amdocs billingplatform

Amdocs Billing Platform

System Performance & Tuning Guide (Internal)

For Internal Use Only

amdocs

Document Information

Software Version:	**6.0 SP8**
Publication Date:	**November 2006**
Catalog Number:	**212681**

Contents

1. Introduction 1
Scope of this Document 1
NFT Technical Architecture 2
Terms and Definitions 2

2. Batch Applications 5
Acquisition & Formatting 5
Database Structure 5
Dropped Calls 5
Duplicate Check – Optimal Buffer Size 5
Date Value and GD1_TEST_MODE 5
MF1_DUPCHECK_KEYS 5
OP_MODE Parameter 6
TimesTen Datastore Sizes 6
Billing 6
Dummy Partitioned Table 6
Billing Process Configurations 6
BTL Specific Configuration 7
Logger and Tracer Configuration 8
The Ratio between Bill Preparation and Extract Groups 10
Table Partitioning 10
AMC 12
Dispatcher 12
Output Directory Size 12
Output File – Number of Records 13
Event Extract 13
External Record 13
Extract Method 13
Rater 14
Rater Architecture 14
General Flow 14
Transaction Length 14
Recovery Mode 14
Transaction Manager 14
Database Connection 16
Implementation Factor 16
Performance Indicator Locks 16
Optimizer 16
Database Tables Partition Number 16

Signal Handling....17
TimesTen Datastore Sizes....17
Multi-Process Rater....18
Rater Memory Allocation Patterns....18
Re-Rating Extract....18
Transaction Broker....18
Compress Indicator....18
Database Structure....19
Tables Partitions....19
Grants and Permissions....19
Parameters....19
Audit & Control....19
Update Handler....20
Acquisition & Formatting – Extract Phase....20
Acquisition & Formatting – Loading Phase....20
Number of Threads....20
XLA Daemon....20
Output File – Number of Records....20
Account Receivables....21
Parameters....21
Collection....21
Business Process Automation Step Management....21
Miscellaneous....22
UTL1_FILE_LOADER....22

3. Online Applications....23

APInvoker....23
Customer Management....23
WebLogic Servers Setup....23
UAMS system Cache Settings....27
EJB Quantities....27
Application Server Kernel Parameters....29
Table Partitioning....34
UAMS....35
Usage Query....36
Application Server Changes....36
Batch Server Changes....36
Publisher Applications....37

4. External Tools....39

AMC....39
Oracle....39
TimesTen....42

5. Hardware Configurations....47

Disk Setup....47

LINUX Suse SLES9 OS 47

6. Memory Allocation Patterns 49
Memory Allocation Problem 49
Proposed Solution 49

Appendix A. Recommended Table Partitioning 51

1. INTRODUCTION

The Non-Functional Test (NFT) lab is a Production-like testing environment that focuses on the core implementation of the Enabler product. NFT simulates real Production from various aspects: architecture, volume, data variety and load.

One of NFT's main goals is shortening the way to an optimal Enabler Production setup.

Scope of this Document

This document provides guidelines for the optimal Enabler configuration based on the results of past NFT runs.

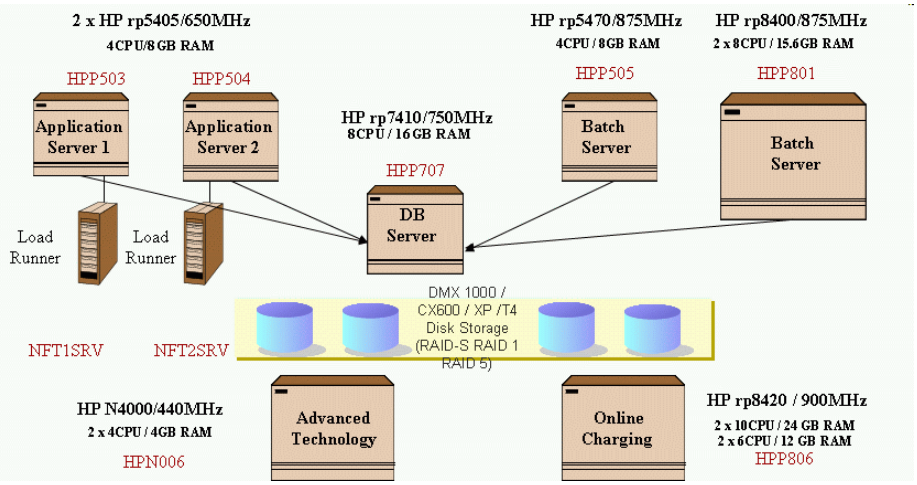
The document contains six chapters:

- Chapter 1, *Introduction* – NFT environment overview and general terms.
- Chapter 2, *Batch Applications* – Guidelines for configuring batch applications (Acquisition & Formatting, Billing, Transaction Broker).
- Chapter 3, *Online Applications* – Guidelines for configuring online applications (Customer Management).
- Chapter 4, *External Tools* – Guidelines for configuring third-party (Oracle, TimesTen) and in-house tools (AMC) for Production-size environments.
- Chapter 5, *Hardware Configurations* – Guidelines for platform-oriented configuration that will ensure a better operation of the applications and tools.
- Chapter 6, *Development Best Practice* – Guidelines for more effective code from the view of performance, including general tips that can improve the overall performance of the applications.

NFT Technical Architecture

The diagram below depicts the current Non-Functional Test (NFT) system architecture.

Figure 1-1: NFT Technical Architecture



Terms and Definitions

The following terms and acronyms are used in this document.

