Target

The remainder of this section includes a review of options for manipulating what gets stored where in your target. Options include:

- Standard options
  - Replication
  - Rollup
  - Audit
  - Other combinations of [no]insert/[no]update/[no]delete
- One source table to multiple targets
  - Multiple Loaders
  - One Loader
- Multiple sources of like data to the same target
  - Issues for the key
- Multiple source tables to one target table
  - Example scenario for data warehousing
  - Storage choices
  - Action choices
- Data manipulation
  - Filtering
  - Materialized values for virtual columns
  - Modifications of the data
- Combinations

- "Queue" example
- Warehousing before deleting
- Maintaining coordination among sometimes connected databases

Standard Options

Transactions written to the target can include or exclude all inserts, updates, and/or delete operations from the source. Combinations of insert or noinsert, update or noupdate, and delete or nodelete (as action choices on the table or maptable keywords) can achieve a variety of results.

Each row in each target, if it is to be referenced again, must have a unique key. In the case of the nodelete, materializing the virtual column for the action and adding it to the key is one way to provide a unique key. Other virtual columns may also prove useful. See "Materialized Values for Virtual Columns" on page 497.

Other keywords can also be used to tune your results. See "FilterMap" on page 497 and "Combinations of Techniques" on page 502.

Combinations of the standard options are used to support replication, rollup, audit or your own choice of what gets moved to the target. Each set of choices is compatible with selecting a subset of the tables or columns.

Replication. Replicate all tables or a subset of tables, all columns or a subset of columns. When all tables and all columns are replicated, there is no schema change between source and target.

Rollup. Write all tables or a subset of tables, all columns or a subset of columns without deleting the last version of a row to appear in the source.

Audit. Write all tables or a subset of tables, all columns or a subset of columns without overwriting (modifying) any row. For the data that you include, all versions of that data will be represented in the target. Generally, you will want to add one of the materialized columns that gives a timestamp or other distinguishing information.

Other Actions. You may have a need for other combinations of inserting, updating, and deleting, or not.

Sending a Row to Multiple Targets

It is possible to define more than one target table per source table within a given Loader.

For example, in the CSP (customer service provider) illustration, if all account data was represented in one table and some of the columns pertained only to a particular type of account, you might want to split the data into two or more tables in the target.

This can be accomplished with a MapFilter statement that directs the data to a MapTable based on the value of the account type. You can provide all data columns to each of the target tables or you can provide different subsets of the data to different target tables.

The following example illustrates the concept

Source Table with Combined Data								
Acct #	Type Code	X	Y	Z	m	n	o	Balance
1	STAN	xxx	xxx	xxx	xxx	xxx	xxx	xxx
2	COMM	xxx	xxx	xxx	xxx	xxx	xxx	xxx
3	COMM	xxx	xxx	xxx	xxx	xxx	xxx	xxx
4	STAN	xxx	xxx	xxx	xxx	xxx	xxx	xxx
5	STAN	xxx	xxx	xxx	xxx	xxx	xxx	xxx

Target Table 1				
Acct #	X	Y	Z	Balance
1	xxx	xxx	xxx	xxx
4	xxx	xxx	xxx	xxx
5	xxx	xxx	xxx	xxx

Target Table 2				
Acct #	m	n	o	Balance
2	xxx	xxx	xxx	xxx
3	xxx	xxx	xxx	xxx

See “Mapping Examples” on page 549 for an example of mapping to multiple target tables.

Sending to Multiple Databases

It is quite possible to use the JCC LogMiner Loader to replicate to multiple target databases. The only restriction is that it must be possible for each of the Loaders to be able to read the LogMiner output. The command procedure for each session must have the logical name JCC_ADD_CLM_SHARED_READ defined as TRUE.

```
$ Define JCC_ADD_CLM_SHARED_READ TRUE
```

Combining like Data from Multiple Sources

If you have data in different tables or even different databases that you would like in the same target table, it *MA*Y be possible to use the Loader for the combination. This section examines different circumstances and the validity or difficulty involved.

Unique Keys. In the case that we will call "Unique Keys," there is a table that exists on multiple databases that has the same format on each. In addition, the key of a given row is unique across all rows in all of the databases. These multiple sources can be written to the same target table without complexity or further consideration.

Keys that Require an Extra Column to be Unique. To complicate the previous case, suppose that the keys for the example are not unique across all databases. Instead, each of the databases have keys that are unique within their own database, but may overlap from one database to another. In this scenario, to use the Loader to write rows from the multiple databases to one table in the target, it is necessary to guarantee a unique key in the target. A simple solution is a column that names the source database that is materialized in the target. Adding this column to the source key, provides a unique key for the target. See "Keyword: VirtualColumn" on page 291 and consider the VirtualColumn JCCLML_CONSTANT.

Table Differences. Now, suppose that the tables in the different databases are not exactly the same. If there are non-key fields in some tables that are not in all tables, the Loader can still be used successfully. If, for example, there are source tables that have data for cars, trucks, and motorcycles with some attributes that are the same and some attributes that only occur in one or two of the tables, but not all of them, it is possible to write all of the data to a target table for vehicles. Each row in the target table must, of course, have a unique key.

Key Column Differences. Current Loader technology enables the definition of a target that accepts data from multiple sources. Each of these definitions (see “Keyword: MapTable” on page 262) is extended by definition of the table key (see “Keyword: MapKey” on page 256). Since each mapping separately defines the key of the target table, there may be several disjoint definitions of the key. Columns defined as part of the key for data from source A may even be null in data from source B. This is not a supported or supportable use of the Loader. The result is a table that is potentially inaccessible to other tools.

Complexity and Loader Limits

The Loader supports a great range of source to target mapping. However, the Loader avoids most enhancements that would require significant system resources on the source system. The reason for this restraint is that the source is often a mission critical production database with a requirement for high throughput.

Instead, the Loader supports passing data to JDBC, Tuxedo, or a customer-supplied API or even a different database for further processing. Presumably, this processing takes place where it will not influence the performance of the source.

Data Transforms

The JCC LogMiner Loader provides a range of tools for changing the data. Use of MapResult (“Keyword: MapResult” on page 257) significantly expands the flexibility of the JCC LogMiner Loader. Other, more specific, support for data transforms also exists. Options include:

FilterMap

For rows that would otherwise be included, it is possible to filter based on an SQL predicate on the row.[1]

Materialized Values for Virtual Columns

Useful data, including timestamps or even your own assigned value, can be added to data rows.[2]

NULL Values

MapColumn can be used to provide a value to use if the value in the source database is NULL.[3]

Dates

Different targets, cultures, or application choices can make dates seem meaningful or awkward. The Loader provides a number of ways of getting the results desired. These are discussed in “Addressing Data Issues” on page 471.

Trim

The Loader includes features to ensure Oracle database compatibility through date time specifics, trim, and the handling of VARCHAR features are discussed in the Oracle target chapter, but can be more widely applicable, particularly if the target views zero length strings as nulls. (See “Oracle Targets” on page 123.)

1. See “Keyword: FilterMap” on page 233.
2. See “Keyword: VirtualColumn” on page 291
3. See “Keyword: MapColumn” on page 252 and also “Comparing Character Data” on page 135 for a discussion of special circumstances with some targets.

Codes and Other Lookups

MapResult[1] makes is possible to use a source value to lookup the value to use in the target. The lookup table will be stored in the filter database.

Advanced Data Transforms

MapResult is capable of more complex transforms. With MapResult any SQL, Rdb built-in function, or function that results in a single value (with a single data type) can be applied to the source data value. Any functions can be included in the MapResult keyword in the Control File or can be stored in the filter database[2] and referenced in the MapResult keyword.

MapResult Examples

The Loader kit includes examples of using MapResult and several examples are included here.

Example: NULL and trim. Trimming Rdb data values to insert into some targets can lead to surprises if the trim option trims to zero characters. Oracle, for example, may interpret the trimmed result as NULL.[3]

An expression to properly address this challenge for a column named 'title' in the source and 'role' in the target, for example, might coalesce the value of the title with the null string and, if that results in the null string, write the value of a single blank character and, otherwise, write the trimmed value.

```
MapResult~<MapTable Name>~role~                    \
    (case trim(coalesce(title,''))                  \
        when '' then '<a single blank>'           \
        else trim(coalesce(tite,''))                \
    end)
```

If NULLs are a problem in the target, the SQL necessary to both trim and deal with NULLs can be encapsulated in a function.

1. See "Keyword: MapResult" on page 257
2. See "Keyword: MapResult" on page 257
3. See the Oracle target chapter and the Data Types section.

Example: Reference Table Lookups. Many applications involve simple tables that are called "reference tables" or "code tables" or "lookup tables" or "list of values (LOV) or some other term. Each of these is likely to have two or more columns, one of which is a shortcut name (or code) for something and another of which explains that something a little more clearly. For example a table with US state codes and full names might include these three rows (and others):

TABLE 1. State Codes

Code	Name
NY	New York
OH	Ohio
SC	South Carolina

A reference table might be created to have exception codes and instructions for responding:

TABLE 2. Exception Instructions

Exception	Action
1	Read the manual.
2	Call for help.
3	Reboot.

A reference table can be created to translate codes used in one application to the codes appropriate for another application. This can be useful when it is necessary to combine data sets.

TABLE 3. Translating Between Approaches

Value in Application 1	Value in Application 2
48	RED
31	BLUE
72	GOLD

Whatever your reference table, you may want to transform a value used in the source to a different column's value for writing to the target. For example, OH in the source might be written to the target as Ohio or 48 in the source might be written to the target as RED.

To start, consider the example of US state codes and the sample Personnel database that comes with Rdb. Further, assume, for the moment, that neither column in the reference table is NULL and the source does not contain a NULL for the code.

You can create your reference table of state codes and full names and store it in the FilterMap database. Then, if your target table is Employee, you will need the MapResult.

```
MapResult~Employee~state_name~                          \
(select s.state_name from filter_db.state_table s       \
where code = s.code)
```

That MapResult is color-coded to help explain. The red relates to the target, the green to the reference table in the FilterMap database, and the blue to the column in the source row.

To expand the example to handle the possibility of NULL values in the reference table, your MapResult becomes more complex.

```
MapResult~Employee~state_name~                          \
(select s.state_name from filter_db.state_table s       \
where code = (coalesce(s.code,'')))
```

You could, alternately, define this as a user-defined function which for example might be called xform_state_code, store the function in the FilterMap database, and write the MapResult as

```
MapResult~Employee~                                     \
state_name~xform_state_code(state_code)
```

The function to do this and several other functions are included as examples in the Loader kit.[1]

Example: Masking Protected Data. A company has been storing information that requires privacy protection in an Rdb database. A new scheme for becoming PCI/DSS compliant by masking the sensitive data is central to all new development. The legacy applications that write the data cannot all be converted at once. However, the company can create a target database that is PCI/DSS compliant and use the Loader to update it using the MapResult keyword to convert the sensitive information to masked values. The user defined function can convert the input data into whatever is needed to meet the new standards.

```
MapResult~MyTable~securessn~JCC$redactssn(ssn)
```

MapResult and FilterMap

For efficiency and accuracy of processing, any FilterMap and MapResult directives that pertain to a row will be executed in a single SQL statement. If the FilterMap result is to include the row, then the MapResult directive will be executed. If the FilterMap result excludes the record, the MapResult is not processed.

Length Limits

The total length of the generated SQL can use the entire 64K character limit for an Rdb statement (minus a few dozen bytes for overhead).[1] The memory required for the structure to pass in the parameters is dynamically allocated as needed, once per table.

Examples in the Loader Kit

The Loader kit includes examples of useful applications of MapResult. These are in JCC_tool_root:[examples.xform].

1. Find the examples in JCC_tool_root:[examples.xform].
1. Some early versions of Rdb 7.3 will limit this to 32K.

Best Practices

The SQL expression can, of course, be much more complex than those shown in the examples. For ease of maintenance and enhanced readability of the Control File, JCC recommends encapsulating complex expressions in stored functions and calling the necessary function with MapResult, as in

```
MapResult~Employee~department_name~jcc$code2text(department)
```

where MapResult is the keyword, Employee is the target table, department_name (which might be 20 or 30 characters of text or whatever else makes sense[1]) is the column name in the target, jcc$code2text is the name given to the function, and department (which might be a four character code or anything else defined) is the source column passed as a parameter to the function.

Storing the logic for particularly complex functions also avoids byte limits for the MapResult keyword.

The examples provided with the kit show several instances of stored functions which encapsulate the MapResult logic so that the logic can be reused without recoding.

Combinations of Techniques

The scenarios included here represent advanced Loader techniques as applied to specific situations.

Building Processing Queues

An interesting use of the technique for writing data from a source table to multiple target tables was developed to satisfy a client request for work queues.

Goal. The statement was that tables could be used for the queues, if the data was written to a given table for only a certain period (say an hour) and then the table

1. The exact data type is defined by the data type returned by the function used.

was “released” for additional processing, while another table took up the load of incoming data.

Solution. The suggested solution is to materialize a time stamp for each row and, then, filter on that time stamp to write to only one table. For example, twenty-four tables provide a separate table for each hour of the day. That provides twenty-three hours to process a table further, before it is time to start writing to it again.

The solution relies on a choice of the virtual columns that provide time stamps and the use of FilterMap. See “Keyword: VirtualColumn” on page 291 and “Keyword: FilterMap” on page 233.

Keeping Production Lean Without Losing the Data

An interesting variant on data warehousing was suggested by a client.

Goal. Keep the production database lean with only the most current data, but don’t “throw away” historic data.

Solution. Use the Loader to capture all the data in a target database. Set the Loader for NOdelete, which turns a delete into an update of the target table. Add a time stamp, the action (delete, in this case) and/or other materialized columns. Purge the production database as required, knowing that the historical data is maintained in the target.

Hiding Privileged Information

As privacy and security concerns change, it can be important to hide information.

Goal. Maintain a copy of the data that hides specific data items without losing the data or changing programs.

Solution. Write external functions that mask and unmask the data and store the functions in the filter database. The functions can be tailored to the specific need.[1]

1. This solution is adequate to mask information in the target which can be made available to people who are not privileged to see the masked information. It is not sufficient by itself to meet PCI “data at rest” requirements because the source will still be storing the privileged information. JCC has done work in encrypting to meet PCI standards, but that requires extensions to what is discussed here.

Other

Note how interesting and varied these solutions are. There will be others.

Performance Implications

As uses of the Loader are extended into more advanced mappings, there will be performance implications to some of the choices.

For example, consider the case of multiple rows in the source database - or source databases - that update a single row in the target. Since each of the source rows may have a different dbkey, the Loader locking model has no hint to manage serialization of updates to the target row. This is likely to lead to unexpected locking in the target.

The Loader is constructed to minimize the overhead as much as possible. The data transforms, for example, are not done in either the source or the target, but in the filter database intended to be used only by the Loader. For further performance advantage, the Loader does the filter operation before any data transforms to avoid work that is not required.

CHAPTER 18

Data Pump

The *Data Pump* works in conjunction with JCC's LogMiner Loader. The Loader — in combination with the Rdb LogMiner — migrates changes from the source database's after image journals (AIJs) to a target.[1] The Data Pump works with any valid Loader target to migrate existing rows in the source database to the target.

The Data Pump is valuable for the initial population of the target or for correction if the target becomes corrupted due to external changes.[2] Under normal operation, the Loader does not handle rows that are not changed, because such rows are not written to the AIJ or handled by the LogMiner. Therefore, in normal operation, the Loader cannot be used to overwrite specific rows in the target with the version that is in the source, because there is no update to the rows.

1. Find a complete discussion of targets supported in "Loader Targets" on page 34.
2. Sometimes, processes other than the Loader write to the target. These are particularly likely in initial testing and can cause corruption.

The Data Pump addition to the Loader enables you to push specific source rows to the target through a *no-change update* to the source. The no-change update causes the normal operation of the LogMiner and Loader to pass the latest version of the row to the target.

The Data Pump does its work in two phases. During the first phase, the rows to be pumped are identified. In the second phase, a no change update is made to each row to be pumped.

The writes to the source are done in a controlled fashion and can be configured to minimize lock contention in the source. The specification of which rows to handle in this way is through SQL predicates.

In addition, the Data Pump understands parent child relationships. The Data Pump supports the specification of relevant rows in a table and can also collect all the dependent rows in other tables.

The Data Pump works with the JCC LogMiner Loader Version 2.0 or later.

Industry Use of the Term "Data Pump"

Jim Gray, who was an architect for nonstop SQL, Rdb Engineering (at Digital), and, later, Microsoft, used the term "Data Pump" to talk about databases with trillions of rows. Basically, his idea was to tag along on various natural processes, like monthly billing, to solve queries. He called these natural processes "data pumps."

Many products have portions that they call "Data Pump." These differ in their design centers and in their capabilities. Oracle's product, for example, is designed to move data efficiently from one Oracle database to another and is basically a replacement for export/import. The sole purpose is massive pumping of all data in a table.

JCC's Data Pump can be used in a much more surgical fashion and has as its design center extremely low intrusion on the source database. It differs from Oracle's data pump by supporting:

- Rdb as the source
- a wide range of targets
- targets that are different from the source
- remapping of columns, tables, and data
- taking subsets based on table, column, data values, and other selection criteria
- defining and controlling hierarchies of tables
- performance enhancements

JCC's Data Pump, that is, includes the full range of advantages of the JCC Log-Miner Loader.

JCC's Data Pump is used for initial data population and for correction of the target after suspected corruption of data.

Syntax

```
$ jcc_lml_data_pump <database name>[<structure file>]
[<data driver file>][vlm]
```

Parameters

<database name>. The database parameter is specified as the root file for the source database.

[<structure file>]. For the Structure file parameter, specify the name of the file containing the table hierarchy definitions that establish the structure of SQL predicates to be used in selecting rows. Alternately, the file name is optional in the statement if the logical name JCC_LML_DATA_PUMP_STRUCTURE_FILE is used to identify the file to be used.

[<data driver file>]. For the Data Driver file parameter, specify the name of the file containing the data values to feed to the predicates of the structure file. Alter-

nately, the file name is optional in the statement if the logical name JCC_LML_-DATA_PUMP_DRIVER_FILE is used to identify the file to be used.

[vlm]. The optional text 'vlm' specifies that the temporary tables should be created with LARGE MEMORY IS ENABLED.[1] This enables the user to pump larger tables in a single driver directive. Without large memory enabled, attempts to pump a table of more than 8 to 10 million rows will run out of memory. [2]

Structure File and Table Hierarchy

A Structure File contains one or more table hierarchies. Each table specified may have children tables specified. Tables specified as children may, likewise, have children. The table and any children define the table hierarchy.

The Structure File provides the definitions of the table hierarchies in modified[3] XML format. If there is more than one table hierarchy in the Structure File, the table hierarchies are separated by a line that contains nothing but a period.

Each table specified for the Data Pump is specified with

- a table name
- a restriction
- an update column

The full table definition consists of these parts and ends with

```
</table>
```

The full table definition can, optionally, include additional syntax which modifies the operation of the Data Pump. The additional syntax is discussed in "Optional Syntax in the Structure File" on page 512.

1. Unfortunately, with Rdb versions before 7.3.1, this will fail. See "Oracle SR 3-12002172341" on page 413.
2. Another approach is to split the large tables across multiple driver file directives, using a range of values for one of the attributes. This slightly more cumbersome approach may be used with versions of Rdb that have not solve the large memory is enabled issue.
3. Some XML editors will note the modifications.

Limits

The table definitions <table ...> are combined to form structure hierarchies. The top level <table …> is considered a generation. All immediate children <table …> definitions are considered the same generation.

The limits became more generous with JCC LogMiner Loader Version 3.4.4.[1] Now there are a maximum of 46655 hierarchies. This maximum supports, for up to 36 hierarchies, up to 12 generations per hierarchy. With a larger number of hierarchies, the number of generations supported declines.[2]

"Optional Syntax in the Structure File" on page 512 describes how to specify a name for the table hierarchy. If the Structure File does not include specification of a name for the hierarchy, the Data Pump will use the table name of the parent table for the hierarchy name.

No continuation character is used with the XML style. Instead, segments are defined and their ends marked with a slash, as in <table .... </table>. Lines can be wrapped as necessary. See individual sections and "Example" on page 519 for details.

The Structure File may contain comments. Comment lines begin with an exclamation mark and are completely ignored in Data Pump processing.

Table Name

The table name is the database table name and is specified as

```
<table name='departments'>
```

The < and > are required. The <table name ...> specification can also include optional parameters. These are discussed in "Optional Syntax in the Structure File" on page 512.

1. The limits for releases prior to 3.4.4 are up to 36 children per parent table and up to 12 generations.
2. The formula is, for 37 to 1296 hierarchies, the maximum depth declines 1 per additional hierarchy; for over 1297 hierarchies, the maximum depth declines 2 per additional.

Restriction

A restriction is an SQL predicate (where clause). A question mark (“?”) may be included in the clause. The question mark is a “parameter marker” and represents a parameter that is to be read from the Data Driver file.