Term	Definition
Audit & Control	Audit & Control is an Amdocs module used to transfer files and control files between components. It includes both a fully integrated monitor of daily event-processing activities and APIs for interfacing with other Amdocs components. Audit & Control monitors all sites, controls the file flow between the sites and supplies auditing information about the files.
Acquisition & Formatting	An Amdocs module whose function is to collect the incoming event records, format them for use by other Amdocs components (such as: Rating and Customer Management), and guide them to the appropriate customer.
AR	An Amdocs module whose function is to manage Account Receivables.
AMC	Application Monitoring & Control. The component responsible for monitoring and controlling Amdocs applications.
APInvoker	The APInvoker is a tool whose chief purpose is to facilitate balancing the Transaction Broker's load and to expedite processing transactions by parsing the XML and invoking EJB or local transactions.
BPA	Business Process Automation

Term	Definition
CC	CC (for Configuration Control) is a tool, developed by Amdocs, used for coordinating the software versions between those on site and those of the Amdocs Development Center.
CLOB	A Character LOB – see LOB.
Customer Management	Part of Enabler, featuring a progressive customer model that lays the foundation for the unique benefits of the Customer Management Entity.
FTC	File Transfer Control The name assigned to the Audit & Control new architecture. This includes the main process that runs on one of the network machines with various threads supporting the services required by the Audit & Control system.
IMSI	International Mobile Subscriber Identity – A unique identification code for each GSM subscriber.
IOT	Index Organized Table
JMS Adaptor	The JMS Adapter enables the Transaction Broker to connect to the JMS server and use it to receive and send messages. In the Enabler ClarifyCRM integrated environment, the JMS Adapter is used to receive and send transactions from and to ClarifyCRM in an asynchronous mode.
LOB	Large Object. An SQL `LOB` is a built-in type that stores a Large Object as a column value in a row of a database table. The driver implements a `LOB` object using an SQL `locator (LOB)`, which means that a `LOB` contains a logical pointer to the SQL `LOB` data rather than the data itself. A `LOB` is valid for the duration of the transaction in which it was created.
MSISDN	Mobile System Integrated Services Digital Network – The telephone number of a GSM cell phone. The MSISDN is stored in the SIM card inside the phone.
MRO	Message Router – Responsible for the connection between AMC and Amdocs Billing Platform component APIs (intrusive mode). MRO receives data and events from the components that are monitored or controlled, and sends this information to AMC.
NFT	Non-Functional Test
PI	Performance Indicator – The PI contains information on all accumulated rated events for a subscriber or group of subscribers. Any attribute and amount of the event can be accumulated. Usually the chargeable attributes as well as the event rates are accumulated. PIs are used for retaining data in connection with allowances, discounts, and budget control schemes.

Term	Definition
PIT	Pricing Item Type – The pricing item type defines a template for the functionality of a single, independent pricing element, as well as the rules that are applied to an event, and the conditions under which the rules may be applied. These elements are set via the pricing item's role, qualification criteria, parameters, and event handlers.
POR	Pricing Order Request. The POR is an internal structure in the memory of the BTL process. POR contains all the data of an input transaction, which are available for dumping into a log for investigating the contents of the transaction.
Operational	A table-driven tool with a GUI front-end that was developed in-house by Amdocs to make batch job scheduling easier, and to provide a means for real-time monitoring of the running of batch jobs and their dependencies.
Transaction Broker	Responsible for the exchange of information among numerous functional components in the Billing system of a telecommunications company. Each Billing component is responsible for a defined set of functions, and is decoupled and self-contained. The Transaction Broker is responsible for transferring the information between these components.
TimesTen	TimesTen is a high performance relational in-memory database that supports the ODBC (Open Database Connectivity) and JDBC (Java Database Connectivity) interfaces. The product is supported on multiple platforms. It allows transaction management, full or partial replication, and also makes its logged data accessible through an open API.
UAMS	Unified Amdocs Security – An Amdocs-developed authentication and authorization module that can override the WebLogic default security realm and perform all security related activities.

2. BATCH APPLICATIONS

This chapter provides tips, recommendations and helpful remarks for configuring Enabler applications. These are based on past Non-Functional Test (NFT) findings.

Acquisition & Formatting

The following recommendations apply to the Acquisition & Formatting module.

Database Structure

For better performance the Guiding (designated: **GD**), Acquisition & Formatting (designated: **MF**) and Audit & Control (designated: **AC**) tables should exist in one instance.

Dropped Calls

The number of dropped calls has a major impact on the Acquisition & Formatting performance results. Of course, this is implementation-specific.

Duplicate Check – Optimal Buffer Size

The buffer size parameters are table-driven and defined inside the MF1_GEN_PROPERTIES reference table. Currently, the default is set to 500 KB.

After file processing, if the duplicate check process has allocated a buffer larger than the optimal size, Acquisition & Formatting de-allocates memory and sets the buffer to its optimal size (as defined by the size parameter in the table).

Date Value and GD1_TEST_MODE

Acquisition & Formatting retrieves the date value, Logical Date or System Date (the default), based on the GD1_TEST_MODE table. Therefore the date parameter should not be defined in the Production setup.

MF1_DUPCHECK_KEYS

MF1_DUPCHECK_KEYS should be defined as partitioned Index Organized Table (IOT). The IOT is partitioned by two keys: the primary key is derived from the period of the call; the secondary key is derived from the last two digits of the calling number. For details refer to the *Acquisition & Formatting - Duplicate Records Check - Implementation Guide*.