A restriction is

```
<restriction>
```

followed by a valid SQL predicate and ending with

```
</restriction>
```

For example,

```
<restriction>
    where department_code = ?
</restriction>
```

For the top level (parent) table, the SQL predicate can include a parameter marker, as in the example. For the top level (parent) table, the SQL predicate is optional, but the <restriction> </restriction> is not.

A child table does not use a parameter marker. A child table must have a restriction that supports the join with the parent table. The restriction for the child table will, generally, include the foreign key to the parent table, but the foreign key is not required. Only *some* set of columns sufficient to make the connection is required. The columns from the parent table are specified in the restriction for the child table as qualified names (e.g. as parent table name, followed by a period, followed by the parent column name). For example

```
<restriction>
    where department_code =
          departments.department_code
</restriction>
```

See also “Example” on page 519.

Update Column

The update column[1] is the column to update with its own value. This update is what causes the LogMiner and Loader to pass the row from the source to the target. No change in the value of the column in the source occurs.

The update column is specified at the end of a table definition with the following syntax

```
<update name='your_column_name'/>
```

For example

```
<update name='department_name'/>
```

Note that the update column statement is <update name =.../>. There is no separate </update>.

Multiple Hierarchies

Multiple hierarchies may be defined within a single structure file, but they must be separated by a line that contains only a single period.

The number of hierarchies defined is limited only by the virtual memory available to the process.

Example

The following example is a subset of the example given on page 519. It is shown here to illustrate the table hierarchy and how restrictions are indicated.

```
<table name='departments'>
  <restriction>where department_code = ?
               or department_name like ?
   </restriction>
   <update name='department_name'/>
   <table name='job_history'>
      <restriction>
```

1. See “Surprises in the Source” on page 526 for a restriction on the choice of a column for update.


```
          where department_code = departments.department_code
        </restriction>
        <update name='supervisor_id'/>
    </table>
</table>
```

The parameter markers (question marks) are filled in with values from driver directive.

Optional Syntax in the Structure File

The Data Pump offers the opportunity for additional control through the addition of optional syntax in the table definition. The full syntax for defining a table is:

```
<table name='...'[hierarchy='...']
                 [commit='...']
                 [delay='...']
                 [seconds='...']>
```

Each of these optional items are inherited from the parent table, if not specifically included.

Hierarchy

Hierarchy declares the hierarchy name. If not specified, the table name (of the parent table) is used.

The hierarchy name is used in the Data Driver directives to specify which set of requirements statements the data provided is intended to satisfy.

A hierarchy name can be specified along with any table specification, but only has meaning when specified for the top level of the hierarchy.

Commit

The commit syntax supports combining work in commit intervals to tune performance. The commit value specifies the maximum number of rows to be committed in a transaction.

Transactions do not span tables; there is always a commit when the Data Pump finishes processing all of the records for a given table. Similarly, each data directive is processed in isolation.

The default commit interval is 10. Commit intervals are inherited, if not specifically defined.

Examples of different commit intervals and different hierarchies, along with the results, are shown in “Example” on page 519.

Delay and Seconds

Delay and seconds provide a mechanism for slowing down the Data Pump to further minimize the impact on the source database. Delay specifies the number of commits before a delay is imposed. Seconds specifies the number of seconds to delay.

The defaults for delay and for seconds are zero. If either is set to zero, both are assumed set to zero and the functionality is disabled. Values are inherited, if not specifically defined.

The example that begins on page page 520 includes examples of delay and seconds and the results they cause when the Data Pump is run.

Example

The example given on page 511 could have any of the optional parameters added.

```
<table name='departments'
   hierarchy='dept'
   commit='12'
   delay='1'
   seconds='2'>
   <restriction> where department_code= ?
                 or department_name like ?
   </restriction>
   <update name='department_name'/>
   <table name='job_history'>
      <restriction>
        where department_code = departments.department_code
      </restriction>
      <update name='supervisor_id'/>
```



```
   </table>
</table>
```

Summary of Structure File Requirements

The Structure File requirements are discussed in the preceding. This section provides a summary.

1. The table name of the first table defined in a hierarchy is the hierarchy name, unless the optional hierarchy command is used to redefine it.
2. Each table defined can have “children” tables defined. Each child table in a hierarchy requires a restriction that links the rows in the child table to a row in the parent table. The columns that provide the link must provide a unique definition of a parent row, but they do not have to be the foreign key.
3. Multiple generations and multiple children per generation are supported. See “Limits” on page 509 for details.
4. Each table will be defined with <table='...'> and </table>, where ... is replaced by the table name intended and other specifications come between the beginning and the end. The table definition will have <restriction>...</restriction> between the <table='...'> and </table>, where the dots for the restriction are replaced by the SQL clause. (For the parent table at the top of the hierarchy, the SQL for the restriction is optional or may include parameter markers for the Data Driver File to satisfy.) Limits on the SQL clauses are itemized in “Limitations” on page 517.
5. The column to use as a no change update must be specified, for each table, as <update name = '...' /> where the dots are replaced by the column name to use. (Note that, in this case there is no separate </update name>.) The update name clause should be placed between the <table name='...'> and </table>.
6. There are optional other specifications that can go between <table='...'> and </table>. These are hierarchy='...', delay='...', seconds='...', and commit='...'. If commit or delay and seconds is defined for a parent table, it is inherited for a child table, unless redefined specifically for the child table.
7. Note that each name or value provided must be enclosed in quotes. Either single or double quotes may be used, but they must be consistent.
8. Blanks are not permitted on either side of an equal sign, except within the SQL statements for restrictions, where they are required. Blanks or a line feed are required between parts of the table definition, that is between the table name and any of the optional specifications, or the restriction or the update name.

9. Each hierarchy must be separated from the next by a line with a period as the only character on the line.

A graphic to emphasize these points follows.

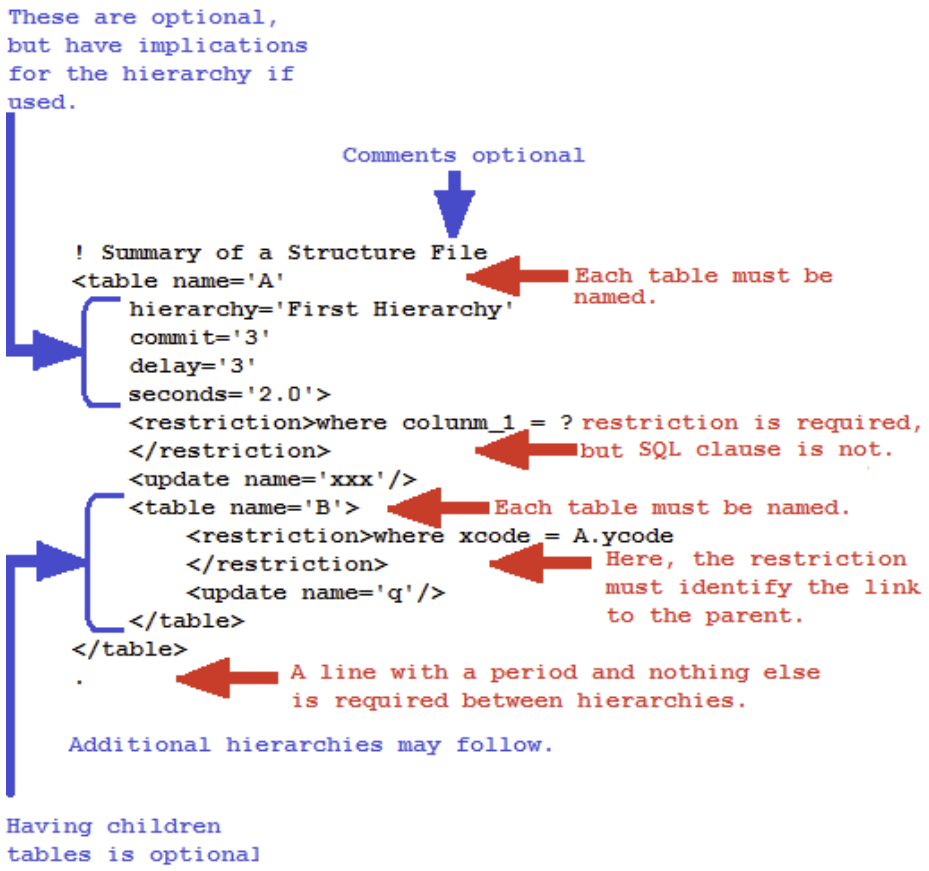


FIGURE 1. Summary of Structure File Requirements and Options[1]

1. See “Limits” on page 509 for exceptions to hierarchy size.

Driver Directive and Column Values

The Data Driver file contains one or more driver directives to be processed by the Data Pump. A driver directive provides the column values to satisfy the parameter markers (question marks) in the Structure File requirements statements.

Each record in the Data Driver file contains the name of a table hierarchy followed by a list of the column-value pairs. The value portions of these pairs are passed to Rdb as the values for the parameter markers. The column names are used as consistency checks to ensure that the driver file organization matches the structure file.

```
<hierarchy name> [<column name>=<data value>           \
                [<column name>=<data value> [ …]]]
```

Requirements

- The column-value pairs must occur in the same order as the parameter declarations in the table hierarchy.
- Text and date data values must be enclosed in quotes. (Either single or double quotes may be used, but the use must be consistent. Note the third line of the example to follow for a use of double quotes to specify a string that includes a single quote.)
- The backslash ("\") character can be used as a line continuation.
- The hierarchy name and first column-value pair must be separated by either one or more spaces or by the backslash continuation character and a line feed.
- Each column-value pair must be separated from the next by either one or more spaces or by the backslash continuation character and a line feed.
- The Data Driver file may contain comments. Comment lines begin with an exclamation mark and are completely ignored in Data Pump processing.

Example

The following example illustrates the Data Driver directives, if we assume that the structure file includes parameter declarations for department_code, department_name, and another department_name, in that order.

```
departments                                 \
    DEPARTMENT_CODE='ABCD'                  \
    DEPARTMENT_NAME="Te'*"                  \
    DEPARTMENT_NAME='THM'                   \
```

Data Pump Log

The Data Pump logs valuable information for analyzing results or finding any issues with the Structure file or Data Driver file input.

- The Structure file and the Data Driver file are both echoed in the log, as they are processed.
- When the Data Driver generates exceptions, those exceptions are included in the log.
- The number of rows selected for each table is shown.
- The way the commit interval is satisfied is shown.
- The action taken and the number of rows affected for each table is shown.
- Action for each hierarchy is reported.

Exceptions

When data driver directives generate exceptions, the exceptions are written to both the log file and the exceptions file and the procedure exits with a warning status. If there are problems with anything other than processing the data directives, the program exits with a suitable message. If all is well, it exits with dba_success.

The exceptions file is created, as needed, in jcc_tool_dp: and is named

```
JCCDP_<timestamp>.JCC_exceptions
```

A different name for the exception file can be specified by defining the logical name JCC_LML_DATA_PUMP_EXCEPTION_FILE.

Limitations

1. The SQL predicates in the hierarchy restriction clauses support most, but not all of the SQL syntax.
    - Conjunctions AND and OR, including parenthesis, are supported.
    - Operators =, <>, >, <, >=, <=, in, starting with, and like are supported.

- Mathematical symbols +, -, *, and / are supported.
- Sub-selects are not supported.
- Parameters and constant values must be on the right of the operator.

2. Interval, ANSI date-time, and segmented string datatypes are not supported for parameters, nor are NULL values. Text data values supplied as parameters may not have both single and double quotes as part of the data value.
3. The deadlock retry count is 10 and cannot yet be changed.
4. There are limits that apply to structure hierarchies. See “Limits” on page 509.

Unwanted Output

The Data Pump can be too informative. The Loader Administrator can turn off the informational message "No rows found matching the selection criteria" by setting the logical name JCC_LML_DP_TRACE_NO_ROWS to ’1’ or ’t’ or’T’.

Warning

The data pump will perform the tasks as directed. It is easily possible to direct the Data Pump to touch all rows in a very large table, and even several large tables in a hierarchy. Review what you have requested.

The pump first collects DB-keys for rows to be updated and then processes these DB-keys. In order to achieve maximal efficiency this is done in an SQL compound statement using temporary tables for the collected DB-keys. As a result, there are few messages exchanged between Rdb and the application program.

This processing model is efficient, but the user should be aware that it can be interrupted only at specific boundaries. If you run the Data Pump interactively, pressing <control-C> or <control-Y> at the terminal may not result in an immediate response to the terminal interrupt.

Once the program recognizes the terminal interrupt you may type the DCL command “Stop” to cause the Pump to actually exit.

Large Loads and Performance

Sometimes the Data Pump is used with large loads when performance is a necessity. Beginning with Version 3.5, the Loader supports enabling large memory. To use the new feature, add “vlm” to the syntax for the Data Pump. This is discussed in “[vlm]” on page 508.

Example

This example uses the Personnel database that comes as an example with Rdb. Shown here are the Structure and Data Driver files, excerpts from the log file and the exception file.[1]

Some spaces are removed in the reporting section of the log and in the exception file to improve readability in documentation format.

1. Note that the syntax produces XML-like statements. This is not, however, XML. Your XML editor may object that the example contains more than one XML document. It is, however, correct, as stated, for running the Data Pump.

Structure File

The structure file is modified XML format as is illustrated in the following.

```
<table name="departments"
        hierarchy="DeptHistory"
        commit='3'
        delay='3'
        seconds='2.0'>
        <restriction>where department_code = ?
                                        or department_name like ?
                                        or department_name starting with ?
                                        or manager_id in (?,?,?)
                                        or budget_projected > ?
        </restriction>
        <update name="department_name"/>
        <table name="job_history" commit="5">
                <restriction>where
                        department_code = departments.department_code
                </restriction>
                <update name="supervisor_id"/>
                <table name="jobs" commit="7" seconds="0">
                        <restriction>where job_code = job_history.job_code
                        </restriction>
                        <update name="job_title"/>
                </table>
                <table name="employees">
                        <restriction>where employee_id=job_history.employee_id
                        </restriction>
                        <update name="last_name"/>
                </table>
        </table>
</table>
.
<table name='employees'>
        <restriction>where employee_id = ?
        </restriction>
        <update name='last_name'/>
        <table name='job_history'>
                <restriction>where employee_id = employees.employee_id
                </restriction>
                <update name='supervisor_id'/>
                        <table name='jobs'>
                                <restriction>where job_code = job_history.job_code
                                </restriction>
                        <update name='job_title'/>
                        </table>
                <table name='departments'>
                        <restriction>where
                                department_code = job_history.department_code
                        </restriction>
                <update name='department_name'/>
                </table>
        </table>
</table>
```

Data Driver File

Notice in the following example Data Driver file

- The hierarchy names identify which hierarchy is being fulfilled and, in doing so, mean that the driver requests can be intermingled. Here, a department hierarchy is followed by two for the employee hierarchy and, then, another department one.
- The hierarchy name 'DeptHistory' is used for department, as it was defined in the department table definition and the default hierarchy name 'employees' is used for that hierarchy because no hierarchy name is included in that table definition.
- Readability is improved in this example through indentation, alignment of the continuation characters, and use of a comment line to separate the hierarchy entries. These aspects are optional, but some style should be adopted to make review easier.

```
DeptHistory                                     \
    DEPARTMENT_CODE='ABCD'                      \
    DEPARTMENT_NAME="Te'*"                      \
    DEPARTMENT_NAME='THM'                       \
    MANAGER_ID='abcde'                          \
    MANAGER_ID='129sa'                          \
    MANAGER_ID='9s9z9x'                         \
    BUDGET_PROJECTED=10000000.00ac
!
employees                                       \
    employee_id='00471'
!
employees
    employee_id='02000'
!
DeptHistory                                     \
    DEPARTMENT_CODE='ADMN'                      \
    DEPARTMENT_NAME='Tex*'                      \
    DEPARTMENT_NAME='THM'                       \
    MANAGER_ID='abcde'                          \
    MANAGER_ID='129sa'                          \
    MANAGER_ID='9s9z9x'                         \
    BUDGET_PROJECTED=10000000.00ac
```


Log

The log begins by echoing the Structure and the Data Driver files. The log also shows exception messages.

Excerpts from the actual log are given in the following. For example, some blank space is squeezed out to improve readability in this text format and some portions of the structure file echoing are replaced with ellipses. The example is also interrupted, occasionally, with explanations. To improve readability, these explanations, that are not part of the log, are shown in blue.

```
$ JCC_LML_DATA_PUMP input_db MF_PERSONNEL.JCCDP_HIERARCHY MF_PERSONNEL.JCCD-
P_DRIVER
JCC LML Data Pump D02.00.08 (built 14-AUG-2003 12:41:35.69)

This application is licensed to JCC.
Start time: Thu Aug 14 12:42:10 2003

Structure> <table name='departments'
                 hierarchy='DeptHistory'
                 commit='3'
                 delay='3'
                 seconds='2.0'>
Structure>    <restriction>where department_code = ?
Structure>                    or department_name like ?
Structure>                    or department_name starting with ?
Structure>                    or manager_id in (?,?,?)
Structure>                    or budget_projected > ?
Structure>    </restriction>
Structure>    <update name='department_name'/>

    o
    o
    o

Structure>       </table>
Structure>    </table>
Structure> </table>
Structure> .
Hierarchy 'DeptHistory' successfully declared
Structure> <table name='employees'>

    o
    o
    o

Structure> </table>
Hierarchy 'employees' successfully declared
Driver> DeptHistory                        \
Driver>    DEPARTMENT_CODE='ABCD'          \
Driver>    DEPARTMENT_NAME="Te'*"          \
Driver>    DEPARTMENT_NAME='THM'           \
Driver>    MANAGER_ID='abcde'              \
Driver>    MANAGER_ID='129sa'              \
Driver>    MANAGER_ID='9s9z9x'             \
Driver>    BUDGET_PROJECTED=10000000.00ac
DeptHistory DEPARTMENT_CODE='ABCD'  DEPARTMENT_NAME="Te'*"
DEPARTMENT_NAME='THM'  MANAGER_ID='abcde'  MANAGER_ID='129sa'  MANAGER_I
```

```
D='9s9z9x'  BUDGET_PROJECTED=10000000.00ac
                                  ^
%dba_parse_find_table: Data conversion error on data for column 'BUDGET_PRO-
JECTED'.
```

Note that the exception causes the Data Pump to abort processing of the request.

```
Driver> !
Driver> employees employee_id='00471'
employees: 1 selected row(s)
job_history: 3 selected row(s)
departments: 3 selected row(s)
jobs: 3 selected row(s)
```

The employees row is identified. The children rows in job_history are identified and the children rows of the job_history rows are identified in department and in jobs. Review the “Structure File and Table Hierarchy” on page 508 to see how these relationships are specified there.

Next, the rows are written with the no change updates. Because the Structure File definition of the employees hierarchy contains no specification for the commit, the default of 10 is used. As this is large enough to commit all rows found for each table, there are no rows remaining after each commit.

```
2003-08-14 12:42:12.71:Committed 1  updated employees record(s) - 0  remain
2003-08-14 12:42:12.76:Committed 3  updated job_history record(s)-0   remain
2003-08-14 12:42:12.86:Committed 3  updated departments record(s)-0   remain
2003-08-14 12:42:12.96:Committed 3  updated jobs record(s) - 0   remain
Updates to the 'employees' hierarchy completed successfully.
Driver> !
Driver> employees employee_id='02000'
employees: 0 selected row(s)
employees employee_id='02000'
^
No rows found matching the selection criteria.
```

Since no row was found for this employee_id, there is no further processing for that Data Directive. The exception shows in both the log and the exception file. Review of these will show the exception and it may be that the value entered is a typo.

```
Driver> !
Driver> DeptHistory DEPARTMENT_CODE='ADMN' \
Driver>     DEPARTMENT_NAME='Tex*' \
Driver>     DEPARTMENT_NAME='THM' \
Driver>     MANAGER_ID="a'cde" \
Driver>     MANAGER_ID='l"9sa' \
Driver>     MANAGER_ID='9s9z9x' \
Driver>     BUDGET_PROJECTED=10000000.00 \
Driver>
departments: 1 selected row(s)
```



```
job_history: 15 selected row(s)
employees: 15 selected row(s)
jobs: 15 selected row(s)
```

The Data Pump found the department specified, the job_history rows for that department, and the employees and jobs rows for the job_history rows. For each, it reports how many rows were found.

The commit for departments is defined in the Structure File as three. Since we have only one, it is written with the first commit. Next, the fifteen job_history rows are written. They are written in three commits because the commit for job_history is defined (in the Structure File) as five.

The children of job_history are written, next, one table at a time. Employees inherits the commit from job_history because none is specifically included in the Employees definition. However, the Structure File definition of jobs includes a commit of seven.

```
2003-08-14 12:42:13.08:Committed 1   updated departments record(s) - 0   remain
2003-08-14 12:42:13.18:Committed 5   updated job_history record(s) -10   remain
2003-08-14 12:42:13.23:Committed 5   updated job_history record(s) - 5   remain
2003-08-14 12:42:13.29:Committed 5   updated job_history record(s) - 0   remain
Delay requested... 15 job_history rows (of 15) updated.  Waiting 2.00 sec-
onds...
2003-08-14 12:42:15.37:Committed 5   updated employees record(s) - 10   remain
2003-08-14 12:42:15.41:Committed 5   updated employees record(s) - 5   remain
2003-08-14 12:42:15.46:Committed 5   updated employees record(s) - 0   remain
Delay requested... 15 employees rows (of 15) updated.  Waiting 2.00 sec-
onds...
2003-08-14 12:42:17.53:Committed 7   updated jobs record(s) - 8   remain
2003-08-14 12:42:17.57:Committed 7   updated jobs record(s) - 1   remain
2003-08-14 12:42:17.60:Committed 1   updated jobs record(s) - 0   remain
Updates to the 'DeptHistory' hierarchy completed successfully.
```

Notice the interruptions to processing that begin with the word “Delay.” The definition in the Structure File for departments includes delay 3 and seconds 2. Since there is only one departments row found, the delay does not get implemented there. However, since the job_history definition does not include a delay or seconds specification, these are inherited from departments. Therefore after three commits for job_history, the Data Pump waits two seconds. Employees, likewise, inherits the values from job_history, but the Structure File definition for jobs includes the specification seconds='0'. Since a value of zero for either delay or seconds turns the delay off, there is no processing delay involved with the jobs records. See “Delay and Seconds” on page 513 for more on this topic.

```
Exception records have been written to JCC_TOOL_DP:JCCD-
P_20030814124210.JCCDP_EXCEPTIONS
```

```
%dba_process_dp_input_file: %DBA-W-DRIVER_EXCEPTIO, Completed with excep-
tions in the driver file.  See the exceptions file.

%DBA-W-DRIVER_EXCEPTIO, Completed with exceptions in the driver file.  See
the exceptions file.
```

The log ends with a reminder that there were exceptions.

Exception File

```
$ dir JCC_TOOL_DP:JCCDP_20030627145801.JCCDP_EXCEPTIONS

Directory JCC_ROOT:[TOM.JCC.DEVELOPMENT.DBA]

JCCDP_20030627145801.JCCDP_EXCEPTIONS;1    1/3    27-JUN-2003 14:58:02.28

Total of 1 file, 1/3 blocks.
$ type JCC_TOOL_DP:JCCDP_20030627145801.JCCDP_EXCEPTIONS
departments DEPARTMENT_CODE='ABCD'  DEPARTMENT_NAME="Te'*"  DEPART-
MENT_NAME='THM'  MANAGER_ID='abcde'  MANAGER_ID='129sa'  MAN-
AGER_ID='9s9z9x'  BUDGET_PROJECTED=10000000.00ac


                                          ^
%dba_parse_find_table: Data conversion error on data for column
'BUDGET_PROJECTED'.
employees employee_id='02000'
^
No rows found matching the selection criteria.
```

Notes for the Administrator

The Data Pump makes extensive use of temporary tables. To pump large volumes of data efficiently, examine the Rdb documentation for hints regarding tuning temporary tables.[1]

Also, note the Data Pump's "sorcerer's apprentice" possibilities and the "Warning" on page 518.

1. Searching for notes on the logical name RDMS$TTB_HASH_SIZE may help.

Included here are additional considerations.

Performance Enhancement

The Data Pump uses the restrictions in the structure file to specify the rows on which to operate. This could lead to the same row being included multiple times for update. Subsequently, the row could be updated multiple times, often in different transactions, providing unnecessary work and the possibility of synchronization conflicts between Loader threads.

To avoid that possible source of lesser performance and of lock conflicts,[1] duplicate versions of the same row are eliminated in the capture phase. This consideration improves throughput for both the Data Pump capture and, subsequently, the Loader.

Surprises in the Source

A no change update is designed to write the row to the AIJ for reading by LogMiner ... without changing the source. It is possible, however, to write a trigger that causes any update to a column in the source to take some action. It is important to consider whether the source has any such triggers before setting the Data Pump loose.

Performance and the Initial Load

Under certain circumstances, it may be desirable to speed up the activity as much as possible. This section addresses how to get the fastest results when originally loading a large target from a large source. For smaller challenges, this level of detail may be much more than is justified.

Note that situations that justify using these techniques are unique enough that the situation also justifies a bit of experimentation to choose which of these techniques applies. Read all of the sections before beginning experimentation.

For recovery from unwarranted changes to the target, some of these tips may apply, but they are described here, specifically, as they relate to the initial population of a large target.

For best results, cause the Data Pump to perform its work in (relatively) small, memory-efficient chunks while, at the same time, ensure maximum throughput

1. This performance enhancement was added with Loader Version 3.1.

through parallelization. For this purpose, it is best to treat tables with large cardinalities differently than those with small cardinality. The ultimate goal for the parallelization is to have all of the worker processes working continuously and for each to have sufficient work that they finish at approximately the same time. Note that, in following this plan, the parallelization will, eventually, be limited by the ability of writing to the AIJ file.

Some general rules:

1. Each large table should be handled by a separate instance of the Data Pump. If necessary, the work on a single table can be split among multiple Data Pump instances. However, this leads to inefficiency in the LogMiner to Loader step and should be avoided, if possible.[1]
2. Small tables can be combined into one or a few Data Pump instances. Desirable limits to the aggregate cardinalities will depend on the large table workloads.
3. Configuration Files for the Loader should be tailored to a Data Pump instance.

Rules for the Structure files to be used with the Data Pump:

1. The commit interval should not exceed 5000, as that is the default in-memory sort limit for the LogMiner. The chosen value should be the same for all tables.
2. Small tables can be processed with a single file with each table defined with no restriction clause.
3. Large tables should be segmented into about one million row chunks[2] and processed sequentially. Segmenting the rows will cause more efficient use of virtual memory. Generally, the primary key provides the most effective way to segment the table.

Each instance of the Data Pump should be matched by an instance ot the Loader. Rules for the Loader Control Files and the LogMiner Options File:

1. The reason that splitting the table may be advisable is that the Data Pump collects all of the DBkeys for the rows to be pumped in an Rdb Temporary Table. Rdb Temporary Tables are created in process address space and that is limited to 1.2GB of memory. Current research in support of the Loader is examining ways to avoid this limit.
2. The delineation point between large and small tables increases as computers get faster and memory sources are more readily available. Currently (at release of Version 3.5), one million is a good number to use. Similarly, the size of the large chunks is arbitrary, but is currently assumed to be one million.

1. Replication options for all tables should be INSERT, NOUPDATE, NODELETE, as this will reduce lock overhead in the target.
2. Lock mode should be set to UNCONSTRAINED or AUTOMATIC)
3. Commit interval should be set modestly at 1 to 5.
4. Parallel threads should also be set at 2 to 5 and possibly should be the same as the commit interval.
5. The same TABLE and MAPTABLE Control Files can be used for all Loader instances, but each will need a unique LoaderName.
6. The LogMiner options file should include only the table(s) to be processed by the Loader instance.

Rules for the target:

1. Minimize the data structures in the target - indices, triggers, and such - to ensure that the target performs only the minimum work required. Using the insert only configuration, the Loader will never perform uniqueness queries and so the target may need no indices at all during the load.
2. These data structures can be added after the load, but do consult the recommendations pertaining to your specific target in the relevant chapter.

If the source database is not being actively updated, the steps then are:

1. Start the Loader instances.
2. Start the Data Pump instances.

If the source database is being actively updated, the recommended steps are:

1. Take a quiet point AIJ backup on the source.
2. Move any prior AIJ backups to where they will not confuse operations.
3. Copy the source.
4. Start the Loader instances *on the copy*.
5. Start the Data Pump instances *on the copy*.
6. When the target has the initial load, start the Loader that will be used going forward *on the original source*. If any AIJ backups were taken after the quiet point AIJ backup of step 1, start with the new AIJ backups. (Any data changes prior to the backup in step 1 will already be reflected in the load.) If there are no additional AIJ backups, start LIVE.

CHAPTER 19

Example: Reorganizing an Rdb Database

Rdb does not permit the DBA to reorganize a database while that database is still active in production. Using the JCC LogMiner Loader, it is possible to reorganize a database with only brief down-times.

This chapter will outline a methodology to accomplish "near on-line" reorganization. This approach has been used successfully to reorganize large databases with as little as 15 minutes of down time.

With improvements in the Loader tools, the steps are much less cumbersome.

The Basic Concept

After starting the LogMiner and doing a quiet point backup, make a copy of the database that needs to be reorganized. Reorganize the copy and test until satisfied. Use the Loader to apply to the copy the data changes that have been made in the original while the reorganization was done. Shutdown the original, permit the

Loader to finish any updates from changes made between the decision to shut down and the shutdown, and bring up the reorganized database.

Resources

The methodology uses two separate contexts for accomplishing the reorganization. In the discussion below, these are described as the production and the development environments.[1] These two environments may coexist on the same systems, provided one is careful to segregate the multiple databases involved. However, it should be recognized that a reorganization is a rather resource intensive activity and that it could impact production activity substantially, if production and development use the same resources.

The methodology, reported here, requires significant storage space in the development environment where both an old and a new copy of the database will coexist. Depending on how the last two phases are executed, similar space requirements may also exist in the production environment.

Establish an Epoch

Enable Continuous LogMiner on the original database, the source.[2] Create a well-defined epoch for starting by taking a quiet point backup and moving all AIJ backups prior to this point to a directory where they will not be confused with new AIJs.

At this point, make your copy (the target).

Note that you will not yet begin running the Loader.

1. Note that these two environments represent the source (the original) and the target (the copy to be reorganized). Source and target are the terms used extensively elsewhere in this document.
2. In the original use of the LogMiner and Loader for reorganizing a database, static LogMiner was all that was available. JCC, now, recommends enabling Continuous LogMiner and using the Loader in Copy Mode.

Create the Copy

Create and populate the copy in any of the ways suggested for creating and populating any other Rdb target.

With improvements in the Loader, users of the JCC LogMiner Loader are finding the Data Pump an attractive alternative for loading initial data into the target. The Data Pump is included in the Loader license.

To make the most efficient use of the Data Pump study “Data Pump” on page 505.

Reorganize

Make the changes that are needed. Test in whatever fashion is available.

This phase can take as long as is needed because the LogMiner can furnish the data changes that are occurring in production (the source) and the Loader can apply them to the copy that you are reorganizing.

Catch-up with the Data Changes

As the work on the copy has occurred, normal updates to the production database have continued. During the catch-up phase these updates will be applied to the reorganized database.

With the production database still in use, use the Continuous LogMiner Loader to apply to the reorganized copy, all data changes that have occurred since the copy was made. In Continuous mode, the Loader can begin in the backup AIJs and then switch (automatically) to the live database when the backups have been processed. In a large, busy system, this may require some time; but it does not interrupt the continued functioning of the source database and the processes that use it.

Switch

When the reorganized database has caught up with the data changes, shutdown the original (production or source) database. Permit the Loader to finish any updates from changes made between the decision to shutdown and the actual shutdown.

Bring up the reorganized database as the new production database.

Other Changes

The JCC LogMiner Loader can also be used to make schema changes. JCC recommends that the physical database reorganization occur separately from any logical database reorganization (schema change).

There are at least two reasons to separate the two activities:

1. to limit the confusion from trying to design and accomplish too much simultaneously.
2. to avoid surprises from application programming that is dependent on the old schema.

However, a similar approach to that cited for a physical reorganization can be used for a logical reorganization (schema change).

CHAPTER 20

Example: Oracle Slave Database

Whatever the target, what "slaving the database" implies is a straightforward replication. The main outline of the issues is:

- The tablespaces in the database must be defined
- Users must be given access to the database
- Memory and other performance structures must be added
- Tables must be defined
- Indexes must be defined

The toolkit supplied with the JCC LogMiner Loader product includes procedures that can assist with some of these tasks. Specifically, scripts are provided which allow the database administrator to create table definition scripts that match existing Rdb tables. These scripts also will generate index definition scripts for the Oracle database that reproduce the index definitions that are present in the Rdb database.

These procedures will generate appropriate Oracle SQL statements that allow you to copy data directly from Rdb to Oracle without having to go through intermediate unload files. Such unload files can be costly in terms of I/O time and disk space and can also lead to incorrect results if carriage control characters are allowed in Rdb text strings, as many comment type columns do.

It is certain that these generated scripts will not yield the optimum structures in the target Oracle database. For instance, all tables are placed in the USERS table space and all indexes are placed in the INDEX table space. Better performance may be achieved by moving these tables and indexes to other table spaces.

In any event, these scripts can serve as templates for final scripts that can be executed in the Oracle database.

All of the preparation work including the initial download of starting data is assumed to occur from a restored copy of the production database.

Generating the Initial Oracle Scripts

The Rdb database engine supports column and table names that are longer than those supported by Rdb. Accordingly Rdb tables and column names must be mapped into Oracle values. A set of procedures has been supplied that will generate an initial set of mappings.

Create and Populate the JCC Names Table

Create the table with

```
jcc_tool_sql:jcc_generate_oracle_names_tbl.sql
```

This should be executed while attached to a restored backup of the production database being slaved to Oracle. You should edit this procedure before running it to ensure that the table and its index are placed in appropriate storage areas.

The procedure

```
jcc_tool_sql:jcc_generate_oracle_names.sql
```

should then be executed to populate rows in the table just created.

Add Required SQL Procedures to The Database

Add the necessary functions to the database to support the remaining procedures. These procedures are stored in

```
jcc_tool_sql:vms_functions.sql
```

This procedure should be executed while attached to the restored copy of the production database.

Create Scripts to Move Data to Oracle

The two alternate procedures are provided to generate the scripts to move data between an Rdb database and an Oracle database. These are

```
JCC_TOOL_COM:JCC_LINK_DATA_TO_ORACLE.COM

JCC_TOOL_COM:JCC_MOVE_DATA_TO_ORACLE.COM
```

Each of these procedures accepts three parameters

1. The name of the restored database
2. The name of the table to be moved to Oracle
3. A unique tag (up to 8 characters long) that will be used to define output file names for the files containing the necessary scripts

Each of these procedures will generate several scripts necessary for the move. You should edit these scripts to provide appropriate placement etc.

If the table being slaved has a proper primary key, the descriptions below illustrating dbkey based slaving may be omitted. They do, however, contain useful examples as to how data is moved from Rdb to Oracle. JCC has found that directly moving the data via database links is the most effective way to accomplish this task.

Add Dbkey Columns and Indexes

If the table being slaved has a proper key, this section may be omitted.

If the originating dbkey approach is to be used to slave the Oracle tables, these columns must be manually added to the Oracle table definition script, as shown in the example:

```
create table <table name>
        (WORK_ORDER_NUMBER             NUMBER(10 ,0  )
        ,OUTLET_LOCATION               CHAR(3        )
        ,ASSIGN_PKG_CODE               CHAR(3        )
        ,ASSIGN_DATE                   DATE
        ,SERVICE_CODE                  CHAR(3        )
        ,originating_dbkey             number

  )
       pctfree 10
       tablespace users

  -- There are 6344429          rows in the table

  -- and each row requires an estimated 35 bytes

  -- allowing 209 rows per block with pctfree of 0.10

  -- Requiring 30357 blocks of 8192 bytes each
       storage (initial       237                 M
                next          59                  M
                maxextents     unlimited
                pctincrease    25)
      ;
      commit work;
```

The following sort of index should also be added to this table. Of course the storage statistics should be adjusted to the size of the table

```
create index <table name>_dbk
         on <table name>
              (originating_dbkey)
```

```
        storage (initial      <size>                M
                 next         <size>                M
                  maxextents    unlimited
                  pctincrease   25)

    compute statistics

    ;
```

Create View in Rdb to Materialize the Dbkey Values

If the table being slaved has a proper primary key, this section may be omitted.

If the dbkey approach is being used, a view is necessary in Rdb to convert dbkeys to numbers. (Oracle does not support the binary values Rdb will place in string formatted dbkeys and therefore these must be converted to NUMBER datatypes as shown above. The following view is appropriate.

```
create view <table name>_K
          (WORK_ORDER_NUMBER,
           OUTLET_LOCATION,
           ASSIGN_PKG_CODE,
           ASSIGN_DATE,
           SERVICE_DATE,
           originating_dbkey) as
          select
              WORK_ORDER_NUMBER,
              OUTLET_LOCATION,
              ASSIGN_PKG_CODE,
              ASSIGN_DATE,
              SERVICE_DATE,
              cvt_dbkey(DBKEY)
          from <table name> ;
```


Set Up Database Link Between Oracle and Rdb

The Rdb database must be prepared to support SQL*net. The reader is referred to the Rdb documentation for the proper procedures to accomplish this. Example COM and SQL files suitable for inclusion in the SQL Services definitions are included as part of this kit. They are `low_overhead_oci.com` and `low_overhead_oci.sql`. There is a sample template SQL Services procedure `low_overhead_oci.sqs` that can serve as a starting point for those definitions.

The Oracle database must have a database link back to the Rdb database. The SQL definition for that is outlined in the appropriate Oracle documentation and would look something like:

```
create database link <sample db>.world
      connect to <username> identified by <password>
      using '<sample db>.world';
```

The TNSNAMES.ORA file contains an entry something like:

```
<sample db>.WORLD =
      (DESCRIPTION =
      (ADDRESS = (COMMUNITY = tcp.world)
                 (PROTOCOL = TCP)
                 (Host = <target machine IP defini-
      tion>)
                 (Port = 1527))
      (CONNECT_DATA = (SERVICE = ldr_init_db)
      )
      )
```

Transferring Data from Rdb to Oracle

If the table being slaved has a proper primary key, this section may be omitted.

The generated SQL statements would transfer the data from Rdb to Oracle and look something like:

```
insert into <my table>
          (CAMPAIGN_CODE
          ,ABBREV
          ,DESCR
          ,CAMPAIGN_TYPE
          ,START_DATE
          ,STOP_DATE
          ,NUM_OF_CUSTOMERS_TO_REACH
          ,originating_dbkey
        )
       select
            CAMPAIGN_CODE
          ,ABBREV
          ,DESCR
          ,CAMPAIGN_TYPE
          ,START_DATE
          ,STOP_DATE
          ,NUM_OF_CUSTOMERS_TO_REACH
          ,originating_dbkey
       from <my table>_k@sample_db_db.world;
```

Adding Remaining Indexes and Catching Up

Once all initial data has been loaded into the Oracle database, one must then catch up the Oracle database to the source production database by using a static version of the LogMiner and Loader. One would play all transactions captured in the AIJ since the database was initially backed up into the target Oracle database.

Once the target Oracle database is caught up, the remaining indexes can be added and the Oracle database will be ready for use.

At that time, a continuous version of the LogMiner and Loader can be instantiated to maintain the Oracle database in synchronization. Alternately, batch jobs can load daily transactions into the Oracle database for appropriate use.

CHAPTER 21

Additional Architectures

The architectures that employ the JCC LogMiner Loader are quite varied. Throughout the documentation, architectures are described. A sampling of important examples is collected in this chapter.

Create an Archive

To create an archive, set the Loader Control File to rollup everything that happens to the data. That is, use rollup (or insert, update, nodelete) as the action parameter(s) of the table keyword for each table to be archived. See “Keyword: Table” on page 277.

You are now free to delete rows from the source database without losing history data. If you wish, you may add columns that identify the transaction date and time (or some other key timestamp), together with the action column (‘M’ or ‘D’) to note when the source row was deleted. See “Keyword: VirtualColumn” on page 291.

Create an Audit Trail

To create an audit trail, you need to be able to tell who changed what, when. With Oracle, this is somewhat automated; but, with Rdb, you must take some steps to make it happen.

One approach is to have a second table for each table that you want to audit and to have triggers that write a row to the audit table for each change to the primary table. The trigger will need to write old values and new values and the user and the date/time stamp. The advantage of having the audit data right in the database that is changing is that it is easy to present if someone wants to drill into why a value changed. However, the audit data may be used seldom and may add to the considerations for database administration. Besides, writing and maintaining the triggers is tedious.

To use the LogMiner and the LogMiner Loader to create an audit trail, define the Control File such that each table to be audited is defined with replicate as the action parameter *and* materialize the virtual column transaction_commit_time and username. (Transaction_start_time could be used, alternately.) See “Keyword: VirtualColumn” on page 291.

If you wish to also preserve deletes and who did them and when, set each table for which you require this information to rollup and materialize the action as well as the timestamp and username. See “Create an Archive” on page 541.

Rolling Up Regional Databases

For comments on rolling up regional databases or other composites of separate databases, see “Schema and Data Transforms” on page 489.

Providing a Separate Database

There are a number of reasons that you may want to provide an alternate copy of the source database information (or some portion of it).

- Off-load some of the query work.

- Distribute (geographically) the query work.
- Provide data for query without risking updates to the source data.
- Provide data to tools and applications that are not designed for Rdb.

Any of these can suggest a one-way path for data updates. Exposing data to the Internet and protecting the validity of production data can be at odds. Providing access for some query tools or for heavy querying can interfere with performance of the transaction-processing database. Imperfect networks and distance can suggest getting the data out into the field. A growing body of tools for use with Oracle databases or with Java or with other computing resources can necessitate fundamental changes in data to achieve apparently seamless integration.