Under appropriate conditions, defining IOT eliminates an extra call that retrieves the table's value after accessing its index.

OP_MODE Parameter

The OP_MODE parameter replaces the obsolete TLG_TEST_MODE parameter. It defines the number of times the program will use the **getenv** function to retrieve environment variables. To minimize these function calls, its value should be: **P** in Production.

TimesTen Datastore Sizes

The recommended data sizes for both the Table and the Index contents in a core TimesTen Acquisition & Formatting Datastore are:

Row Size (KB)	Sizing Factor	Table Name
0.126	Customer	GD1_CUSTOMER_DETAILS
0.212	Event distribution per Subscriber	GD1_SUBSCRIBER_DEST
0.196	Resources per subscriber	GD1_SUBSCR_KEY

A customer has a minimum of one subscriber, depending on the customer type: residential, small businesses, corporate, and so forth.

A subscriber has a minimum of one event distribution.

A subscriber has at least two resources: MSISDN and IMSI.

In residential implementations, a subscriber requires about 0.7 KB.

Billing

This section presents various Billing parameters, and the tables where they are found, that are available for NFT performance tuning.

Dummy Partitioned Table

To improve performance, Billing extracts an array of sequences from a dummy partitioned table. The accessed partition is defined randomly by the application. This dummy partitioned table is defined in the Billing configuration file, bl1_config.xml, by the following parameters:

- NumberTablePartitions – Number of partitions in the dummy table.
- SequenceHelperTableName – Name of the dummy partitioned table.

Billing Process Configurations

The following Billing process configurations are defined in the BL1_CONF_SECTION_PARAMS table. In addition to the Billing parameters, the table below presents the recommended NFT values for them.

Class	Configuration	Parameter Name	Description	NFT Values
BILLING	config	BL1_RC_RATES_MAX_BUF_SIZE	Maximum array size for execute array update in BTL (writing to the RC Rates table)	4000
BLBDI	config	DbRecordsPerFlush	DB buffer size for Bill Day Initiator	N/A

Class	Configuration	Parameter Name	Description	NFT Values
WRDB	config	DbRecordsPerFlush	DB buffer size for Write To DB processes	100
WRDB	config	EntitiesPerCommit	Entities per commit	100
BILLING	config	PipeLine	Bill preparation final indication for releasing related groups	full
BILLING	config	QAOperation	Allow Collecting QA criteria	Y
BILLING	config	seq_def_buf_size	Default sequences buffer size	101
SPLTMRG	config	RERATE_PARALLEL	Number of rerate map parallel instances	8
SPLTMRG	config	RERATE_EXT_PARALLEL	Number of rerate extract parallel instances	4
SPLTMRG	config	RERATE_INTERVALS	Interval between invoking rerate maps on different usage machines	10
BILLING	config	EodTimerInterval		3
BILLING	config	PROC_SLEEP_TIME	Process sleep time when idle	3
BILLING	config	PeriodPartitionDimention	Period key database partition dimension	24
BILLING	config	PeriodPartitionFactor	Period Partition Factor	0
BILLING	config	CustomerPartitionDimention	Customer Key database partition dimension	100

BTL Specific Configuration

The following parameters are placed in the GN1_ART_SECTION_PARAM table and enable performance tuning:

Class	Section	Name	Description	NFT Values
BTL	SERVER_PARAMETERS	MIN_EH_THREAD_NUM	Minimum number of active threads	4
BTL	SERVER_PARAMETERS	MAX_EH_THREAD_NUM	Maximum number of active threads	4
BTL	AC	CUSTOMER_MODULO	Maximum groups – default 10	10

Class	Section	Name	Description	NFT Values
BTL	TRB	CUSTOMER_MODULO	Maximum groups – default 10	10

Logger and Tracer Configuration

For configuring logging and debugging the following records exist in the BL1_CONF_SECTION_PARAMS table.

Class	Configuration	Parameter Name	Description	NFT Values
BILLING	Logger	Enable	Enable logger activity using Y/N	Y
BILLING	Logger	Outputs	Facilities to which the logger write to	Simple File
BILLING	Logger	Severity	The severity of messages to be logged	50000
BILLING	Logger.output. SIMPLEFILE	Class	Logger facility - SimpleFile	BL_ Simple File Output
BILLING	Logger.output. SIMPLEFILE	MaxOpenPeriod	Maximum time a log file can be open	0
BILLING	Logger.output. SIMPLEFILE	MaxLines	Maximum number of line in log file	200000
BILLING	Logger.output. SIMPLEFILE	ImmediateFlush	Flush messages immediately or use buffer	true
BILLING	Logger.output. SIMPLEFILE	MaxFileSize	Maximum log file size	10485760
BILLING	Logger.output. SIMPLEFILE	BufferSize	Buffer size of logged messages	8192
BILLING	Logger.output. SIMPLEFILE	Filename	Mask of the file log file name	${ABP_ BL_ ROOT}/ log/%s_ %D_%T_ %t_ %n.log
BILLING	Tracer	Enable	Enable tracer activity using Y/N	Y
BILLING	Tracer	Severity	The severity of messages to be traced	50000
BILLING	Tracer	Outputs	Facilities to which the tracer write to	Simple File
BILLING	config	DebugLevel ForCHandDE	Customer Hierarchy and Discount Engine debug level: ▪ N - None ▪ E - Error	N

			▪ W - Warning ▪ P - Paranoia	

A configuration for a specific process can be overridden by adding a record where the Class equals the Process Name. For example in order to enable the tracer only for the CHGCRE process, you add the following entry:

Class	Configuration	Parameter Name	Value
BILLING	Tracer	Enable	N
CHGCRE	Tracer	Enable	Y

To disable POR serialization (flushing XML data to log) for BTL-SOR, do the following:

In **bl1_conf_section_params** set:

- param_value= '40000'
- param_name= 'Severity'

The Ratio between Bill Preparation and Extract Groups

In the full Bill Preparation run (Preparation with Extracts map), after all Bill Preparation processes have finished, the Extract map continues to work. The reason for this is that the Extract map has to work on the last extract group which includes several groups of the Preparation map.