With the Loader, an architecture is possible that provides a "safe" environment for the production database, but exposes the data for use with other applications. This architecture leaves the production database (the source) completely alone, except for turning on the LogMiner. The target database is available for querying *without impact on the production database.*

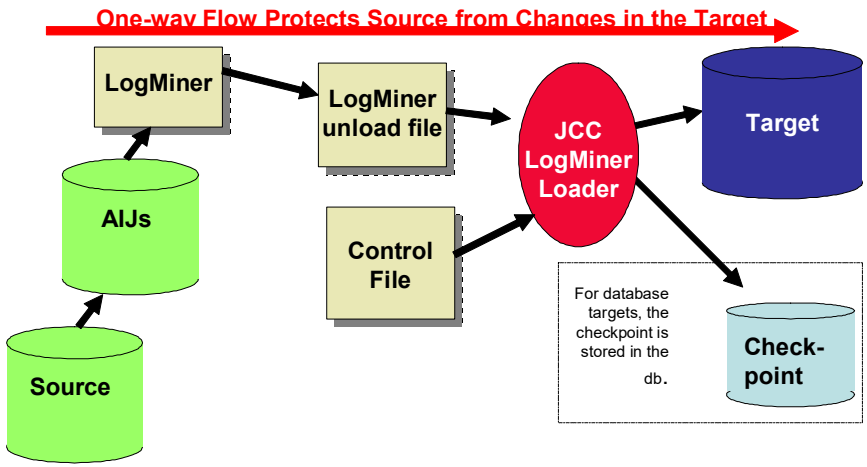


FIGURE 1. Separating the Production Database from Query Database(s)

It is also possible to place a subset of the tables or a subset of a table's columns into the target. It is possible to make the target a union of several other databases and it is possible to write more than one target for which the targets may be the same or not.

Of course, additional levels of indirection are possible. The query database created by the LogMiner Loader might be the source for additional data stores or the Loader might maintain multiple target databases. Your choice among these options depends on your environment and what you are trying to achieve.

In some circumstances, architectures suggest that the diagram should illustrate a two-way path. Co-existence during conversion is a frequent example of this need. Updates made to an SQL Server database during an Internet session is another example. The later example has been resolved with special code and JCC Consulting, but extreme care must be taken that a row is not updated based on stale data that has already been superceded on the source. For conversion projects or projects that call for continued co-existence and overlap of Rdb and Oracle data sources, JCC (with the assistance of Oracle Engineering) has explored the technology of providing loading like that done by the Rdb LogMiner and the JCC LogMiner Loader that uses Oracle as a source. To date, funding sources have not justified the project.

VAXes and Becoming More Current

A company approached JCC for assistance. They were runnning their applications on VAXes with software versions that were no longer current. The Loader does not run on a VAX.

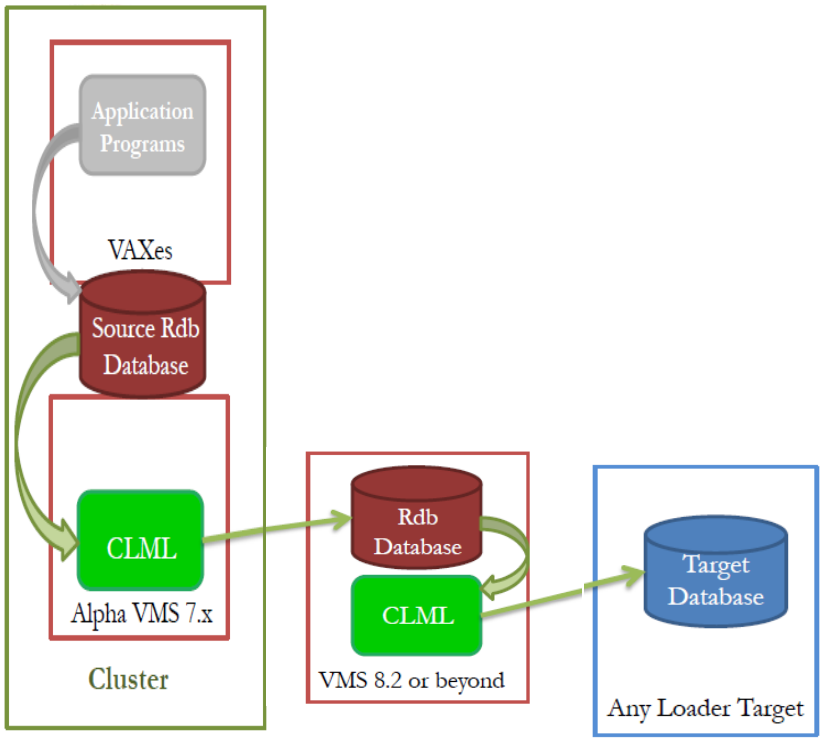
They also wanted to publish data to the JDBC target and java is not available on the older OpenVMS computers that the customer ran.

JCC designed an architecture to Load the data changes into the desired target through the introduction of some intermediate steps.

- Cluster the VAX with an Alpha server so the Alpha can open the database used by the VAX.
- Use LML on the Alpha to publish the data.
- Make the intermediate target Rdb on an Alpha running OpenVMS 8.2 or later (to get support for Java)

- Use LML again to publish to whatever data store is required using the Loader's JDBC target.

FIGURE 2. Architecture for Moving Off a VAX



Testing and Tuning

The Loader makes an excellent tool for replaying a production workload. Of course, only updates are represented in that workload. Queries are not. Even with that, one has the opportunity to review the database response to applying such changes. Errors in SPAM parameter settings are especially evident when using the Loader.

See “Modes of Operation” on page 73 and “Interpreting Complex Scenarios” on page 469 for a discussion of available tools. Notice that, to take advantage of the tools, you will also need the Loader Monitoring tools. See “Monitoring an Ongoing Loader Operation” on page 313, in particular “Detail Report Example” on page 326.

These tools can be a benefit for

- Testing tuning options for your production database.
- Regression testing your application. Note that, in this case, you would want to suppress timestamp and date columns in the output so that you can run differences for results both before and after application change. Any differences should be explainable by predicted changes to application behavior.

Other Architectures

There are, certainly, other architectures currently using the Loader and yet more that are likely in the future. You may contact the support desk account that has been provided to you with your license or, if you do not yet have a license, contact jcc_lmloader@jcc.com to discuss your architectural challenges.

CHAPTER 22

Extended Examples and Tools

This chapter provides examples that may run too long to have merit in other parts of the document, but have proven valuable in answering support desk questions.

Using MapTable to Isolate Metadata Changes

For an existing Loader application, if you add columns to a table in the source, but do not need to replicate them to the target, little change is required ... provided that you have followed the recommendations for creating Control Files. See “Building the Control File” on page 217 and “Referencing Other Control Files” on page 219.

If you have followed the recommendations, you have a Control File that is partitioned. The source metadata file can be regenerated with procedures in the Loader kit whenever there is change. The file that describes mapping to the target does not need to change if columns are added to the source that are not mapped to anything in the target.

To use the MapTable options and create the two separate portions of the Control File, your interaction will follow this pattern:

```
atlas > jcc_lml_create_control_file mf_personnel table
1 !
319 substitutions
```



```
1 substitution
JCC_ROOT:[KEITH.SQL_CLASS.MF_V71]MF_PERSONNEL_TABLE.INI;17 74 lines
%DELETE-I-FILDEL, JCC_ROOT:[KEITH.SQL_CLASS.MF_V71]MF_PERSONNEL_TABLE.INI;16 deleted (515 blocks)
atlas > jcc_lml_create_control_file mf_personnel maptable
1 !
131 substitutions
JCC_ROOT:[KEITH.SQL_CLASS.MF_V71]MF_PERSONNEL_MAPTABLE.INI;14 133 lines
%DELETE-I-FILDEL, JCC_ROOT:[KEITH.SQL_CLASS.MF_V71]MF_PERSONNEL_MAPTABLE.INI;13 deleted (515 blocks)
atlas > dir /since *.ini;

Directory JCC_ROOT:[KEITH.SQL_CLASS.MF_V71]

MF_PERSONNEL_MAPTABLE.INI;14                  8/10 9-AUG-2007 09:13:31.90
MF_PERSONNEL_TABLE.INI;17                     6/10 9-AUG-2007 09:13:24.04

Total of 2 files, 14/20 blocks.
```

In this example, MF_PERSONNEL_TABLE.INI contains the following definition for the EMPLOYEES table:

```
Table~EMPLOYEES~1~~NoMapTable
Column~EMPLOYEES~EMPLOYEE_ID~1~5~0~14~0
Column~EMPLOYEES~LAST_NAME~2~14~0~14~0
Column~EMPLOYEES~FIRST_NAME~3~10~0~14~0
Column~EMPLOYEES~MIDDLE_INITIAL~4~1~0~14~0
Column~EMPLOYEES~ADDRESS_DATA_1~5~25~0~14~0
Column~EMPLOYEES~ADDRESS_DATA_2~6~20~0~14~0
Column~EMPLOYEES~CITY~7~20~0~14~0
Column~EMPLOYEES~STATE~8~2~0~14~0
Column~EMPLOYEES~POSTAL_CODE~9~5~0~14~0
Column~EMPLOYEES~SEX~10~1~0~14~0
Column~EMPLOYEES~BIRTHDAY~11~8~0~35~0
Column~EMPLOYEES~STATUS_CODE~12~1~0~14~0
```

MF_PERSONNEL_MAPTABLE.INI contains the following definition for the EMPLOYEES table:

```
!
MapTable~EMPLOYEES~EMPLOYEES~Replicate
!
!Using_defined_primary_key_constraint.
!
MapColumn~EMPLOYEES~EMPLOYEE_ID
MapColumn~EMPLOYEES~LAST_NAME
MapColumn~EMPLOYEES~FIRST_NAME
MapColumn~EMPLOYEES~MIDDLE_INITIAL
MapColumn~EMPLOYEES~ADDRESS_DATA_1
MapColumn~EMPLOYEES~ADDRESS_DATA_2
MapColumn~EMPLOYEES~CITY
MapColumn~EMPLOYEES~STATE
MapColumn~EMPLOYEES~POSTAL_CODE
MapColumn~EMPLOYEES~SEX
MapColumn~EMPLOYEES~BIRTHDAY
MapColumn~EMPLOYEES~STATUS_CODE
MapKey~EMPLOYEES~EMPLOYEE_ID
!
```

The keyword NoMapTable on the source table definition is a little confusing, but it tells the LogMiner Loader to not automatically create a MapTable definition because there will be an explicit MapTable definition later. The MapTable keyword:


```
MapTable~EMPLOYEES~EMPLOYEES~Replicate
```

says to map the data from the input table EMPLOYEES to the output table EMPLOYEES. The output table can contain all or a subset of the columns from the input table.

When you use the procedure JCC_CREATE_LOG_MINER_INI_FILE to create your table definitions, the result is a single definition that implicitly creates the MapTable. When the JCC LogMiner Loader was first introduced the keywords defined the source and the target at the same time. Separating the two and providing the mapping statements supports greater flexibility. In the case of this example, isolating the definitions for mapping to the target from the definitions for the source tables limits the changes needed due to source metadata changes.

Using separate Table and MapTable definitions and using the INCLUDE_FILE keyword to build the Control File is the recommended configuration.

Mapping Examples

The MapTable keyword syntax is

MapTable~<source table name>~<map table name>[,<target table rename>] \
[~<actions>[~<options>]]

The following example maps a source table to two target tables.

If you have a table T with columns a, b, c, d, e, and f and you wish to map columns a and b to target table Z and the remaining columns to target table X, you would need a Control File with the segments

```
   Table~T~ ...
o
o
o
   Column~T~a~...
   Column~T~b~...
   Column~T~c~...
   Column~T~d~...
   Column~T~e~...
o
```


o
o

MapTable~Z~...

o
o
o

MapTable~X~...

o
o
o

MapColumn~X~T~a

MapColumn~X~T~b

MapTable name must be unique in the configuration file, but the MapTable rename can have any number of duplicates. This enables us to say what we wish with a bit of indirection.

Examples of Data Transforms with MapResult

The following examples are based on the sample database packaged with Rdb. It includes a table called people that can be mapped as follows:

```
MapColumn~people~people_id
MapColumn~people~progname
MapColumn~people~progindex
MapColumn~people~seqnum
MapColumn~people~last_name
MapColumn~people~first_name
MapColumn~people~age
MapColumn~people~since_year
MapColumn~people~city~
MapColumn~people~state
MapColumn~people~huge_text
MapKey~people~people_id
```

Transform Via Table Lookup

The following statement uses MapResult to transform the value for state in the source table where it is a two character code into the full name of the state.

```
MapResult~people~state_name~xform_state_code(state)
```

The following procedure for xform_state_code can be defined in the filter database. Note that this is provided solely as an example and is not meant to be realistic. It does, however, illustrate how to work with NULLs.

```
set verify;
drop module reg_test cascade if exists;
commit;
create module reg_test
language sql
--
-- Generate NULL values if there is no state code in the table
--
function xform_state_code( in :state_code char(2))
    returns char(30);
    begin
    declare :f_state_name char(30);
    declare :row_cnt      integer;
    set :f_state_name = NULL;
    if :state_code is NULL
    then
        select state_name into :f_state_name from states s
            where s.state_code is NULL;
    else
        select state_name into :f_state_name from states s
            where s.state_code = :state_code;
        get diagnostics :row_cnt = ROW_COUNT;
        if :row_cnt = 0
        then
            set :f_state_name = 'Unknown state';
        end if;
    end if;
    return(:f_state_name);
    end;
end module;
commit;
```


To accomplish the table lookup, you will need to store the table in the filter database.

Transform Via Calculation

The following statement uses MapResult to transform based on manipulation of multiple arguments.

```
MapResult~people~xform_integer~xform_integers(age,since_year,seqnum)
```

The following procedure could be used with this MapResult statement.

```
set verify;
drop module reg_test cascade if exists;
commit;
create module reg_test
language sql
--
-- Illustrate manipulation on multiple arguments
--

function xform_integers (in :age tinyint,
    in :since_year smallint, in :seqnum bigint)
    returns bigint;
    begin
        declare :val       bigint;
        declare :NULLcount int;
        set :NULLcount = 0;
        if (:age is NULL)
        then
            set :NULLcount = :NULLcount - 1;
        end if;
        if (:since_year is NULL)
        then
            set :NULLcount = :NULLcount - 1;
        end if;
        if (:seqnum is NULL)
        then
            set :NULLcount = :NULLcount - 1;
        end if;
        if (:NULLcount = 0)
        then

             set :val = (:age * :seqnum) + :since_year;
        else
            set :val = :NULLcount;
        end if;
    return(:f_state_name);
    end;
end module;
commit;
```

Logical Name Controls for Loader Procedures

Logical names are used in many ways with the Loader. For a summary list, see the appendix, “Logical Names” on page 585.

The logical name maintenance tool is discussed in “Logical Name Controls for Loader Procedures” on page 463.

The purpose of the logical name maintenance facility is to allow you to control various Rdb run-time parameters on a Loader by Loader basis and to distinguish performance characteristics of Loader threads and LogMiner threads.

The jcc_tool_com:jcc_runtime_parameters procedure will define logical names based on information stored in an indexed file. The indexed file is available at jcc_tool_data:jcc_runtime_parameters.dat. This file is maintained by the jcc_tool_com:jcc_edit_runtime_parameters procedure.

A menuing interface is available to maintain the indexed file.

This facility is recommended for complex environments. It is not required for simple environments.

Prerequisites to Logical Name Maintenance

To use these procedures, one must first execute the jcc_tool_com:jcc_lml_user procedure to set the Loader environment.

The procedures call the jcc_runtime_parameters routine as follows:

$ jcc_runtime_parameters <process type> <LoaderName>

The process type is

- CTL for the Control process
- LML for the Loader thread processes
- CLM for the Continuous LogMiner process

This procedure will, based on the passed parameters, define logical names in process context, based on the values in the indexed file.

Procedure to Maintain the Indexed File

$ JCC_RUNTIME_PARAMETERS <job type> [<job name>]

<job type>. The job type might be an OpenVMS processing mode or a Loader job type. These are discussed more fully in “Indexed File for Maintaining Logical Names” on page 555 and in the examples that follow that section.

<job name> ***optional*****.** The job name is a specific procedure name.

The jcc_runtime_parameters procedure accesses the indexed file (JCC_runtime_parameters.dat) to find an entry. If an entry exists, the logical names associated with the entry are defined. The index entries are not case sensitive.

The index used for the lookup is composed of <job name> concatenated with <job type> and truncated to thirty-nine characters. Note the implication that the <job name> is optional. The concatenation rule means that if there is no <job name>, the index is the <job type> alone.

The logical names defined by this procedure are defined in the PROCESS logical name table. There are three logical names that are defined by default for all processes that call the jcc_runtime_parameters procedure. These are defined in the PROCESS logical name table.

- JCC_PROCESS_NAME is defined as the job name parameter to the JCC_runtime_parameters procedure.
- JCC_PROCESS_PID is defined as the process ID of the process executing the jcc_runtime_parameters procedure.
- JCC_MASTER_PID is defined as the PID of the parent process.

FIGURE 1. Example Run of JCC_RUNTIME_PARAMETERS

In the above example, the input parameters are the job type “clm” and the job name “subrdb03.” Each time a logical name is defined, this procedure prints out a line describing the logical name and it's associated value.

Output from the jcc_runtime_parameters procedure is separated into two columns by colons (":"). The column on the left is what caused the index to be defined. The column to the right is the name of the index and the value to which it was set.

In the left column, the value "Default" states that the logical name to the right was added as a default logical name. The value of "CLM" (note that this is the specified <job type>) indicates that the logical name to the right was added due to the index file entry named CLM. The value of "SUBRDB03CLM" (note that this concatenation of [job name] and <job type>) indicates that the logical name to the right was added due to the index file entry named SUBRDB03CLM. In this example, the logical name TEST value is defined as HELLOWORLD by the CLM entry, but then the same TEST logical name's value is superseded by the SUBRDB03CLM entry and set to GOODBYEWORLD.

Indexed File for Maintaining Logical Names

JCC_RUNTIME_PARAMETERS.DAT is the indexed file that the jcc_runtime_parameters procedure accesses. It should be created as part of the kit install process. It will be needed prior to using the jcc_edit_runtime_parameters procedure. Initial creation of the jcc_tool_data:jcc_runtime_parameters.dat indexed file is accomplished with

```
$ convert JCC_tool_source:JCC_runtime_parameters_sequential.dat-
jcc_tool_data:jcc_runtime_parameters.dat- /fdl=jcc_tool_-
source:jcc_runtime_parameters.fdl
```

FIGURE 2. Initial Creation of the Indexed File for Defining Logical Names

The result is an indexed file that contains a few entries that can be modified as desired. The default entries have no logical names associated with them (i.e. they are only placeholders.) They are:

BATCH
INTERACTIVE
NETWORK
OTHER
CLM
CTL
LML

There is one entry for each OpenVMS processing mode (BATCH, INTERACTIVE, NETWORK, OTHER) and one entry for each of the JCC Continuous LogMiner Loader process types (CLM, CTL, LML.)

There is no requirement to use the jcc_edit_runtime_parameters procedure, nor to even create the indexed file. The jcc_runtime_parameters will, when executed, emit

an operator request if it cannot locate the jcc_tool_data:jcc_runtime_parameters.dat file. Any file by this name will prevent the operator request. To create an empty file, use

```
$ convert nl: jcc_tool_data:jcc_runtime_parameters.dat -
/fdl=jcc_tool_source:jcc_runtime_parameters.fdl
```

```
$ jcc_runtime_parameters clm subrdb03
Default :           JCC_PROCESS_NAME="SUBRDB03"
Default :           JCC_PROCESS_PID="21400964"
Default :           JCC_MASTER_PID="21400931"
CLM :               TEST="HELLOWORLD"
SUBRDB03CLM :       TEST="GOODBYEWORLD"
```

Menuing Interface

Running

```
$ JCC_EDIT_RUNTIME_PARAMETERS
```

initiates a menuing inteface which allows the user to add, delete, modify, and display the entries in the indexed file. Using the menu interface to maintain the indexed file enables the user to define logical names for all jobs of a specific type or for individual jobs.

An example of the menuing interface is provided in “Logical Name Controls for Loader Procedures” on page 553.

Controlling Maintenance of the Indexed File

Maintenance of the indexed file can be limited to a specific system, if company policy requires it. To do so, define the logical name jcc_development_machine to be the system on which the edit functions can be used.

Source code management functions can be enabled by defining the logical name jcc_tool_cms as the valid CMS library for the parameters file. Note that since CMS does not deal exceptionally well with binary files, the indexed file is converted (automatically) to a sequential file for storage in the CMS library.