The configuration in the core is 100 groups for Preparation and 10 groups for Extracts (a ratio of 10:1 between Preparation and Extract).

In order to improve the time of the map the ratio between Preparation and Extract should be decreased so the last group of the extracts will work on the smallest population.

Table Partitioning

Configuration

In Amdocs Billing Platform 6.0, thirteen billing tables are partitioned according to two fields: PERIOD_KEY and CUSTOMER_KEY.

The values of these two fields are calculated as the modulus of the **PeriodPartitionDimention** and **CustomerPartitionDimention** parameters, which are defined in the Billing BL1_CONF_SECTION_PARAMS table.

The main guideline for Production-like configurations is to **set the partition dimensions with high values, but physically build a small number of partitions in the database** (several hundreds, the resultant of multiplying the number of period_key partitions by the number of customer_key partitions).

This configuration adds system flexibility. If the data contains many values, they can be split and merged into many physical partitions to accommodate each account (for example, large customers or extending the history retention). This flexibility alleviates the need to migrate the data should the period dimension, which was initially defined with too small a value, has to be increased.

Ranges of Partition Dimensions

The range is defined in the Billing configuration XML. The recommendation is:

- PERIOD_KEY be calculated as modulo 120
- CUSTOMER_KEY be calculated as modulo 100

The actual values may be changed, according to account's demands.

Definition of Physical Partitions in the Database

- **PERIOD_KEY**

  The formula for calculating the PERIOD_KEY value (which is one of the partition keys) in Billing tables is as follows:

  mod ((<cycle close month> + 12*<cycle close year>),period dimension)

  PERIOD_KEY should have 24 physical partitions (plus a partition for maxvalue), each containing one period_key value, that is, 1, 2, 3, and so on. Each partition represents a cycle_month. This number can be changed according to account requirements, but should be kept relatively low.

  When the first 24 partitions are populated (after 2 years, if each period represents a month), a new partition should be added for the next period_key value (for example, the first partition to be added will be PERIOD_KEY=25).

  The aim should be to keep the number of PERIOD_KEY partitions constant, which means one of two options should be implemented:

  - The oldest PERIOD is cleaned (that is, moved out of the operating system) and a new one added at the end.
  - The oldest PERIODs (for example, 1 and 2) are merged, so that the oldest partition grows with time, retaining historical data, but the total number of partitions remains the same. In this case, too, a new partition is added at the end.

  The PERIOD_KEY function has a normalizing factor so that all accounts can build the physical partitions from 1 to a maximum number of physical period_keys), instead of calculating the values of the PERIOD_KEYS according to the specific account data.

  That is, instead of:

  period_key=
  mod(cycle_close_month
  +12*cycle_close_year),<period_dimension>)

  the function is expressed by:

  period_key=
  mod(cycle_close_month+<factor>+12*cycle_close_year),
  <period_dimension>)

  Since the factor is configurable and, for each account, it is calculated such that the final PERIOD_KEY value for the first partition is 1, then all accounts will have more or less the same physical configuration. The factor configuration is stored in the BL1_CONF_SECTION_PARAMS table with the name: PeriodPartitionFactor.

Important points to keep in mind:

1. The actual clean-up of data is preferable to merging into one history partition.
2. When defining the PERIOD ranges for physical partitions for a Production account, it is important to take into account the history data that is converted into the system, and then define the PERIOD_KEY values so that the history (if required in the operating system) is accumulated into a minimum of partitions.
3. The highest partition should be defined as MAX_VALUE, so that if maintenance activity of ADD PARTITION was not implemented for some reason, the most recent data will go to this partition, and not fail on insert.

- **CUSTOMER_KEY**

  For a typical Production account 20 partitions should be defined, each for a range of five CUSTOMER_KEYS, that is, values less than 5, values less than 10, and so forth.

  It should be kept in mind that this is not a hard-coded requirement and may be changed to better accommodate Account requirements.

Number of Groups During Bill Run

For a typical Production account 100 groups should be defined for Bill preparation.

This number, however, is configurable to provide maximum flexibility.

AMC

See Chapter 3.

Dispatcher

The function of the Dispatcher is to insert Rater results into the RATED_EVENT table.

In some configurations it also supplies these results to external agencies that either need to be informed of the results or participate in the rating process. In this case the Dispatcher generates the additional files and puts them into the shared Unix directory: pm/interfaces/output. The number of files is implementation-specific. In some implementations this number can be extremely large.

Output Directory Size

The directory size itself depends on the maximum amount of space needed to hold all the directory entries. Normally, directories do not shrink and a sure way to reduce their value is to recreate them. So, regardless of whether the file has later been deleted or moved, its size can extend to a significant size and affect the searching or writing time. Tests showed a degradation of more than 40% due to this.

This is not a Dispatcher-specific issue, and can happen to any directory that grows intensively.

Output File – Number of Records

The number of Dispatcher records is set by the MAX_COUNT field in the RPR1_DISP_TARGET table. It should be pointed out that a small size negatively affects the size of the output directory, that is, pm/interfaces/output, and causes a higher write call frequency. The recommended value is 1000 records per file.

In any case, the number of records should not exceed the XLA daemon number of records.

Event Extract

External Record

It has been noted that using external records, opposed to entity records, causes a degradation of 52%.

Extract Method

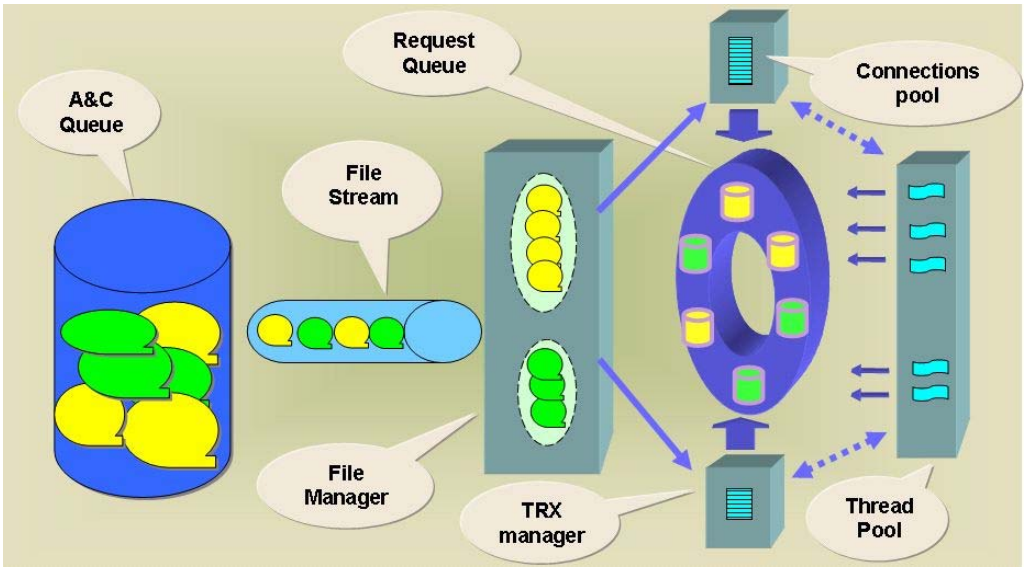
All events in the cycle month should be extracted in one event extract run rather than several separated runs. Extracting a subset of the rated events in the requested cycle period, as opposed to all rated events, causes large performance degradation. This is because each rated event is retrieved and analyzed prior to determining whether it matches the target population.

Rater

Rater Architecture

The diagram below depicts the current Rater configuration:

Figure 2-1: Rater Architecture



General Flow

The new Postpaid Rater uses the multithreaded paradigm for processing the Rater incoming events. The main idea is to keep the working threads (which do the actual event processing) as busy as possible.

Other threads retrieve the files and records asynchronously and serve them to the working threads.

Transaction Length

The transaction length parameter is very useful when dealing with small files.

Small files reduce the processing efficiency by several degrees (by more than 20%). Defining a transaction length that is big enough will reduce the file handling overhead (this is true also for processes following the Rater).

Recovery Mode

Working in recovery mode causes considerable performance degradation (mainly when using many buckets).

Transaction Manager

The Transaction Manager prepares the transaction about to be processed by the working threads and closes that transaction when the working threads are done. Each Transaction Manager is limited to processing one transaction at a time.

Working Threads

The working threads are stateless Pricing Engine threads that process the events stored in the Event Handler queue. The working threads serve each transaction manager that exists in the same process.

Using Buckets

A bucket is a collection of records, existing in the event handler queue. A bucket is processed exclusively by one thread. Other buckets in the queue are processed simultaneously. Generally speaking the more buckets in the queue the larger parallelism level is achieved.

Simplified Example

Consider the case in which the Rater has been configured with one Transaction Manager, four working threads and a transaction length of 1000. In order to benefit the four working threads one needs to split the transaction into at least four buckets. However the throughput can be further improved if it is split into a larger number of buckets.

How is this behavior possible? For the sake of simplification we assume that the buckets are equal in size. This does not mean that each thread will finish processing its bucket at the same time, actually, it is more likely that there is one thread that is the slowest (not containing the same number of Offers, or Pricing Item Types, and so forth).

Since a transaction is not completed until all working threads are done, all finished working threads will wait for the "lazy" thread to finish processing its bucket. These working threads will be idle during that time.

Ideally it would have been more efficient if the waiting threads could "help" the "lazy" thread processing its bucket by sharing its workload. However, since the same bucket can be processed by one thread only, this will not happen.

The duration idle threads exist can be, however, reduced by splitting the transaction into more buckets, thus decreasing the processing duration of the "lazy" thread

Assuming, for our example, that the “lazy” thread was around 10% less efficient, this would mean that it processed a bucket containing around 25 (1000/4*10%) additional records compared to the other buckets. Alternately, splitting into 16 buckets will reduce the additional records to 6 (1000/16*10%) records.

Considerations for Transaction Manager Configuration

As mentioned above, the working threads are shared by all the transaction managers which exist in the same processes. As a result the idle time per working threads may be reduced even more. This is because the working threads are stateless, when a thread is done processing its bucket, it can start to process another Transaction Manager bucket if one exists in queue.

Further, the workload is more balanced (more threads will be working for the most loaded Transaction Managers) since the queue will contain more buckets of the Transaction Manager that has to handle most of the workload. (It should be noted that **even** if more than one Rater process are loaded and

one of them has very little work to do while the other has a huge amount of work, the working threads **will not** alternate between processes).