Prior to using source code management functionality, the file should be created in the CMS library. To do this, use the commands:

$ define cms$library jcc_tool_cms

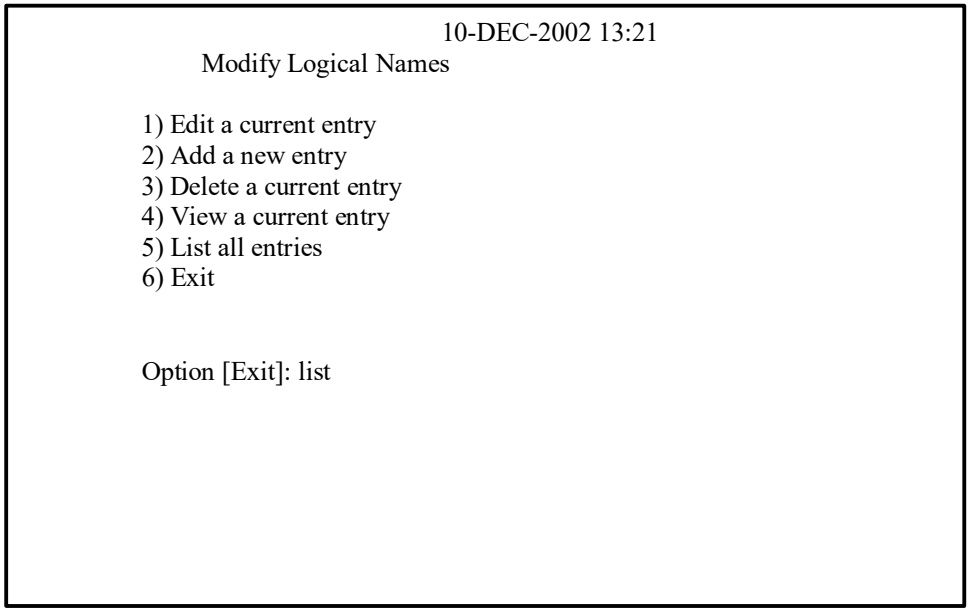
```
$ cms create element/noconcurrent/keep/input=jcc_tool_source: -
JCC_RUNTIME_PARAMETERS_SEQUENTIAL.DAT        -
"Sequential version jcc_runtime_parameters.dat"
```

The following files are used to convert the JCC_RUNTIME_PARAMETERS file between indexed and sequential formats. They should not be altered or deleted.

jcc_tool_source:JCC_RUNTIME_PARAMETERS.FDL
jcc_tool_source:JCC_RUNTIME_PARAMETERS_SEQUENTIAL.FDL

First Screen

When the menuing system comes up, you will see a screen with a list of options. At the bottom is the word “Option”. The word in braces — in this case, “[Exit]” — is the last choice made. After the colon, your choice will show.



Make your choice by

- Typing one of the options

- Typing enough characters to uniquely identify one of the options (for example “Ex” for Exit)
- Typing one of the numbers 1 through 6

Activity Screen

Each activity screen has a title at the top and a line at the bottom that tells you to “Press <return> to continue ....” Otherwise, each of the activity screens are different. Each activity screen supports one of the choices listed on the “First Screen.”

Every time you press return on one of the activity screens the First Screen reappears to give you an opportunity for another choice. To exit the application, type “Exit” or “6”.

Example Run

In the following, only the activity screens are shown. Between each of these, a first screen would reappear for a choice to be made.

Notice that this example illustrates a complete run. What you see on later screens is dependent on what was typed on earlier screens.

The rest of the run follows, displayed with two screens per page. Remember that the first screen would reappear between each of these and that each of these is in the context of the preceding choices.

```
                                        10-DEC-2002 13:21
                Modify Logical Names

List:
                        BATCH
                        CLM
                        CTL
                        INTERACTIVE
                        LML
                        NETWORK
                        OTHER


Press <return> to continue...
```

```
                                        10-DEC-2002 13:21
                Modify Logical Names

View:
                                Entry Name: clm


Press <return> to continue...
```



```
                                    10-DEC-2002 13:21
            Modify Logical Names

Edit:
                        Entry Name [CLM]:

Type REMOVE to exclude a logical name.

                        Would you like to create extra logicals [Y]: y
                        Logical Name: test
                        Logical Translation: hello world
                        Would you like to create extra logicals [Y]:
                        Logical Name: test2
                        Logical Translation: "Hello World"
                        Would you like to create extra logicals [Y]: n


Press <return> to continue...
```

```
                                    10-DEC-2002 13:21
            Modify Logical Names

View:
                        Entry Name [CLM]:

                        TEST = "HELLOWORLD"
                        TEST2 = ""Hello World""


Press <return> to continue...
```

```
                                    10-DEC-2002 13:21
            Modify Logical Names

Edit:
                        Entry Name [CLM]:

Type REMOVE to exclude a logical name.

                        TEST = ["HELLOWORLD"]:
                        TEST2 = [""Hello World""]: REMOVE
                        Remove TEST2 [Y]:
                        Would you like to create extra logicals [Y]: n


Press <return> to continue...
```

```
                                    10-DEC-2002 13:21
            Modify Logical Names

Add:
                        New Entry Name: subrdb03clm

                        Template Name: clm

Type REMOVE to exclude a logical name.

                        TEST = ["HELLOWORLD"]: goodbye world
                        Would you like to create extra logicals [Y]: n


Press <return> to continue...
```



```
                                        10-DEC-2002 13:21
            Modify Logical Names

View:
                        Entry Name [SUBRDB03CLM]:

                        TEST = "GOODBYEWORLD"


Press <return> to continue...
```

```
                                        10-DEC-2002 13:21
            Modify Logical Names

Add:
                        New Entry Name: expendable

                        Template Name: clm

Type REMOVE to exclude a logical name.

                        TEST = ["HELLOWORLD"]:
                        Would you like to create extra logicals [Y]: n


Press <return> to continue...
```

```
                                        10-DEC-2002 13:21
                Modify Logical Names

Delete:
                                Entry Name [EXPENDABLE]:



Press <return> to continue...
```

```
                                        10-DEC-2002 13:21
                Modify Logical Names

List:
                        BATCH
                        CLM
                        CTL
                        INTERACTIVE
                        LML
                        NETWORK
                        OTHER
                        SUBRDB03CLM


Press <return> to continue...
```


NLS Language Setting Example

According to the 'Oracle Database Globalization Support Guide' the conversion of the character-sets occurs at the database level. The client NLS settings are used for the entire session and the conversion occurs when the data is being read or written.

For example, the following are two example settings.

```
> NLS_CHARACTERSET AL32UTF8
> NLS_LENGTH_SEMANTICS BYTE
```

Chapter 2 of the 'Oracle Database Globalization Support Guide' states:

> "Character semantics were introduced in Oracle9i. Character semantics is useful for defining the storage requirements for multibyte strings of varying widths. For example, in a Unicode database (AL32UTF8), suppose that you need to define a VARCHAR2 column that can store up to five Chinese characters together with five English characters. Using byte semantics, this column requires 15 bytes for the Chinese characters, which are three bytes long, and 5 bytes for the English characters, which are one byte long, for a total of 20 bytes. Using character semantics, the column requires 10 characters.
>
> The following expressions use byte semantics:
>
> VARCHAR2(20 BYTE)
>
> SUBSTRB(string, 1, 20)
>
> Note the BYTE qualifier in the VARCHAR2 expression and the B suffix in the SQL function name.
>
> The following expressions use character semantics:
>
> VARCHAR2(10 CHAR)
>
> SUBSTR(string, 1, 10)
>
> Note the CHAR qualifier in the VARCHAR2 expression.

> The NLS_LENGTH_SEMANTICS initialization parameter determines whether a new column of character datatype uses byte or character semantics. The default value of the parameter is BYTE. The BYTE and CHAR qualifiers shown in the VARCHAR2 definitions should be avoided when possible because they lead to mixed-semantics databases. Instead, set NLS_LENGTH_SEMANTICS in the initialization parameter file and define column datatypes to use the default semantics based on the value of NLS_LENGTH_SEMANTICS.
>
> Byte semantics is the default for the database character set. Character length semantics is the default and the only allowable kind of length semantics for NCHAR datatypes. The user cannot specify the CHAR or BYTE qualifier for NCHAR definitions."

Based on that, note that the accented characters are being converted at insert time to multi-byte characters which will require more space in the database. If you have not provided sufficient space, you may see an exception message like this one.

```
ORA-12899: value too large for column
"RAO_STG"."RBR_KMT_GSO"."KNM_KENMERK"
(actual: 77, maximum: 75)
```

Different clients have successfully used one or the other of these NLS settings:

```
$ define nls_lang AMERICAN_AMERICA.WE8MSWIN1252
```

or

```
$ define nls_lang AMERICAN_AMERICA.UTF8
```

You could verify this by attempting to insert the same data using SQLPlus or whatever you are using for the target. If you get a different result, make certain that the Oracle NLS_LANG setting you are using when executing the SQLPlus or whatever is the same as that set for the JCC LogMiner Loader procedures.

Putting Statistics in a Database

The CSV statistics style is well suited to writing records which can be loaded into an Rdb database. All of the necessary procedures to support this are included in the kit.

This appendix is included for reference purposes, but JCC recommends using the T4 output from the JCC_LML_STATISTICS and using the TLViz tool for data analysis. See “T4 Report Example” on page 345.

Table Definition

```
jcc_tool_source:jcclml$statistics.sql

create table jcclml$statistics
       (Report_Datetime    timestamp(2)
       ,Loader_name        char(31)
       ,Commit_Datetime    timestamp(2)
       ,Row_Rate           integer
       ,Transaction_Rate   integer
       ,Source_Txn_Seconds integer(2)
       ,Statistics_Interval       integer(2)
       ,Thruput_Ratio             integer(2)
       ,Trailing_Seconds   integer(2)
       ,Input_Timeouts            integer
       ,Output_Failures    integer
       ,Loader_Threads            integer
       ,Total_Latency             integer(2)
       ,Input_Latency             integer(2)
       ,Output_latency            integer(2));
```

Define Fields

```
jcc_tool_source:JCCLML$STATISTICS.RRD

DEFINE FIELD REPORT_DATETIME DATATYPE IS TEXT SIZE IS 23.
DEFINE FIELD LOADER_NAME DATATYPE IS TEXT SIZE IS 31.
DEFINE FIELD COMMIT_DATETIME DATATYPE IS TEXT SIZE IS 23.
DEFINE FIELD ROW_RATE DATATYPE IS TEXT SIZE IS 12.
DEFINE FIELD TRANSACTION_RATE DATATYPE IS TEXT SIZE IS 12.
DEFINE FIELD SOURCE_TXN_SECONDS DATATYPE IS TEXT SIZE IS 13.
DEFINE FIELD STATISTICS_INTERVAL DATATYPE IS TEXT SIZE IS 13.
DEFINE FIELD THRUPUT_RATIO DATATYPE IS TEXT SIZE IS 13.
DEFINE FIELD TRAILING_SECONDS DATATYPE IS TEXT SIZE IS 13.
DEFINE FIELD INPUT_TIMEOUTS DATATYPE IS TEXT SIZE IS 12.
DEFINE FIELD OUTPUT_FAILURES DATATYPE IS TEXT SIZE IS 12.
DEFINE FIELD LOADER_THREADS DATATYPE IS TEXT SIZE IS 12.
DEFINE FIELD TOTAL_LATENCY DATATYPE IS TEXT SIZE IS 13.
DEFINE FIELD INPUT_LATENCY DATATYPE IS TEXT SIZE IS 13.
DEFINE FIELD OUTPUT_LATENCY DATATYPE IS TEXT SIZE IS 13.
DEFINE RECORD JCCLML$STATISTICS.
   REPORT_DATETIME .
   LOADER_NAME .
   COMMIT_DATETIME .
   ROW_RATE .
   TRANSACTION_RATE .
   SOURCE_TXN_SECONDS .
   STATISTICS_INTERVAL .
   THRUPUT_RATIO .
   TRAILING_SECONDS .
   INPUT_TIMEOUTS .
   OUTPUT_FAILURES .
   LOADER_THREADS .
   TOTAL_LATENCY .
   INPUT_LATENCY .
   OUTPUT_LATENCY .
END JCCLML$STATISTICS RECORD.
```


Creating the Table Example

```
$ sql
SQL> create database filename jcclml$statistics_db
cont> reserve 10 storage areas
cont> number of users is 8
cont> number of cluster nodes is 1
cont> default storage area is statistics_default
cont> page size is 12 blocks
cont> create storage area rdb$system filename statistics_rdb$system
cont> page size is 12 blocks
cont> create storage area statistics_default filename statistics_default
cont> page size is 12 blocks;
SQL> @jcc_tool_source:jcclml$statistics.sql
SQL> commit;
SQL> exit
```

Loading V2.1 Statistics (Example)

```
$ define JCC_LOGMINER_LOADER_STAT_CSV_DATE "|!Y4-!MN0-!D0:!H04:!M0:!S0.!C2|"
$ define JCC_LOGMINER_LOADER_STAT_OPTIONS "noheader,nointeractive"
$ define stats_rrd jcc_tool_source:JCCLML$STATISTICS.RRD
$ define error_file jccstat$errors.txt
$ pipe jcc_lml_statistics regtestrdb 60 csv | -
_$       rmu/load/commit=1/row=1/rec=(file=stats_rrd,form=delim,pre="",suff="",
null="(none)",except=error_file) -
_$       /log jcclml$statistics_db JCCLML$STATISTICS sys$pipe:
%RMU-I-DATRECSTO,   1 data records stored.
%RMU-I-DATRECSTO,   2 data records stored.
%RMU-I-DATRECSTO,   3 data records stored.
%RMU-I-DATRECSTO,   4 data records stored.
%RMU-I-DATRECSTO,   5 data records stored.
%RMU-I-DATRECSTO,   6 data records stored.
%RMU-I-DATRECSTO,   7 data records stored.
%RMU-I-DATRECSTO,   8 data records stored.
```

Data from Log Files Created with Earlier Versions

When loading data into an Rdb database from CSV output created with versions of the Loader that precede version 2.1, a special procedure is required.

```
jcc_tool_com:jcc_lml_load_pre21_statistics.com
```

This procedure converts the csv lines into the current format of csv lines. The fields that were not previously supplied will be set to NULL. The calling format is

```
jcc_lml_load_pre21_statistics                -
<pre 2.1 LML CSV log file>                   -
<LoaderName>
```

This procedure calls jcc_tool_com:jcc_lml_convert_pre21_csv_for_load.com to do the conversion. This last is a support procedure that is not called directly.

Excluding Tables from the Options File

See “Rdb LogMiner Options File” on page 102 and “Excluding Tables from the Options File” on page 103. An example is shown in the following.

```
$ d db

Directory TEST_ROOT:[TEST_LOGIN]

MF_PERSONNEL.RDB;1                              130/130      13-AUG-2010
12:25:15.23

Total of 1 file, 130/130 blocks.
$ JCCLML_RESUMES = "EXCLUDE"
$ JCC_CREATE_LOG_MINER_OPT_FILE db

Directory TEST_ROOT:[TEST_LOGIN]

MF_PERSONNEL_LM_UNL.OPT;1

Total of 1 file.
    1       !
218 substitutions
TEST_ROOT:[TEST_LOGIN]MF_PERSONNEL_LM_UNL.OPT;2 14 lines

Directory TEST_ROOT:[TEST_LOGIN]

MF_PERSONNEL_LM_UNL.OPT;2               MF_PERSONNEL_LM_UNL.OPT;1

Total of 5 files.
%DELETE-I-FILDEL, TEST_ROOT:[TEST_LOGIN]MF_PERSONNEL_LM_UNL.OPT;1 deleted
(515 blocks)
$ type MF_PERSONNEL_LM_UNL.OPT
!
!  Continuous Logminer Options file
!  Generated at 2010-08-13 12:31:28
!
table=CANDIDATES,output=rdb_logminer_output_file
table=COLLEGES,output=rdb_logminer_output_file
table=DEGREES,output=rdb_logminer_output_file
table=DEPARTMENTS,output=rdb_logminer_output_file
table=EMPLOYEES,output=rdb_logminer_output_file
table=JOBS,output=rdb_logminer_output_file
table=JOB_HISTORY,output=rdb_logminer_output_file
!_**_Excluded_by_symbol_**_table=RESUMES,output=rdb_logminer_output_file
table=SALARY_HISTORY,output=rdb_logminer_output_file
table=WORK_STATUS,output=rdb_logminer_output_file
```

```
$ d db

Directory TEST_ROOT:[TEST_LOGIN]

MF_PERSONNEL.RDB;1                                130/130      13-AUG-2010
12:25:15.23

Total of 1 file, 130/130 blocks.
$ JCCLML_RESUMES = "EXCLUDE"
$ JCC_CREATE_LOG_MINER_OPT_FILE db

Directory TEST_ROOT:[TEST_LOGIN]

MF_PERSONNEL_LM_UNL.OPT;1

Total of 1 file.
    1       !
218 substitutions
TEST_ROOT:[TEST_LOGIN]MF_PERSONNEL_LM_UNL.OPT;2 14 lines

Directory TEST_ROOT:[TEST_LOGIN]

MF_PERSONNEL_LM_UNL.OPT;2              MF_PERSONNEL_LM_UNL.OPT;1

Total of 5 files.
%DELETE-I-FILDEL, TEST_ROOT:[TEST_LOGIN]MF_PERSONNEL_LM_UNL.OPT;1 deleted
(515 blocks)
$ type MF_PERSONNEL_LM_UNL.OPT
!
!  Continuous Logminer Options file
!  Generated at 2010-08-13 12:31:28
!
table=CANDIDATES,output=rdb_logminer_output_file
table=COLLEGES,output=rdb_logminer_output_file
table=DEGREES,output=rdb_logminer_output_file
table=DEPARTMENTS,output=rdb_logminer_output_file
table=EMPLOYEES,output=rdb_logminer_output_file
table=JOBS,output=rdb_logminer_output_file
table=JOB_HISTORY,output=rdb_logminer_output_file
!_**_Excluded_by_symbol_**_table=RESUMES,output=rdb_logminer_output_file
table=SALARY_HISTORY,output=rdb_logminer_output_file
table=WORK_STATUS,output=rdb_logminer_output_file
```


APPENDIX 1 *More Resources*

The JCC LogMiner Loader kit includes the software, this documentation, and numerous examples. This appendix lists kit contents, example directory trees, logical names used, operator alarms available and frequently asked questions (FAQ).

Blogs are referenced throughout the documentation and provide important insights. Blogs have the additional advantage of being more easily updated than documentation. See what's available at http://www.jcc.com/lml-blog

Kit Contents - Directories and Files

The charts show the file names and uses for files in each of the branches of the directory tree.

Some special versions are required for particular circumstances. The characters "_ST.exe" indicate the variant executables that aren't linked with the OpenVMS p-threads library. The characters "_O92.exe" indicate the Oracle 9.2 shareable images.

The files listed are for the alpha images.

Directory or Logical Name	Logical Name for the Directory or Use of the Logical Name
JCC_TOOL_ROOT:[LOCAL]	**JCC_TOOL_LOCAL**
this_file_may_be_deleted.txt	The sole purpose of this file is to create the jcc_tool_root:[local] directory as part of the installation. The file may be deleted by the user (or not.)

Directory or Logical Name	Logical Name for the Directory or Use of the Logical Name
JCC_TOOL_ROOT:[SOURCE]	**JCC_TOOL_SOURCE:**
jcc_runtime_parameters.fdl	FDL file used to create the indexed file for the jcc_runtime_parameters procedure which supports the logical name facility. See “Logical Name Controls for Loader Procedures” on page 463.
jcc_runtime_parameters_sequential.dat	A sequential version of the jcc_runtime_parameters indexed file that has some default parameters defined. This file can be used to create an initial jcc_runtime_parameters indexed file.
jcc_runtime_parameters_sequential.fdl	FDL file used to convert the jcc_runtime_parameters index file to a sequential file if changes are to be stored in a CMS library.
loader_virtual_columns.tbl	File used with Tuxedo in the specification of NULLs.
jcclml$statistics.rrd	Record definition for loading jcc_lml_statistics CSV files into the jcclml$statistics table.
jcclml_msg.doc	Provides a list of exception messages, their meaning, and what actions can be taken to rectify the problem.
packet-commented.dtd	transaction-based DTD for XML targets
packet-record-commented.dtd	record-based DTD for XML targets

Directory or Logical Name	Logical Name for the Directory or Use of the Logical Name
JCC_TOOL_ROOT:[COM]	**JCC_TOOL_COM:**
jcc_add_odbkey_set_index.com	Procedure to create scripts to maintain dbkey indexes
jcc_move_data_to_oracle.com	Procedure to generate scripts to create RMU/UNLOAD and SQL*Loader scripts to move data from an Rdb database to an Oracle database. This procedure should not be used with NULL values. See “Populating the Target” on page 128.
jcc_link_data_to_oracle.com	Procedure to create scripts to move data from Rdb to Oracle using database links
low_overhead_oci.com	Sample SQL*net definition file to support database links

Directory or Logical Name	Logical Name for the Directory or Use of the Logical Name
JCC_TOOL_ROOT:[COM]	**JCC_TOOL_COM:**
jcc_clml_reopen_log.com	Procedure to cause the CLML control process to reopen its log files
jcc_clml_shutdown.com	Procedure to shut down the CLML process in an orderly way
jcc_lml_bugcheck.com	Procedure to request a Loader session to bugcheck. This procedure should only be used at the request of JCC LogMiner Loader Support. It will write detailed Loader internal information to the log file.
jcc_local_environment.com	Procedure to copy to the jcc_tool_local_directory and edit to create Loader-related, site-specific environment changes. This procedure can create logical names, modify file protection, install customer, images, etc. If this file exists, it is executed as the last step of the JCC_DBA_STARTUP procedure. See “Tailoring Procedures” on page 59.
jcc_tool_security.com	Procedure to copy to the jcc_tool_local_directory and edit to customize the Loader directory tree security to meet site-specific needs. See “Tailoring Procedures” on page 59.
clean_up_ivp.com	Procedure to clean up the IVP directories. They are intentionally left in place to serve as examples.
jcc_create_logminer_highwater.com	Sample procedure to create high water structures
jcc_create_log_miner_ini_file.com	Procedure to create sample Control File for metadata for a database
jcc_create_log_miner_opt_file.com	Procedure to create LogMiner options file for a database
ivp_logminer_loader_differences.com	Procedure run during the IVP to detect differences
jcc_compare_tables.com	Procedure to compare two tables between slaved databases
jcc_dba_startup.com	Startup procedure. Must be executed during system startup. *Do note copy this procedure,* execute it from the installation directory as it is self-referent
jcc_lml_api_startup.com	Sample startup procedure; must be executed if customer-supplied API is used
jcc_logminer_loader_compare.com	Differences procedure to compare tables in two different databases
jcc_logminer_loader_ivp.com	Installation verification procedure
jcc_rename_file.com	Procedure to rename files
jcc_run_clm.com	Procedure to run continuous LogMiner. Do not run this directly
jcc_run_clm_lml.com	Run the continuous LogMiner/Loader session
jcc_run_lml.com	Procedure to run the continuous Loader. Do not run this directly
jcc_unload_aijs.com	Example procedure to run LogMiner against the AIJs.
shareable_install.com	Procedure executed during system startup to install JCC images
unload_load.com	Useful procedure to move data between two databases fast
unload_load_setup.com	Initialization procedure for the above. Edit to add your favorite performance options
jcc_create_log_miner_tux_-field_def.com	Procedure to create Tuxedo field header files for Tuxedo.