So ideally, the more Transaction Managers per processes, the better efficiency is achieved. The best balance will be achieved in one process, but it is also possible to have several processes with a few Transaction Managers in each.

An additional aspect of having more than one Transaction Manager in the same process is that it makes the *decreased working threads idle time as opposed to the increased number of buckets* "feature" less "powerful", that is, it is not necessary to split too many buckets.

Database Connection

Using a database remote connection via a Customer database account will fail since the Performance Indicator (PI) has a LOB field and Oracle does not support updating a LOB in the Remote mode. The solution is, therefore, to connect to the Usage database.

Implementation Factor

As a rule of thumb, the most significant performance factor is the implementation type.

Tests showed a degradation of 35% between flat rate (the best results) and tier or step combinations.

Performance Indicator Locks

The Rater has two Performance Indicator (PI) Locking modes that are configured by the value set in the *lockPI* parameter:

- **True** – Safe mode. Rater locks the PI on the customer level.
- **False** – Fast mode. No locks exist and the integrity of the PI calculation is based on grouping by the customer's last digits.

The *lockPI* parameter is defined in the Rater's XML file. By default, the Rater should be configured to run in the fast mode. It is also suggested that Rater should write its mode type via its log.

Optimizer

NFT tested the effect of optimized code. When applied to past versions of Rater and Rerate, there was an improvement of more than 40% compared to the CC code. The recommended level is level two.

With Amdocs Billing version 6.0, the applications have been ported to 64-bit code. Unfortunately HP-UX does not support DOC (Debug-able Optimized Code) for 64-bit code as it did for 32-bit code. DOC provides that after the debugged code is approved, the code can be compiled again with an optimizer. On the other hand, Sun, as well as HP-UX with the new compiler, Itanium, claims to support DOC.

Database Tables Partition Number

Two parameters define the partition number:

- NUMBER_PARTITION – A field in the CYCLE_DEFINITION reference table
- PM_RATED_EVENT_SUB_PARTITIONS – An environment variable in the Unix op_pm_env_sh job script

Both definitions have to be identical and equal to the number of partitions in RATED_EVENT.

In future releases only one definition will remain.

Signal Handling

Signal handling permits an end user to determine whether the signal (exception) occurred in the Customization or the Core layer.

Signal operations are mostly executed in the System mode, a limited shared system resource used by all processes.

Given that scalability deterioration occurred when running several Raters together, the more processes that run simultaneously, the greater the contention is.

The attribute *HandleSignals* in the EventProcessingParameters tag sets it up. By default, it is `True`, meaning that signal handling is in use.

TimesTen Datastore Sizes

The data sizes for both the Table and the Index contents on a core TimesTen Rater (Customer and Usage) Datastore is given in the following table:

Row Size (in KB)	Sizing Factor	Table Name
0.313	Agreements	CUSTOMER_OFFERS
0.328	Agreements	CUSTOMER_PARAMETERS
0.392 (average)	Pricing items per subscriber in cycle month	PERFORMANCE_IND
Negligible	Number of Raters	PM1_LAST_EVENT_PROC
Varied	User groups	PM1_USR_GRP_MEM
Negligible	Number of cycles	RATED_EVENT
Negligible	Number of cycles	REJECTED_EVENT
Varied	Customers in cycle group	RPR1_CUSTOMER_EX
0.232	Subscribers for Rerate	SUBSCRIBER_RERATE

An agreement contains subscribers with offers.

A subscriber has a varied pricing schema that affects the amount of rows and sizes in the PERFORMANCE_IND table.

The RATED_EVENT and REJECTED_EVENT tables have fixed row numbers that are updated only during the run.

The PM1_LAST_EVENT_PROC table contains a row for each Rater to trace the condition of the run.

The RPR1_CUSTOMER_EX table is a temporary table for the event extract of Billing. During the run, it contains the customers of a cycle group. The data is deleted afterwards.

A subscriber requires about 0.7 KB in the Customer tables and about 0.4 KB for each Pricing Item Type (PIT) in the PI table.

Multi-Process Rater

In order to run Rater in the multi-process mode (NOT multi-thread) the following parameter needs to be set:

PM1_RUNRATER_$_1_UDP

The $_# needs to be defined for each cycle partition.

Rater Memory Allocation Patterns

The effect of environment variables associated with the *malloc* function on Rater performance was tested.

The results showed that the best fit for four or eight working threads with two acceptors, and two transaction managers is:

_M_ARENA_OPTS=12:1600
_M_SBA_OPTS=513:1000:32
_M_CACHE_OPTS=100:12:10

It improved the overall performance (execution time, throughput and memory consumption) in 16%-25%.

For more information see the “Memory Allocation Patterns” section in Chapter 6.

Re-Rating Extract

The Oracle limitation on extracting a LOB remotely means that the job must define the connections to a Usage database instance.

Technically, the GN1_TASK_CONNECT Operational table is filled with new RPR1CYCLE entries.

Transaction Broker

Compress Indicator

The Compress indicator defines whether transactions will be published in a compressed or an uncompressed mode.

In the uncompressed mode, chances are higher that transactions will have a length greater than 4K. Thus the drive inserts the transaction into a CLOB column instead of a Varchar column. Access to a CLOB column is more expensive and the penalty is an extra call to the database. The Compress indicator in the XML, however, minimizes the number of extra calls.

Database Structure

The Transaction Broker engine is an I/O bound module. The engine writes intensively to the database. In Oracle the critical I/O portion is the redo log. Redo logs are shared in the instance level. For I/O bottleneck elevation and to support higher transaction load, the Transaction Broker engine is defined as a stand-alone instance.

Tables Partitions

The following are Transaction Broker partitioned tables:

Table Name	Number of Partitions	Partitioned Columns
TRB1_MST_LOG	24	IMP_PERIOD
TRB1_PUB_LOG	?	SOURCE_COMP_ID
TRB1_SUB_LOG	15	SOURCE_COMP_ID, SUB_APPL_ID

Grants and Permissions

You must grant select on v_$session to public for all instances.

The usage is: **grant select on v_$session to public;**

Parameters

- **TRB1_SUB_QUE_SIZE = 5000**
- **TRB1_MST_IN_QUE_SIZE = 250**
- **TRB1_BRK_IN_QUE_SIZE = 50**
- NFT configuration used 4 brokers and 2 managers for each engine:

  TRB1_MAX_NUMBER_OF_MEMBERS = 4 and
  TRB1_MIN_NUMBER_OF_MEMBERS = 4 for MEMBER_ID = 5000 (Broker)

  TRB1_MAX_NUMBER_OF_MEMBERS = 2 and
  TRB1_MIN_NUMBER_OF_MEMBERS = 2 for MEMBER_ID = 6000 (Master)

Audit & Control

Audit & Control uses a distributed framework.

The site processes and the Audit & Control FTC engine can be located on the same server or on different machines. Accordingly the master Audit & Control tables and the site's Audit & Control tables can be located on the same instance or on different instances located on different machines. In the later case the master and site tables will be updated by the manager running on the Audit & Control FTC engine.

When working with sites on different machines, data files are copied to the sites by the Audit & Control engine. In this configuration large files take their toll and are the main factor influencing performance.

Update Handler

Acquisition & Formatting – Extract Phase

- The optimal number of processes for the Unload phase should be three or less.
- The Extract phase requires Operational table changes; otherwise, the owner name gets the database username instead of the operating system account name. Technically in the GN1_CONNECT_ PARAMS table the **UH_FTP_HOST=<host name>** parameter is replaced by **FTP_HOST=<host name>**.

Acquisition & Formatting – Loading Phase

An SQL loader limitation has been found in the Usage database connection.

Number of Threads

As a rule of thumb, the number of threads depends on the number of CPUs.

In the tests that were conducted for previous Enable versions it was shown that setting the number of threads to four achieves high utilization Defining more threads, however, does not contribute to the performance. The parameter is table-driven inside the GN1_ART_SECTION_PARAM table

The specific row data should have the values:

- **PARAM_CLASS='GEXT'**
- **SECTION_NAME='SERVER_PARAMETERS'**
- **PARAM_NAME='MIN_EH_THREAD_NUM'**
- **PARAM_VALUE=4**

In the future when load balancing is supported, an additional parameter will become important specifying the maximum number of threads. In this case the specific row data should have the following values:

- **PARAM_CLASS='GEXT'**
- **SECTION_NAME='SERVER_PARAMETERS'**
- **PARAM_NAME='MAX_EH_THREAD_NUM'**
- **PARAM_VALUE=10**

XLA Daemon

Output File – Number of Records

The XLA's output file record limit is set by three fields in the GN1_ART_SECTION_PARAM table (whichever reaches its limit first):

- MAX_SIZE – The maximum file size is currently set to about 4 MB
- MAX_RECORDS – The maximum number of records is currently set to 1000
- MAX_OPEN_PERIOD – The file timeout is currently set to 30 seconds.

The XLA output file size affects the Dispatcher performance, which relies on these files as an input. The Dispatcher closes its output files at the end of each input file (regardless of the parameter that is set for the output directory size).

Account Receivables

Parameters

The following parameters were set in the file **AR1App.properties** (generated by the sonar):

- **PARTITIONS_SIZE = 5**
- **NUMBER_OF_THREADS = 5** (1 was also tested to compare the results to v5.6 where multipule threading was not yet available).
- **BATCH_SIZE = 100**

Collection

Business Process Automation Step Management

For Business Process Automation (BPA) the HP-UX **max_thread_proc** kernel parameter was set to 1024.

In addition the following parameters were set in the $HOME/J2EEServer/config/ABP-BEFE/ABPServer/properties/**CL1EPI.xml** file in order to prevent the LockServer from getting into the Recovery mode or the BPA from throwing AssertionExceptions.

Thread Pool Service

- MaxThreads set to 100
- MinThreads set to 100
- InitialThreads set to 20

These settings mean that there will be more threads available for the BPA in-server executions.

Default

- MaxThreads set to 40
- MinThreads set to 40
- InitialThreads set to 20

InServer

- MaxThreads set to 90

- MinThreads set to 90
- InitialThreads set to 20

LockServer

- MaxThreads set to 40
- MinThreads set to 40
- InitialThreads set to 20

LockServer Service

- Period set to 300000 (5 minutes)
- Delay set to 300000 (5 minutes)

Distributed Lock Manager Service

- Period set to 300000 (5 minutes)
- Delay set to 300000 (5 minutes)

These settings mean that the LockServer and Distributed Lock Manager will send and receive heartbeat messages every 5 minutes instead of every 30 seconds.

CL Unstarted Case Scanner

- Period set to 0
- Delay set to 65000

Because the number of in-server execution threads is increased, one should also increase the number of DB connections in the ABP_CORE_POOL. The max on this pool was increased to 100.