Directory or Logical Name	Logical Name for the Directory or Use of the Logical Name
JCC_TOOL_ROOT:[COM]	**JCC_TOOL_COM:**
dba_parse_lml_stats.com	Ad hoc procedure to parse log files and produce CSV output.
jcc_add_logical.com	Support procedure. Not accessed directly.
jcc_clml_communicate.com	Support procedure. Not accessed directly.
jcc_clml_start_thread.com	Procedure to manually start a thread.
jcc_clml_stop_thread.com	Procedure to manually stop a thread.
jcc_deassign_logical.com	Support procedure. Not accessed directly.
jcc_del_logical.com	Support procedure. Not accessed directly.
jcc_edit_runtime_parameters.com	See “Logical Name Controls for Loader Procedures” on page 463.
jcc_find_lml_processes.com	Procedure to list CLML processes running in a cluster. See “Finding Sessions in the Cluster” on page 455.
jcc_find_message.com	Support procedure. Not accessed directly.
jcc_get_db_info.com	Support procedure. Not accessed directly.
jcc_get_loader_info.com	Support procedure. Not accessed directly.
jcc_lml_license.com	See “Applying the License Key” on page 54.
jcc_runtime_parameters.com	See “Loader Process Logical Names” on page 106.
repair_aip_entries.com	See “Preparing Sources Created with Earlier Versions of Rdb” on page 104.
jcc_lml_convert_pre21_csv_for_-load.com	Support routine for jcc_lml_pre21_statistics.com.
jcc_lml_pre21_statistics.com	Converts the csv lines into the current format of csv lines.
jcc_lml_create_control_file.com	See “Building the Metadata Control File” on page 222.
jcc_lml_user.com	Provides the foundation for running other procedures. See “Groundwork for Communication” on page 55.
jcc_add_logical_table.com	Support routine
jcc_copy_logical_values.com	Support routine
jcc_del_logical_table.com	Support routine
jcc_clml_maximum_threads.com	Procedure to change the maximum number of threads while the Loader is running. See “Keyword: Parallel” on page 279.
jcc_clml_minimum_threads.com	Procedure to change the minimum number of threads while the Loader is running. See “Keyword: Parallel” on page 279.
dba_conditions.com	Procedure creates DCL symbols for each of the Loader exit statuses. These symbols can be used in your DCL procedures to test for specific statuses or perform specific actions. *This file will be updated with each release of the Loader.*
jcc_lml_t4_template.com	Template to illustrate getting T4 statistics. See page 345.
jcc_extract_thread_logs.com	See “Splitting Log Files Into a File for Each Thread” on page 358.

Directory or Logical Name	Logical Name for the Directory or Use of the Logical Name
JCC_TOOL_ROOT:[JAVA]	**JCC_TOOL_JAVA:**
jcclmljdbc2.class	Class for Loader JDBC interface as re-implemented for Version 3.2
lmltable.class	Class for Loader JDBC interface as re-implemented for Version 3.2

Directory or Logical Name	Logical Name for the Directory or Use of the Logical Name
JCC_TOOL_ROOT:[API]	**JCC_TOOL_API:**
jcc_lml_api.h	Sample C language header file that describes the API
test_api_interface.exe	Sample API

Directory or Logical Name	Logical Name for the Directory or Use of the Logical Name
JCC_TOOL_ROOT:[EXE]	**JCC_TOOL_EXE:**
jcc_lml_statistics.exe	Reports information about the progress of the loader
jcc_lml_statistics_st.exe	Reports information about the progress of the loader (ST variant)
jcc_logminer_loader.exe	The JCC LogMiner Loader
jcc_logminer_loader_st.exe	The JCC LogMiner Loader (ST variant)
jcc_continuous_logminer_loader.exe	Control process to run the Loader
jcc_continuous_logminer_loader_st.exe	Control process to run the Loader (ST variant)
mbx.exe	Useful image to create mailboxes. Used by the unload-load procedure
jcc_continuous_logminer_loader_ev6.exe	All .exe files that end with _ev6 are the EV6 optimized version of the same file. This one is the EV6 version of the control process to run the continuous Loader
jcc_lml_dump_checkpoint.exe	See “Re-opening the Logs” on page 74 and “Displaying Checkpoint Information” on page 372.
jcc_lml_dump_checkpoint_st.exe	st variant of jcc_lml_dump_checkpoint.exe
jcc_lml_dump_checkpoint_ev6.exe	All .exe files that end with _ev6 are the EV6 optimized version of the same file.
jcc_lml_show_locks.exe	See “Control-t and Statistics Running Time” on page 318.
jcc_lml_show_locks_ev6.exe	All .exe files that end with _ev6 are the EV6 optimized version of the same file.
jcc_lml_statistics.exe	See “Online Statistics Monitor” on page 314.
jcc_lml_statistics_ev6.exe	All .exe files that end with _ev6 are the EV6 optimized version of the same file.

Directory or Logical Name	Logical Name for the Directory or Use of the Logical Name
JCC_TOOL_ROOT:[EXE]	**JCC_TOOL_EXE:**
jcc_logminer_loader_ev6.exe	All .exe files that end with _ev6 are the EV6 optimized version of the same file.This one is the main Loader image.
jcc_version.exe	Procedure to display the current version of the JCC tool kit.
jcc_version_st.exe	Procedure to display the current version of the JCC tool kit (ST variant).
jcc_version_ev6.exe	Procedure to display the current version of the JCC tool kit (EV6 optimized variant).
jcc_lml_data_pump.exe	See “Syntax” on page 507.
jcc_lml_data_pump_st.exe	See “Syntax” on page 507. (ST variant)
jcc_lml_data_pump_ev6.exe	See “Syntax” on page 507. (EV6 optimized variant)

Directory or Logical Name	Logical Name for the Directory or Use of the Logical Name
JCC_TOOL_ROOT:[SHARE]	**JCC_TOOL_SHARE:**
jcc_logminer_loader_base_share.exe	Shareable image containing Rdb symbols etc. Used, for example, in swapping between Loader versions. See “Multi-Version Support” on page 56.
jcc_logminer_loader_oci_share.exe	Shareable image containing loader interaction with Oracle SQL*net
jcc_logminer_loader_rdb_share.exe	Shareable image containing loader interaction with Rdb
jcc_logminer_loader_share.exe	Shareable image containing main code of the Loader
jcc_logminer_loader_base_share_ ev6.exe	All .exe files that end with _ev6 are the EV6 optimized version of the same file.
jcc_logminer_loader_oci_share_ ev6.exe	All .exe files that end with _ev6 are the EV6 optimized version of the same file.
jcc_logminer_loader_oci_share_ ev6_o9.exe	Shareable image linked with 9.0.1 Oracle SQL*Net and optimized for EV6
jcc_logminer_loader_oci_share_o9.exe	Shareable image linked with 9.0.1 Oracle SQL*Net
jcc_logminer_loader_rdb_share_ ev6.exe	All .exe files that end with _ev6 are the EV6 optimized version of the same file.
jcc_logminer_loader_share_ev6.exe	All .exe files that end with _ev6 are the EV6 optimized version of the same file.
jcc_logminer_loader_rdb_share_st.exe	Sharable image linked for Rdb targets.
jcc_logminer_loader_tuxw_share.exe	Sharable image linked as a Tuxedo workstation client
jcc_logminer_loader_tuxw_share_ ev6.exe	All .exe files that end with _ev6 are the EV6 optimized version of the same file.
jcc_logminer_loader_tux_share.exe	Sharable image linked as a Tuxedo domain member.
jcc_logminer_loader_tux_share_ev6.exe	All .exe files that end with _ev6 are the EV6 optimized version of the same file.

Directory or Logical Name	Logical Name for the Directory or Use of the Logical Name
JCC_TOOL_ROOT:[SHARE]	**JCC_TOOL_SHARE:**
jcc_logminer_loader_base_share_st.exe	Sharable image linked without the OpenVMS p-threads library.
jcc_logminer_loader_share_st.exe	Sharable image linked without the OpenVMS p-threads library.
jcc_logminer_loader_oci_share_o92.exe	Sharable image linked for OCI and Oracle 9.2 with the OpenVMS p-threads library.
jcc_logminer_loader_o-ci_share_ev6_o92.exe	Sharable image linked to optimize EV06 and for OCI and Oracle 9.2 with the OpenVMS p-threads library.
jcc_logminer_loader_jdbc2_share.exe	
jcc_logminer_loader_jdbc2_share_ev6.exe	

Directory Tree of Examples in the Kit

The examples come largely from the regression testing and are included to provide a foundation. Find the examples directory tree in jcc_tool_root:[examples].

[Examples]

api.dir

jcc_loader_sample_db.com

jdbc.dir
ora.dir

rdb.dir

tux.dir

[Examples.api]

api_message.msg

build_api.com

build_parse_api.com

loader_api_interface.ini

loader_api_time_format.ini

loader_control_api.ini

loader_exclude_api.ini

loader_includes_api.ini

loader_metadata.ini
parse_api_output.bas
regtest_api.c
regtest_api.opt
run_api_clm.com
test_api_interface.c
test_api_interface.opt

[Examples.jdbc]

rdb.dir
sqs.dir

[Examples.jdbc.rdb]

loader_regression_test_control_jdbc.ini
loader_regression_test_excludes_jdbc.ini
loader_regression_test_includes_jdbc.ini
loader_regression_test_metadata_jdbc.ini
loader_regression_test_virtual_jdbc.ini
run_jdbc_clm.com
target_rdb_jdbc_def.com

[Examples.jdbc.rdb]

loader_regression_test_ctl_jdbc_sqlsrv.ini
loader_regression_test_excludes_jdbc.ini
loader_regression_test_includes_jdbc.ini
loader_regression_test_metadata_jdbc.ini
loader_regression_test_virtual_jdbc.ini
run_jdbc_sqlsrv_com.com

[Examples.ora]

loader_control_ora.ini
loader_control_ora_interface.ini
loader_excludes_ora.ini

loader_includes_ora.ini
loader_metadata.ini
oci.dir
run_ora_clm.com

[Examples.ora.oci]

db_links.sql
sample.sqs
sample_low_overhead_oci.com
sample_low_overhead_oci.sql
tnsnames.ora_on_target
tnsnames.ora_vms

[Examples.rdb]

loader_control.ini
loader_excludes.ini
loader_includes.ini
loader_metadata.ini
run_rdb_clm.com
run_rdb_clm.com

[Examples.tux]

application.dir
loader_excludes.ini
loader_includes.ini
loader_metadata_tux.ini
loader_tux.ini
loader_tux_interface.ini
run_rdb_clm_tux.com
tuxedo_date_format.ini

[Examples.tux.application]

build_details.com
build_people.com

create_lml_regtest_queues.com

create_tlog.com

create_tuxconfig.com

details.c

people.c

tuxconfig.ubb

[Examples.xform]

xform_control_file.ini
xform_interface.ini
xform_loader_regression_test_db_maptbl.ini
xform_loader_regression_test_db_table.ini
xform_readme.txt
xform_loader_regression_test_functions.sql
xform_technical.ini

Operator Alarms

The Loader and Control processes will generate OPCOM messages when exiting with an exception. The Control process will generate an OPCOM message if it detects that the CLM process exits with an exception. The statistics monitor generates a message if the acceptable lag threshold is exceeded and, again, if the Loader catches up.

The OPCOM messages are sent to the CENTRAL operator by default. The operator class can be set to other or additional options through use of the Operator keyword. See “Keyword: Operator” on page 274.

From the Loader

```
JCCLML: has terminated with a command line syntax error
JCCLML: '<LoaderName>' has terminated - error cancelling asynch Freeze lock; <exception message>
JCCLML: '<LoaderName>' has terminated - error obtaining FREEZE lock; <exception message>
JCCLML: '<LoaderName>' has terminated in initialization with an exception; <exception message>
JCCLML: '<LoaderName>' has terminated with an exception; <exception message>
```

For Highwater Mark Creation

```
JCCCTL: '<LoaderName>' has no existing highwater data.  (CREATE, QUIT)
JCCLML: '<LoaderName>' has no existing highwater data.  (CREATE, QUIT)
JCCLML: '<LoaderName>' has encountered a deadlock on highwater read.  (QUIT)
```


From the Monitor

```
JCCSTAT: JCC Loader '<LoaderName>' output is trailing realtime by <N.NN> seconds
JCCSTAT: JCC Loader '<LoaderName>' input is trailing realtime by <N.NN> seconds
JCCSTAT: JCC Loader '<LoaderName>' is below tardy interval of <N.NN> seconds; output trailing real-
time by <N.NN> seconds
JCCSTAT: JCC Loader '<LoaderName>' is below tardy interval of <N.NN> seconds; input trailing realtime
by <N.NN> seconds
JCCSTAT: JCC Loader '<LoaderName>' latency is <N.NN> seconds
JCCSTAT: JCC Loader '<LoaderName>' input latency is <N.NN> seconds
JCCSTAT: JCC Loader '<LoaderName>' output latency is <N.NN> seconds
JCCSTAT: JCC Loader '<LoaderName>' is below tardy interval of <N.NN> seconds; latency is <N.NN> sec-
onds
JCCSTAT: JCC Loader '<LoaderName>' is below tardy interval of <N.NN> seconds; input latency is<N.NN>
seconds
JCCSTAT: JCC Loader '<LoaderName>' is below tardy interval of <N.NN> seconds; output latency is <N.NN>
seconds
```

From the Control Process

```
JCCCTL: has terminated with a command line syntax error
JCCCTL: has terminated due to LoaderName not being set.
JCCCTL: '<LoaderName>' CLM has terminated prior to setting PID
JCCCTL: '<LoaderName>' CLM has terminated with an exception; <exception message>
JCCCTL: '<LoaderName>' Failed to generate heartbeat.  Disabling feature; <exception message>
JCCCTL: '<LoaderName>' LML thread <thread number> has terminated with an exception; <exception mes-
sage>
JCCCTL: '<LoaderName>' LML thread <thread number> has terminated without properly maintaining the
active bitmap
JCCCTL: '<LoaderName>' a LML thread has terminated without properly maintaining the active bitmap
JCCCTL: '<LoaderName>' has terminated - error cancelling asynch Freeze lock; <exception message>
JCCCTL: '<LoaderName>' has terminated - error cancelling asynch LSN lock; <exception message>
JCCCTL: '<LoaderName>' has terminated - error obtaining FREEZE lock; <exception message>
JCCCTL: '<LoaderName>' has terminated - failure initializing checkpoint; <exception message>
JCCCTL: '<LoaderName>' has terminated - failure reading restart information; <exception message>
JCCCTL: '<LoaderName>' has terminated - failure formatting AERCP
JCCCTL: '<LoaderName>' has terminated clearing event flag for LML startup with an exception; <excep-
tion message>
JCCCTL: '<LoaderName>' has terminated during heartbeat sharable initialization with an exception;
<exception message>
JCCCTL: '<LoaderName>' has terminated during initialization with an exception; <exception message>
JCCCTL: '<LoaderName>' has terminated during input stream creation with an exception; <exception mes-
sage>
JCCCTL: '<LoaderName>' has terminated during sharable initialization with an exception; <exception
message>
JCCCTL: '<LoaderName>' has terminated obtaining event flag for CLM with an exception; <exception mes-
sage>
JCCCTL: '<LoaderName>' has terminated obtaining event flag for LML with an exception; <exception mes-
sage>
JCCCTL: '<LoaderName>' has terminated opening logfile channel for CLM with an exception; <exception
message>
```



```
JCCCTL: '<LoaderName>' has terminated opening logfile channel for LML with an exception; <exception
message>
JCCCTL: '<LoaderName>' has terminated setting event flag for CLM with an exception; <exception mes-
sage>
JCCCTL: '<LoaderName>' has terminated setting event flag for LML with an exception; <exception mes-
sage>
JCCCTL: '<LoaderName>' has terminated setting startup event flag for LML with an exception; <excep-
tion message>
JCCCTL: '<LoaderName>' has terminated starting CLM with an exception; <exception message>
JCCCTL: '<LoaderName>' has terminated starting LML with an exception; <exception message>
JCCCTL: '<LoaderName>' has terminated waiting for CLM rundown with an exception; <exception message>
JCCCTL: '<LoaderName>' has terminated waiting for LML startup with an exception; <exception message>
JCCCTL: '<LoaderName>' has terminated waiting for LML with an exception; <exception message>
JCCCTL: '<LoaderName>' has terminated with an exception attempting to translate a logical; <exception
message>
```

Example

```
%%%%%%%%%%  OPCOM  23-MAY-2017 21:45:40.51  %%%%%%%%%%    (from node ARES   at 23-
  MAY-2017 21:45:40.62)

Message from user JEFF_TUX on ARES

JCCLML: 'REGTSTTXIPF' has terminated with an exception;  %DBA-I-INPUT_EOS, End of the
  input stream has been reached.
```


Logical Names

OpenVMS logical names are extensively used with the JCC LogMiner Loader. These include

1. The logical name to provide your license key to the Loader.
2. Logical names to tell the Loader where to find or where to put various things.
3. Logical names to change the features of the Loader.
4. Logical names needed to support third-party products that the Loader utilizes.

There are additional logical names defined by the Loader startup procedure that must be run at system startup.[1]

This appendix provides a summary of the logical names that you may want to define. A brief comment is provided to help you recognize which might be important in your environment. Please consult the reference listed for a more complete explanation of how and when to use each.

Logical Names in the Architecture

Logical names can provide additional input to control how the Loader behaves. The simplified architecture diagram shows logical names as input. The output from Rdb LogMiner and a Loader Control File are required input. Most logical names are optional.

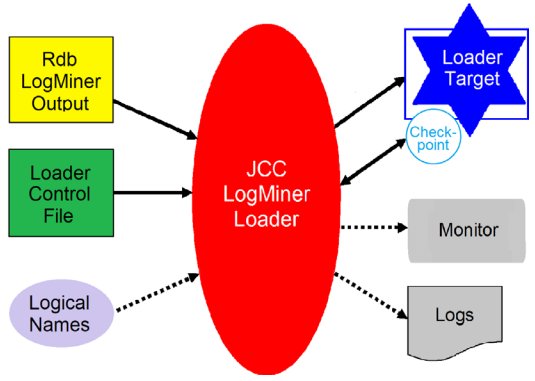


FIGURE 1. JCC LogMiner Loader Input and Output

1. The startup procedure is discussed in "System Startup" on page 59. Note that the system logical name table is used for the standard version of the Loader and, *if* the Loader is installed multi-version, the logical names for the variant are defined in the version-specific logical name table. Most of these logical names are not reflected in the chart to follow because they are not logical names that you will define to tune the Loader.

Use of Logical Names

To create system logical names requires SYSNAM privilege. See “Privileges” on page 48. The startup procedure must also be used carefully to build the expected directory structure. See “Startup and Directories” on page 53 and “Set-Up for the Standard Version” on page 55.

A maintenance facility for the logical names used is part of the Loader kit. See “Logical Name Controls for Loader Procedures” on page 463.

Although definition of logical names is, generally, not required, certain logical names can be quite important. JCC_AIJ_backup_spec for example, is critical if you are going to have AIJ backups to process. The necessity for a logical name is generally situation dependent. For that reason, this chapter is a summary only. Consulting the reference pages listed is strongly recommended.

Summary of Logical Names

Logical names that may be defined by the Loader user are shown listed alphabetically. References and comments are shown following the logical name.

Logical Name	Reference Page	Use and Comment
jcc_add_clm_debug		
	“The Log Files” on page 356	Use to control the level of logging by the Rdb LogMiner during development or problem solving.
jcc_add_clm_ignore_old_version_tables		
	“A Rare Exception: Old Table Versions” on page 452	Use in the exceptional circumstance that an old table version gets into the AIJ.
jcc_add_clm_log		
	“The Log Files” on page 356	Use to control the level of logging by the Rdb LogMiner during development or problem solving.
jcc_add_clm_mbx_asynch		
	“Asynchronous Writes to the VMS Mailbox” on page 414	Use to accept the Rdb asynchronous writes that were new with Version 7.3.1.3. Note that using this can trigger performance issues.
jcc_add_clm_quick_sort		
	“LogMiner Quick Sort” on page 468	Use to override the default and control the CLM quick sort with an integer between 10 and 100,000.
jcc_add_clm_shared_read		
	“Multiple CLM Processes” on page 71	Define to TRUE to enable multiple CLM processes to read from the same AIJ backups at the same time.

Logical Name	Reference Page	Use and Comment
jcc_add_clm_sortwork_files		
	"Sortwork File Control" on page 398	Use to tune performance.
jcc_add_clm_statistics		
	"Modifying CLM Statistics Output" on page 355	Use to modify or disable LogMiner statistics.
jcc_add_clm_trace		
	"The Log Files" on page 356	Use to control the level of logging by the Rdb LogMiner during development or problem solving.
jcc_aij_backup_spec		
	"Name Changes for Stored Procedures" on page 65 (the last bullet) and "Finding AIJ Backups" on page 71 and "Restart for Continuous Log-Miner and Loading" on page 422	REQUIRED if starting CLML in the backed up AIJ files. Define (in a comma separated list) to the searchlist of directories needed.
jcc_clm_log_mailbox_size		
	"Space in the CLM Logging Mailbox" on page 467	May be used to *increase* the mailbox size used for CLM logs.
jcc_clml_heartbeat_enable		
	"Loader Heartbeat and AIJ Backup" on page 475	Use to enable heartbeat processing. (Disable with zero.)
jcc_clml_heartbeat_interval		
	"Loader Heartbeat and AIJ Backup" on page 475	Use to set the heartbeat interval. (Set to other than zero for only one Loader per source database.)
jcc_clml_logging_style		
	"Thread Log Files" on page 357	Use to control log files for use with threads (parallel use of the Loader).
jcc_clml_process_name_separator		
	"Controlling Generated OpenVMS Process Names" on page 456	Use to modify characters used to replace blanks. (Not generally needed.)
jcc_clml_remove_static_aercp		

Logical Name	Reference Page	Use and Comment
	"Static Mode Exceptions" on page 78	Needed for Rdb prior to 7.2.4.0 when running the Loader in static mode.
jcc_comc_va_memory_model		
	"Analyzing Performance" on page 408	Use to achieve the performance benefits of 64-bit memory.
jcc_development_machine		
	"Logical Name Controls for Loader Procedures" on page 463	Use with the logical name maintenance facility if it is desireable to limit logical name maintenance to a specific system.
jcc_lm_default_from_timestamp		
	"Finer Control of the Start Time" on page 91	Change the start time for static LogMiner.
jcc_lml_activation_log_attempts		
	See "Activation Log" on page 368	The number of attempts that the Loader makes to write to the activation log. Define to a positive integer between 1 and 2.1 billion. The default is 200.
jcc_lml_cache_use_table		
	"Restrictions on the Cache Checkpoint Server" on page 438	Used with the Special Restart that intentionally skips records on recovery.
jcc_lml_case_sensitive_target		
	"jcc_lml_case_sensitive_target" on page 154	Turn on Loader support for case sensitive naming.
jcc_lml_checkpoint_cache_override		
	"Logical Names for the Cache Checkpoint Server" on page 440	Used with the Special Restart that intentionally skips records on recovery.
jcc_lml_data_pump_structure_file		
	"Structure File and Table Hierarchy" on page 508	Use to provide the name of the structure file to use with the data pump.
jcc_lml_data_pump_driver_file		
	"Driver Directive and Column Values" on page 516	Use to provide the name of the driver file to use with the data pump.
jcc_lml_dp_trace_no_rows		
	"Unwanted Output" on page 518	Use to turn off the informational message "No rows found matching the selection criteria".

Logical Name	Reference Page	Use and Comment
jcc_lml_file_checkpoint_secs		
	"Change the File Flush Interval" on page 213	Gain more control when using a file target.
jcc_lml_file_checkpoint_unit		
	"Change the Units for Checkpoint Intervals" on page 213	Gain more control when using a file target.
jcc_lml_data_pump_exception_file		
	"Exceptions" on page 517	Use to provide the name of the exception file to be used by the data pump.
jcc_lml_java_bootclasspath		
	"OpenVMS Java Changes Across Versions" on page 172	Include or exclude bootclass paths.
jcc_lml_java_command_line		
	"Java Command Line Options" on page 147	JDBC specific. Use to provide command line options to the Java Virtual Machine.
jcc_lml_jdbc_batch_disable		
	"Logical Names to Use with JDBC" on page 151	JDBC specific. Define to 1 to disable batching for drivers that cannot handle this feature. For some drivers, batching is automatically disabled.
jcc_lml_jdbc_batch_size		
	"Logical Names to Use with JDBC" on page 151	JDBC specific. Define to an interger that will be the maximum number of records in a batch. Coordinate this with the amount of JVM available.
jcc_lml_jdbc_default_version		
	"User Procedure for JDBC" on page 146	JDBC specific. Define as the default Java version
jcc_lml_jdbc_default_setup		
	"User Procedure for JDBC" on page 146	JDBC specific. Define as the default Java setup procedure
jcc_lml_jdbc_gc_commit		
	"Logical Names to Use with JDBC" on page 151	JDBC specific. Define to 1 to force garbage collection when using older versions of Java.
jcc_lml_jdbc_name_delim_start and jcc_lml_jdbc_name_delim_end		
	"Logical Names to Use with JDBC" on page 151	JDBC specific for column name delimiters. To use delimiters, both logical names must be defined, even if the same character is used for both starting and ending a delimited name.

Logical Name	Reference Page	Use and Comment
jcc_lml_jdbc_single_statement		
	"Logical Names to Use with JDBC" on page 151	JDBC specific. Define to 1 to disable use of multiple, concurrent open statements for drivers that cannot handle this feature. For some drivers, batching is automatically disabled.
jcc_lml_jdbc_target_schema		
	"jcc_lml_jdbc_target_schema" on page 154	Provides the schema name for end targets that require it.
jcc_lml_locking_level		
	"Locking and Locking Control Modes" on page 402	Use to control locking for tuning performance
jcc_lml_null_delta_date		
	"Delta Dates Represented as NULL" on page 473	Use to represent delta dates as NULLs, rather than negative dates.
jcc_lml_optimize_insert		
	"Optimization of Inserts" on page 141	Loader performance is optimized for mixed update/insert environments. In heavy insert or insert only environments, performance may be better with this logical name set to 1.
jcc_lml_ora_roll_disc		
	"Alternative Control for Exceptions in the Target" on page 141	Cause the Loader to retry when it encounters an exception when writing to an Oracle target.
jcc_lml_null_bad_date		
	"Unexpectedly Large Dates from the Source" on page 472	Use to process rows with dates beyond the supported range as if the date is null.
jcc_lml_Rdb_isolation_level		
	"Changing the Isolation Level" on page 121	Use, if necessary, for Rdb targets.
jcc_lml_result_txn_prestart		
	"Transaction Control for the FilterMap Database" on page 462	Provides the option of prestarting transactions on the FilterMap database.
jcc_lml_result_txn_type		
	"Transaction Control for the FilterMap Database" on page 462	Gain more control of the Loader specific database that supports FilterMap and MapResult.

Logical Name	Reference Page	Use and Comment
jcc_lml_store_unsigned		
	"Unsigned Values for Materialized Data" on page 474	If your target supports unsigned data, you can direct the Loader to send it for some of the materialized data.
jcc_lml_target_log_threshold		
	See "Target Latency Reflected in the Log" on page 365	Used to define the number of seconds of target latency before a message is written to the log. Define to a positive real value. Default is 3600.
jcc_lml_tuxedo_async_limit		
	"ASYNCH Limit Configuration for Tuxedo Target" on page 196	Used to reduce the maximum number of asynch buffers that can be sent concurrently. (This may improve performance.)
jcc_lml_tuxedo_convert_latency		
	"Statistics Reporting and Tuxedo Targets" on page 194	Supplies a comma separated list of Tuxedo functions to define a finer distinction on latency reported.
jcc_lml_tuxedo_log_threshold		
	"Determining Slowness in the Tuxedo Interface" on page 194	The more generic jcc_lml_target_log_threshold may be used instead. Maintained for backward compatibility.
jcc_lml_use_oracle_rowid		
	"Tuning for the Virtual Address Space" on page 140	Enables control of a strange Oracle use of virtual address space.
jcc_logminer_loader_async_retry_delay		
	"Tuxedo Limits and Loader Retry Delay" on page 190	Tuxedo specific. Define to the number of seconds to wait after an attempt to retrieve replies for outstanding asynchronous calls.
jcc_logminer_loader_db		
	"Logical Names for the Cache Checkpoint Server" on page 440	Define to specify the target database for *static* Loading.
jcc_logminer_loader_filter_dir		
	"FilterMap Database" on page 243	Define to set (modify the default) the directory for the filter database that supports FilterMap.
jcc_logminer_loader_filter_name		
	"FilterMap Database" on page 243	Define to set (modify the default) the name for the filter database that supports FilterMap.

Logical Name	Reference Page	Use and Comment
jcc_logminer_loader_hw_response		
	"Running the Loader for the First Time" on page 107	Define to 'CREATE' if you will be starting over frequently and wish to avoid the operator interaction to initiate the Loader restart context.
jcc_logminer_loader_init		
	"Modes of Operation" on page 77	Define to specify the Control File for *static* Loading.
jcc_logminer_loader_input		
	"The LogMiner Unload File" on page 90	Define to specify the input file for *static* Loading or *continuous* Loading.
jcc_logminer_loader_key		
	"Applying the License Key" on page 54	Define to the license key provided by JCC. REQUIRED to run the Loader.
jcc_logminer_loader_lock_threshold		
	"Locking Threshold for Large Transactions" on page 404	Define to change the default setting.
jcc_logminer_loader_name		
	"Keyword: Loadername" on page 254, and elsewhere.	Define to specify the Loader name or use the Keyword Loadername.
jcc_logminer_loader_rdb_version		
	"SQL Interface and Rdb Targets" on page 399	Available for backward compatibility.
jcc_logminer_loader_retry_delay		
	"Retry Delay" on page 434	Set for the delay before retrying following an exception.
jcc_logminer_loader_stale_interval		
	"Commit Interval" on page 388	Set as an alternative to the input_failure keyword to specify how long to wait to read data before a thread's checkpoint record is marked as stale.
jcc_logminer_loader_stat_csv_date		
	"Date Format in the CSV Output" on page 343	Define as the OpenVMS date format to use with CSV and T4 statistics output.
jcc_logminer_loader_stat_file_seconds		
	"User Control for Flushing Statistics" on page 352	Use to modify how often the statistics output is flushed to a file.
jcc_logminer_loader_stat_interval		
	"Deltas or Cumulative Statistics" on page 359	Define to cause the Loader threads to periodically display activity.

Logical Name	Reference Page	Use and Comment
jcc_logminer_loader_stat_options		
	"Statistics Options" on page 352	Use to define several options that are relevant to CSV statistics.
jcc_logminer_loader_stat_output_dir		
	"Statistics Options" on page 352	Use to change the default output directory for CSV or T4 statistics.
jcc_logminer_loader_stat_output_file		
	"File Naming" on page 354	Use to change the name of the statistics file, if you wish.
jcc_logminer_loader_stat_tardy_field		
	"Choosing the Tardiness Indicator" on page 458	Use to for additional control of warnings about tardiness.
jcc_logminer_loader_stat_type		
	"Table Activity Reflected in the Log" on page 360	Use to set whether the Loader displays statistics as deltas from the last reporting interval or as cumulative numbers.
jcc_logminer_loader_stat_wait_seconds		
	"Display and Scroll" on page 316	Use to set how long to wait to find a running Loader session to monitor before exiting.
jcc_logminer_loader_stat_file_seconds		
	"User Control for Flushing Statistics" on page 352	Use to set how frequently (in seconds) to flush statistics output buffers to disk. Default is 600.
jcc_logminer_loader_throttle		
	"Re-tries and Exceptions" on page 464	Use to choose a throttle style if you want CLML to run at something other than "near realtime."
jcc_logminer_loader_throttle_realtime_percentage		
	"Realtime Throttle Percentage" on page 482	Use to choose how much faster than realtime you want changes applied to the target in a test environment using 'COPY' mode.
jcc_logminer_metadata_file		
	"Setting the Mode" on page 88	Use with static mode (both the original and the new, improved version) and to create files for 'COPY' mode.
jcc_logminer_mode		
	"Setting the Mode" on page 88	Use to set mode to static, continuous, or copy mode.
jcc_logminer_output_file		
	"Setting the Mode" on page 88	Use to when creating file for testing using 'COPY' mode.

Logical Name	Reference Page	Use and Comment
jcc_tool_... api, batch, com, data, dp, exe, java, local, root, share, source, sql		
	Installation chapter beginning on page 47	These logical names (along with the two to follow) are defined by the startup procedure (for both the standard and any multi-version Loaders). It is possible to change the definition to relocate files. Doing so is not recommended, except in the case of the _java_lib and _logs described separately.
jcc_tool_examples		
	For use with your own or JCC's testing routines.	Define to JCC_TOOL_ROOT:[EXAMPLES.<your choice>] where <your choice> is API, JDBC, ORA, RDB, or TUX.
jcc_tool_java_lib		
	"<attribute> for classpath" on page 252	JDBC specific. Define to the directory used to support the JDBC target.
jcc_tool_logs		
	"Choosing the Disk" on page 51	Define in the process context to redirect the log files.
jcclml_installed_version and jcclml_link_datetime		
	not intended for user changes	defined by the Loader statup procedure in the Loader-version-specific logical name table to contain the values of the Loader version and the link datetime.

Thread Details for the Statistics Monitor

The statistics screens, particularly the S (State) report type, display abundant information in a small space. Some interpretation can help. The Statistics Monitor is discussed in "Monitoring an Ongoing Loader Operation" on page 313. Many of the reports use a single character code to report on the status of each thread. See Figure 2, "Thread State Codes," on page 321.

The S (State) report type includes longer and, hopefully, more meaningful descriptions of the thread states.

Some of these descriptions are specific to a target type. The chart begins with the ones that are appropriate to all target types and continues for specific target types. (The chart includes displays that will only be seen with the separately licensed Kafka Option.)

Note that the items to be displayed include information in angle brackets that will be replaced on the screen with the appropriate variable information. For example, <target type> means that the display will show the target type, <checkpoint type> means that the display will show the checkpoint type, <#of records> means that the display will show the number of records, etc.

Each line also includes latency information as the first column. The chart to follow includes some aids to understanding the latency data.

Color Coding

Light red in the Displayed column identifies items that are likely to be transient because that portion of the operation should be brief. An example is Open Input.

Light blue in the meaning column identifies items that are part of the Input phase.

Light yellow in the meaning column identifies items that are part of the Output phase.

Displayed	Meaning
LML	Transient state between input and output. Latency shown is since the thread is started.
Initialize	Executing initialization; reading Control Files, mapping global sections, etc.
Open Input	Opening the data input stream - either the CLM mailbox or the file containing data that was previously processed by the LogMiner.
Input	In the input phase and transitioning from one sub-phase to another. Latency shown is since the time the thread entered the input state.
Input->Read MBX	Has the mailbox (MBX) lock and is either in overhead processing or actively processing data from the LogMiner.
Input->MBX wait	Waiting for another thread to release the mailbox (MBX) lock.
Input->MBX release	Releasing mailbox (MBX) lock.

Displayed	Meaning
Input->CLM wait	Waiting for data to read from the CLM mailbox.
Input->Verify record	Verifying a record read from the CLM mailbox.
Input->Buffer data	Adding the verified record to the data buffers.
Output	In the output phase and transitioning from one sub-phase to another. Latency shown is since the time the thread entered the output state.
Create Output	Creating the output stream to the target.
Output->Synch wait	Waiting to synchronize locks. Options are waiting for a lock on one or more originating dbkey (or row) locks or waiting for a lock at the locking level declared by JCC_LML_LOCKING_LEVEL logical name.
Output->Write wait	Queueing a request for the write lock.
Output->Write release	Releasing the write lock.
Output->Process buffer	Processing the buffered records. Latency shown is frrom the time that all necessary locks were received and processing started.
Output->Protected write wait	Holding a protected write lock and waiting to write until all other threads complete their writes.
Output->Concurrent write wait	Waiting for other threads seeking protected access to the target.
Output->Realtime throttle wait	Waiting as instructed by the JCC_LOGMINER_LOADER_THROTTLE and JCC_LOGMINER_LOADER_THROTTLE_REALTIME_PERCENTAGE which can be set by the Administrator.
Output->Record	Preparing a record to be written, including FilterMap and MapResult processing and target specific formatting.
Output->Message	Preparing to write to the target. Whether what is written is a record or a message depends on the definition of the <message contents> parameter of the Output keyword.
Output->Send message	Finding the target routines to write.
Filter&Result	Applying a filter or modifying the output as directed by MapResult keyword.
Checkpoint	Determining the routine appropriate to write checkpoint data.
VA->Create	Writing a virtual array.
VA->Sort	Sorting a virtual array.
VA->Append	Appending one virtual array to another.
VA->Position	Re-positioning the current record in a virtual array.

Displayed	Meaning
VA->Read	Reading from a virtual array.
VA->Write	Writing to a virtual array.
Open-><target type>	Calling into the target specific shareable image to open the output stream.
Write-><target type>	Calling into the target specific shareable image to write to the output stream.
SetTrans-><target type>	Calling into the target specific shareable image to set a transaction on the output stream.
Rollback-><target type>	Calling into the target specific shareable image to rollback.
Commit-><target type>	Calling into the target specific shareable image to commit.
Close-><target type>	Calling into the target specific shareable image to close the output stream.
WriteChkpt-><checkpoint type>	Calling into the target specific shareable image to write checkpoint data to the checkpoint stream.
ReadChkpt-><checkpoint type>	Calling into the target specific shareable image to read checkpoint data from the checkpoint stream.
Java (for JDBC or Kafka)	
CreateJavaVM	Creating the Java Virtual Machine (JVM).
GetJavaEnv	Retrieving the Java Native Interface (JNI) to handle the created JVM.
AttachCurrentThread	Connecting the threads JNI to the JVM.
JDBC	
<jdbc_connect_string>	Using the connect string specified in keyword JDBC~connect~<string name> to connect via the JVM to the target.
<table>[batch:<#of records>]	Outputing the number of records indicated to the table indicated.
Commit[rows:<#of records>]	Commiting the number of records indicated.
Rollback[rows:<#of records>]	Rolling back a transaction with the number of records indicated.
closeConnect	Closing the connection to the target.
Rdb	
Rdb->Connect	Executing the Rdb-specific SQL to attach to the database.
Rdb->Disconnect	Executing the Rdb-specific SQL to disconnect from the database.
Rdb->Prepare	Preparing an SQL statement for use with the target.

Displayed	Meaning
<table>	Executing Rdb-specific SQL statement to replicate the specified table to the target.
Rdb->Commit	Committing the transaction to the database.
Rdb->Rollback	Rolling back the transaction.
Rdb->SetTransaction	Setting a transaction.
Rdb->WriteCheckpoint	Writing a checkpoint to the target.
Rdb->ReadCheckpoint	Reading a checkpoint.
OCI (Oracle)	
OCI->Connect	Executing the Oracle-specific SQL to attach to the database.
OCI->Disconnect	Executing the Oracle-specific SQL to disconnect from the database.
OCI->ValidateMetadata	Validating the metadata within the Oracle target.
<table>	Executing Oracle-specific SQL statement to replicate the specified table to the target.
OCI->commit	Committing the transaction to the database.
OCI->Rollback	Rolling back the transaction.
OCI->SetTransaction	Setting a transaction.
OCI->ReadCheckpoint	Reading a checkpoint.
OCI->DeleteCheckpoint	Removing unnecessary checkpoint data.
OCI->InsertCheckpoint	Inserting checkpoint data.
OCI->UpdateCheckpoint	Updating checkpoint data.
Tuxedo	
tpalloc(TPINIT)	Allocating memory for the Tuxedo connection.
tpalloc(FML32)	Allocating memory for a data buffer.
tpfree	Freeing memory that is no longer needed.
tpchkauth	Checking authorization requirements.
tpenqueue:<qspace>:<table>:[buffer:<#of records>]	Enqueueing a data buffer with the number of rows, qspace, and table shown. Display is tpenqueue:<qspace>:<table>:[buffer:<#of records>] with no wrap.
tpcall:<table>:[buffer:<#of records>]	Calling a Tuxedo application routine for the table, data buffer, and number of rows shown.

Displayed	Meaning
tpcall:<table>[<buffer#>][buffer:<#of records>]	Asynchronously calling a Tuxedo application routine for the specified table with a data buffer of the specified number of rows. The buffer# is the count of buffers previously sent within the context of the current commit interval.
tpgetrply:<table>[<buffer#>][buffer:<#of records>]	Asking the Tuxedo application for the status of the buffer for the table, buffer, and number of rows shown.
tpcancel	Cancelling an asynchronous call to a Tuxedo application.
tpbegin	Starting a transaction.
tpcommit	Committing a transaction.
tpabort	Aborting a transaction.
tpterm	Disconnecting from a Tuxedo application.
Kafka (only available with a license for the JCC LML Kafka Option)	
newKafkaProducer	Creating a new Kafka Producer connection.
initTransactions	Initializing the Kafka transaction manager.
beginTransaction	Beginning a transaction.
send(<topic>(<recordID>))	Sending the record shown to the topic shown.
flushBuffers	Flushing buffers.
<topic>(<recordID>).get()	Asking for acknowledgement that the buffer has been received by the topic in order to free some memory when memory resources are low for the JVM.
abortTransaction	Requesting an abort.
commitTransaction[recs:<#of records>]	Requesting a commit. Number of records shows how many rows were written to the target in this commit interval.
closeConnect	Requesting closing the connection.

Support Desk

A license - whether temporary or permanent - to the JCC LogMiner Loader comes with directions for how to reach Loader support. Loader support queries are addressed by experts who also have experience with a wide range of the companion products.

Blogs

Blogs provide additional information, updates, and insights.

See what's available at http://www.jcc.com/lml-blog

Frequently Asked Questions

1. **Question:** How do I get started?
   Answer: This is a hard question ... because there are so many options and so many different applications of the Loader technology. However, temporary licenses are available and Loader support will help you as you learn about the product. Alternately, JCC can provide a Consultant or Consultants to discuss your issues and goals and help you develop your architecture or complete your installation or application.
2. **Question:** Why do I need the Loader from JCC, if I have the LogMiner from Oracle?
   Answer: Oracle's Rdb LogMiner and JCC's LogMiner Loader are designed and developed to work together. LogMiner creates binary files; the Loader supports your definition of how they should be applied. The Loader also provides a great many tools to make your work more effective.
3. **Question:** Do I need to install Oracle before I can use the Loader?
   Answer: If you are using the Loader OCI interface to update an Oracle target, you will need to install the Oracle client software on your OpenVMS system. Otherwise, your Loader use will not require the Oracle OCI client.
   Similarly, if you are using a JDBC target and/or, ultimately, writing to an Oracle, SQL Server or other database, there will be some support tools and files to install.
   Consider the installation chapter and the chapter that is specific to the target that you are using.
4. **Question:** How do I know that the AIJs are applied in the correct order?
   Answer: You don't, unless you are running the Continuous LogMiner. The Continuous LogMiner will process through backed up AIJs in the correct order. The Loader will direct the LogMiner to start at the point where your session was previously interrupted.
5. **Question:** Do I have to be attached to the source database to Load the target?
   Answer: If you are running the Continuous LogMiner, it must, of course, be attached to the database. If you are running CLM and have also set the Loader to use the heartbeat mechanism, the Loader parent process for the Loader and LogMiner will attach to the database. If you are running the *static* LogMiner and Loader, the LogMiner will need a definition of the metadata, but will not have to run attached to the source database.
6. **Question:** What versions of system software are required to run the Loader?
   Answer: The answer has varied over time. Please see the blogs for the latest

information. Regression testing of the Loader begins on new Rdb and on new OpenVMS releases as soon as they are available to us.

7. **Question:** Is training available on LogMiner and LogMiner Loader?
   Answer: Yes. JCC provides a two-day seminar on LogMiner, LogMiner Loader, and how to use them. JCC also offers the material in a workshop format that can illustrate all of the steps necessary to create a functioning application. JCC also offers a variety of other Rdb and application seminars, both in our training facilities and as on-site seminars. For more information, see **http://www.jcc.com/services/training**

8. **Question:** Is support available?
   Answer: Yes JCC offers various levels of support for its products.

   Each license comes with the first year of standard maintenance. Standard maintenance includes the right to new releases during the maintenance period and support during JCC business hours. GOLD support extends the support period to 24 X 7 X 365. Both levels of support are renewable for subsequent years.

   Most support questions are addressed in email. Email support provides coverage without worrying whether an individual might be on an airplane or otherwise available. It also provides a history of items important in your environment and offers the option of responses from more than one person.

   The people responding to your support questions are highly versed in the Loader, the companion products, and the issues of architectures that use the Loader.

   For the best results, you may want to include:

   - Your contact information
   - Rdb version (Use "`$ rmu/show version <db root>`".)
   - LML version (Use "`$ jcc_version`".)
   - VMS version (Use "`$ show system/noprocess`".)
   - As much detail as possible.
   - If the answer is not easy, you will, likely, be asked for your the LogMiner options file, the Loader Control File, and the following logs.[1]
     - Log file that contains the jcc_continuous_logminer_loader for <loader-name>
     - JCC_TOOL_LOGS:JCC_RUN_CLM-<loadername>.LOG

1. Note that adding logging~initialization as the first line of your Control File after Loader-name ensures that the Control File is echoed in the log.

- JCC_TOOL_LOGS:JCC_RUN_LML-<loadername>.LOG

JCC business hours are 8:00 AM to 5:00 PM US Eastern Time Zone.

To learn more about standard and gold support or if you have questions during testing, send e-mail to **JCC_LMLoader@JCC.com**. When you have a license and have chosen a support level, you will receive a different e-mail address to use, but the team of support experts will remain the same.

9. **Questions:** Is help available?

   Answer: In addition to help through e-mail and standard and gold support, JCC also offers on-site consulting. JCC consulting can assist you in configuring the LogMiner Loader or in other Rdb DBA and performance tuning areas, as well as in application development or technology planning. For information on consulting rates and availability, contact JCC at **office-administration@JCC.com** or use any of the contact information provided in “Contact Information” on page 3.

10. **Question:** What if tables A, B, and Q are being loaded with LML and the metadata needs to be changed in table C?

    Answer: If the metadata change does not affect the tables that you are writing to the target, you do not need to worry.

11. **Question:** If I have two databases with the same tables and columns, can I use the same metadata definition?

    Answer: Maybe not. The Loader works at a level that has more to do with the physical placement of the data than what we think about when we work with the logical model. If the columns were defined in a different order or, even, if the metadata version is different, you will need different metadata definitions for the Loader.

12. **Question:** What if the values in the primary key columns change?

    Answer: If you wish to replicate the source data to a target and the values can change for columns that are part of the primary key in the source, you will not be able to use the primary key to identify the row in the target database. However, you can use a dbkey mechanism that has been developed for this purpose. Alternately, you can provide an automatic identity column in the source and define it as the key for the target.

13. **Question:** What about constraints?
    Answer: Constraints have already fired in the source database and, therefore, rows have been checked. If you have constraints on the target database, you may have circumstances that are valid, but for which the constraints fail. For example, if detail records are inserted before a parent record, you will have a

constraint failure on the foreign key in the detail. Constraints are not appropriate for the target tables.

14. **Question:** What if you roll past midnight?
    Answer: If the real question here is how do I write to different targets depending on the day, you can use the filter keyword on a column that is a relevant timestamp or on a materialized column.
15. **Question:** What is the right value to use for retry frequencies for the customer defined API?
    Answer: Some of these parameters are best set after you have experience with your data and your situation. The defaults are carefully chosen and should not be changed without reason.
16. **Question:** Can we use wildcards in our file specifications?
    Answer: No … not in most cases. The logical name JCC_aij_backup_spec accepts wildcards, but everything else requires a file specification without wildcards.
17. **Question:** Can I specify a date format that shows more digits of precision than hundredths of a second?
    Answer: On Alpha systems, dates are meaningful to one thousandths of a second. For the customer supplied API, you can request that fractional seconds be extended beyond what is meaningful. You may request up to seven digits beyond the decimal point, but only the first three have any meaning.
18. **Question:** What do you mean, the Loader knows whether to update or insert?
    Answer: Assuming that you have asked for both update and insert, the Loader will distinguish whether the data passed in the LogMiner unload is an update or insert. The SQL generated by the Loader examines the data in the target table. If the row exists, it updates; otherwise, it inserts.
19. **Question:** Can you slave a table where all columns participate in the primary key?
    Answer: Beginning with Version 2.0, you can, *providing the key values never change*. In this case, you need to use the table actions “insert,noupdate,delete”.
20. **Question:** Why can’t you use rollup and dbkey?
    Answer: For tables that are included in the Control File for rollup, rather than replication, rows are not deleted from the target even if they are deleted from the source. Dbkeys can be reused. This combination would then cause rows to be overwritten.
21. **Question:** Are ranges supported for the value in the filter?
    Answer: Ranges are not supported with the keyword Filter, but the keyword FilterMap makes ranges possible. In fact, FilterMap supports any SQL restriction that only operates on a single row.

22. **Question:** What does the sort have to do with API?
    Answer: The sort orders rows in the output document in the same order as they would be ordered with a database target. This permits downstream processes to operate correctly on their targets. Note, however, that the Loader does not require that the records be sorted. You can leave the sort keyword out of the Control File.
23. **Question:** What logging do you recommend?
    Answer: This depends on what you are trying to do. Generally, logging slows down the LogMiner/Loader process. Also, if you enable extensive logging the log files will become quite large. If either or both of these are an issue in your situation, you may want to use minimal statistics. However, if you are testing or if you are tracking a problem, more extensive statistics can be important. JCC does recommend/request that you include logging~initialization very near the beginning of the Control File so that the Control File itself is included in the log.
24. **Question:** Is dynamic change possible for logging?
    Answer: At this time, it is not possible to change the Control File while it is running a Loader session.
25. **Question:** How do I get started?
    Answer: This is a hard question ... because there are so many options and so many different applications of the Loader technology. However, temporary licenses are available and Loader support will help you as you learn about the product. Alternately, JCC can provide a Consultant or Consultants to discuss your issues and goals and help you develop your architecture or complete your installation or application.
26. **Question:** With the Continuous LogMiner and Loader, can you backup active AIJs while you are trying to catch up?

    Answer: No. You cannot remove data that has yet to be “seen” by the LogMiner ... without getting an exception message. If the backup occurs, you will need to restart the Loader.
27. **Question:** What account does the CLML program run in?

    Answer: Whatever account you start it under.
28. **Question:** What happens when we go to Daylight Savings Time?

    Answer: The LogMiner extracts records from the AIJ in the same order as they were committed by the application. When the system clock switches, the commit timestamp correspondingly switches. If the Loader [1] is to materialize the

1. You would use the Control File (keyword VirtualColumn) for this.

commit timestamp in the output tables, then the commit timestamp will reflect the new time in the source system.

The Loader does not use timestamps to record position within the AIJ file. Rather it uses the AERCP and the TSN of the last transaction committed to the target to maintain its highwater context. The Loader is, therefore, insulated from the time changes.

29. **Question:** What does it mean if I get the message “incorrect AIJ file sequence!UL when !UL was expected”?

    Answer: If CLM is processing AIJ backup files, it will complete the backups and move to the active file. If the active file is not the one expected CLM cannot continue. If you see this message, find out if someone backed up the AIJ while CLM was catching up. See “Automated AIJ Backups” on page 483 for more discussion.

30. **Question:** What happens if there is no high-water data?
    Answer: The checkpoint code will return a status that a new file (or database record) is being created and will request operator approval. (It is also possible to set a logical name to provide the operator response automatically.

31. **Question:** For me, the parser is not capturing the first 10 records in the CSV output. Why?
    Answer: The parser only captures the 'current' data. The 'current' data is the difference between the last display and the current. For the first, there is no previous.

32. **Question:** I am receiving messages that may indicate issues with memory when the Loader should be replicating a transaction with 6.4 million rows. Why? What can I do about it?

    ```
    %comc_va_write:Unable to allocate memory for buffer
    %dba_buffer_inpute:unable to write VA for modify buf-
    fer.
    %COMC-E-NO_MEMORY, Unable to allocate memory for the
    firtual array.
    %dba_buffer_input::Buffer size is 906033
    ```

    Answer: The Loader buffers no less than an entire transaction in memory before attempting to replicate it. Your 6.4 million rows will require a reasonable amount of memory, but the Loader has been used to replicate significantly more. There are configuration changes that you can make to accommodate the memory demands of large transactions:
    a. Enable 64-bit memory in the Loader (See “Analyzing Performance” on page 408.)
    b. Increase the pagefile quota for the account running the Loader.

33. **Question:** Why would I want to use a logical name to define the loadername?
Answer: There may be many reasons. One example is that it helps if you have several databases that are similar, but not exactly the same, for which you want to use the Loader. You can use a top level Control File that is the same for each of the databases and have an outer procedure define logical names for the targets and the database metadata and the loadername.

34. **Question:** What happens if a keyword occurs more than once in a Control File?

Answer: For some keywords, such as table, column, primary key, the keyword is likely to occur multiple times. For keywords that are expected only once, the last definition of the keyword is the one that will be used. Careful organization of the Control File(s) helps to prevent unwanted results. See "Referencing Other Control Files" on page 219.

35. **Question:** Do I have to have the same UIC as the process running the Loader to run the statistics?

Answer: No, the JCC LogMiner Loader creates system global, pagefile-backed section files.

36. **Question:** What happens when a table is truncated?

Answer: The LogMiner does not yet support truncate. Therefore, the Loader does not.

37. **Question:** I have discovered that Oracle does not take column names that are as long as some of the ones that I have used in Rdb. Does the Loader offer any help?

Answer: Yes. The Control File syntax for defining a column (or a primary key) is

```
COLUMN|PRIMARY KEY~<table name>~<column name>
[<output_column_name>]~ ...
```

Notice that output_column_name is an optional parameter that can be used to name the column differently in the target database. There is similar syntax for tables.

Also, tables and columns may be named differently in the target and mapping may be indicated with the keywords MapColumn and MapTable.

38. **Question:** Control? What's this mean for the Loader?

Answer: For the Loader, there is a Control File that you use to define the metadata and to direct the Loader. There is also a *control process*. The control process is the parent process that runs the Continuous LogMiner and the Loader.

39. **Question:** Do the Continuous LogMiner and the Loader work in realtime?

Answer: We prefer to use the term "near realtime." The Loader can never get ahead. That is, if it hasn't happened in the source database, the Loader can't produce it for the target. Further, the LogMiner does not supply anything to the Loader until the transaction is committed. Since some transactions involve many updates, the Loader may, then, take a few moments to get the target in sync.

For example, in one environment, the Loader is running less than two tenths of a second behind in replicating, to many targets, a source database that commits 500 transactions per second.

40. **Question:** How do I acquire a license to the Loader?

    Answer: JCC provides permanent licenses at a license fee that is dependent on how many CPU cores access the source databases that you wish to use. JCC also provides short-term temporary licenses for testing concepts. To acquire either, send mail to **INFO@JCC.com** or use any of the contact information listed in "Contact Information" on page 3. Providing a description of your system and your issues and goals will be appreciated.

41. **Question:** I got the following as part of a Fatal Exception on a table. What does it mean?

    ```
    %dba_put_rdb_output. Fatal Exception on table customer 17-May-2002
    07:56:22 21ED249E LML TFY DCA_CL %SQL-F-FLDNOTCRS, Column ORIGINAT-
    ING_DBKEY was not found in the tables in current scope
    ```

    Answer: The Loader tried to delete a record in the customer table. The SQL that it generated relied on the column ORIGINATING_DBKEY. There are three ways that the Control File can cause the ORIGINATING_DBKEY column to be used:

    - Specify "ORIGINATING_DBKEY" in the "Table" command for the table.
    - Do not define any column of set of columns in the table as part of the "PRIMARY KEY"
    - Specify the materialized column "ORIGINATING_DBKEY" as part of the table, using the VirtualColumn keyword.

    If originating_dbkey is a surprise column for you, check whether you have a primary key defined.

42. **Question:** Okay. I added the originating_dbkey columns that we discussed and restarted it. Is it supposed to run like a dog?

    Answer: No. Did you add the indexes on the new column(s)?

43. **Question:** What is the impact of adding two more AIJ files to my source database?

Answer: Adding more journals should be completely transparent. (This assumes that you are not dropping journals and, then, re-adding them.)

44. **Question:** Does changing index types from hash to sorted in the source database make any difference to the LogMiner or the Loader?

    Answer: The Loader doesn't care how the queries are solved.

45. **Question:** During an upgrade of Rdb is it correct that I need to restart LogMiner in the live AIJ when we bring the database back up?

    Answer: Yes. The format of the AIJ may change between versions. It is wise to do an AIJ backup before the upgrade and it is very important to drain all the data out of the AIJs before the upgrade. See "Upgrades and Changes" on page 449 for further discussion.

46. **Question:** How good is the LogMiner to catch up after a longer disconnect between the source and target databases? Will consistency be guaranteed?

    Answer: The speed of catching up after a disconnect is going to depend on a number of factors, such as the speed of the I/O subsystem, CPU, ability of the target to absorb the data, etc.

    At one site, a restart that processed 5.5 million blocks of backed up AIJ (166 files) caught up in 6 minutes. This was a fairly simple case in that only a couple of the tables are being processed by LogMiner and there were only about a million changes that needed to be applied to the target. The source was on an 8-cpu gs140 (700mhz cpus) with a high performance SAN for I/O. The target was a large Tandem configuration (via Tuxedo).

47. **Question:** Do I understand correctly that I can have more than one target?
    Answer: You can run more than one Loader on the same source and each Loader can have a distinct target. Doing it this way, for example, one target can be Oracle, one Rdb. Alternately, using maptable and other keywords, you can map a given source table to multiple target tables in the same database. See "Schema and Data Transforms" on page 489 for additional discussion.

48. **Question:** We currently run SQL from Oracle to the Rdb database through SQL*net. We can also connect to the Rdb database from SQLplus. What else do we need to install on the OpenVMS system to support the use of the Oracle target?
    Answer: The issue is direction of the data flow. Rdb implements an Rdb specific of the SQL*net software that enables OCI calls to an Rdb database. To talk to an Oracle database, you need a different Oracle client package that talks to the Oracle databases.

49. **Question:** I tried to start a session under the new release, but got a message about invalid license.

Answer: When you are using the MV support to support an older version as the standard version and have a new version too, the license logical will be defined in logical name tables specific to each version. You should 1)Copy the jcc_lml_license.com file from the version-specific jcc_tool_com: directory into your jcc_tool_local: directory and 2) Edit the jcc_tool_local: version to either add the license number or comment out the definition altogether and use $ deassign/table=<version specific name table> jcc_logminer_loader_key

50. **Question:** When we change the system clock while the Loader is running are there any concerns.
Answer: The processing logic in the Rdb LogMiner and in the JCC LogMiner Loader is not based on timestamps. There is no issue there. However, the processing rate calculations in JCC_LML_STATISTICS may be inaccurate.

51. **Question:** I am running the JCC CLML between two data centers [on opposite sides of a continent] ... When I set the checkpoint interval to 1, I can never catch up to the source DB. If I set the checkpoint interval to 100 or 200, it seems to keep pace well. ... I am wondering why I would get so much better throughput with a larger checkpoint? Is it buffering that is the difference? I noticed that the latency detail shows that 99% of the time is spent in the target. ... The target shows almost no activity at all. The target is Oracle 10G R2. [Ping time between the sites is 77 ms.]
Answer: There are several issues. There is latency involved in a commit and there is latency each time the Loader SQL requests must be 'compiled' in Oracle. Each of these incur latency in the target Oracle database as well as latency for the network requests. The latency in the Oracle commit requires a significant amount of time for a small transaction, but a smaller percentage of the total time for larger transactions. In addition, the Oracle SQL interface allows caching exactly one compiled request at a time. There is one request per table that the Loader is processing. A transaction commit invalidates the cached compiled request. Compiling a new SQL request requires two messages between the client and server (one round trip). Assuming that the same tables are used in subsequent transactions, increasing the commit interval increases the chance of being able to use a cached SQL request and, therefore, reduces the number of network messages. In addition, executing a request for an input record requires a network round trip.
With 77 ms round trip ping time and the 1.07 rows per transaction shown in the logs, if we assume two tables, we can calculate 1.07 * 77 ms to compile the request, 1.07 * 77 ms to execute the request (update the data), 1.00 * 77 ms for the commit, yielding an average of 242 ms of network delay per source transaction, plus whatever time it takes to compile execute, and commit on the target. For 100 transactions, this would be 24,200. With a commit interval of 100, this

becomes 2 * 77 ms to compile the request, 107 * 77 ms to execute the request, 1 * 77 ms for the commit, yielding an estimate of 8,470 ms per 100 source transactions. That's almost three times the throughput.

52. Question: What do you need to ask?

C

D

E

F

G

H

I

J

K

L

M

N

Q

R

S

T

U

V

X