Miscellaneous

UTL1_FILE_LOADER

UTL1_FILE_LOADER is an application table. One of its columns, FILE_DATA, is a CLOB column.

CLOB is a limited type of column and cannot be remotely accessed via a database link.

The table is used by both Customer and Usage jobs and is updated not more often than during sub-version maintenance.

The solution is to define the table in every database instance and a job that updates the table crosses the instances when needed.

3. ONLINE APPLICATIONS

This chapter provides guidelines for optimizing the online application configurations.

APInvoker

The following should be taken into consideration in connection with APInvoker

1. Routing destinations

   Assuming the time for an average Customer Management **action** is 2 seconds, the number of routing destinations that should be configured in the APInvoker should be 100-200.

2. Transaction size

   Since the transaction size has an effect on the performance of the APInvoker, the transactions should be as small as possible. This can be achieved by working with compressed transactions.

3. Type effect

   Working in the local mode has a slight performance advantage over working in the remote mode.

4. Acknowledge effect

   Working with acknowledgement reduces the APInvoker performance. Therefore in order to achieve maximum performance the APInvoker should be configured to work without acknowledgements. The decision to work with or without acknowledgements is mostly dependent on the business needs, thus it cannot be determined solely by the APInvoker needs.

5. Heap size

   In order for the APInvoker to work properly it is recommended to configure it to have at least 512 MB to 1 GB of heap size.

6. For the Sun Solaris platform

   The JDK version is required to be 1.4.2_10 because there's a bug in JVM in lower versions which causes the API Invoker to fail after a few hours due to garbage collection problems.

Customer Management

WebLogic Servers Setup

Install BEA WebLogic 8.1 server SP3.

EJB Server

WebLogic Configuration Parameters:

1. Change the HEAP Java parameters in the start up script in setenv.sh to:

   -server –Xms860M –Xmx860M -XX:MaxPermSize=128M

Where:

- -server – Running in server mode
- -Xms – Is the minimum Heap Size in MB
- -Xmx – Is the maximum Heap Size in MB
- Heap size should be specified according to the physical memory available to the application server machine.
- Set the Java permanent area to 128MB (if it is not the default, depending on the JVM version. Not available on AIX OS)
- Change the JDBC connection Pool for ABP_CORE_POOL in config.xml:

<JDBCConnectionPool CapacityIncrement="1"

DriverName="oracle.jdbc.driver.OracleDriver"

InitialCapacity="35" MaxCapacity="35"

Name="ABP_CORE_POOL"

2. Change Cache Size statement for ABP_CORE_POOL:

   StatementCacheSize="300"

3. Do not enable ABP_CORE_POOL Shrinking:
   ShrinkingEnabled="false"
4. Enlarge the "default Execute Queue" to 35 Thread Count.
   <ExecuteQueue Name="default" ThreadCount="35"
   ThreadsIncrease="1"/>

Shutting Down Debug Printing for Better Performance:

1. Make the following change in the ~/J2EEServer/config/ABP-BEFE/ABPServer/properties directory:

   amdocs.system.severity=fatal:yes;
   error:no;
   information:no;
   warning:no;
   debug:no

   amdocs.system.filter=true

   in the following files:

   - GN1GeneralLogHandler.properties
   - CM1GeneralLogHandler.properties
   - FE1CMClientLog.properties
   - UTL1GeneralLogHandler.properties
   - RPR1GeneralLogHandler.properties
   - BL1LogHandler.properties
2. Make the following change (put in remark) in the ~/J2EEServer/config/ABP-BEFE/ABPServer/properties/ GN1GeneralLogHandler.properties file:

   #amdocs.system.traceLevel=12
3. Make the following change in the ~/J2EEServer/config/ABP-BEFE/ABPServer/properties/messageHandlingServices.xml file:

<Filter>

<Severity information="no" warning="no" error="no" fatal="yes" debug="no"/>

</Filter>

4. Make the following change in the ~/J2EEServer/config/ABP-BEFE/ABPServer/properties/CM1Environment.properties file:

   amdocs.cm.debugMode=N

   amdocs.cm.testMode=N

5. Make the following change in the ~/J2EEServer/config/ABP-BEFE/ABPServer/properties/CM1JF.properties file:

   amdocs.log.enableLogging=N

   amdocs.log.enableLoggerNaming=N

6. Make the following change in the ~/J2EEServer/config/ABP-BEFE/ABPServer/properties/BL1App.properties file:

   amdocs.bl.debugMode=N

7. Make the following change in the ~/J2EEServer/config/ABP-BEFE/ABPServer/propertiesfRPR1JF.properties file:

   LoggerActive=false

8. Make the following change in the ~/ J2EEServer/config/ABP-BEFE/ABPServer/properties/SONARmessageHandling.xml file:

   <name>debug</name>

   <fullName>debug</fullName>

   <value>

   <current>no</current>

   ….

   <name>information</name>

   <fullName>information</fullName>

   <value>

   <current>no</current>

   ….

   <name>warning</name>

   <fullName>warning</fullName>

   <value>

   <current>no</current>

Customer Hierarchy Minimum Number of Connections:

Make the following change in the ~/J2EEServer/config/ABP-BEFE/ABPServer/properties /CM1Environment.properties file:

amdocs.util.pool.minConnection=35

RefDataManager Cache Size:

Make the following change in the ~/J2EEServer/config/ABP-BEFE/ABPServer/properties/CM1JF.properties file:

amdocs.rdm.cache.maxItems=5000

*WebLogic **must** be started by the startServer script with the **–n** flag or by changing the relevant variables in startServer script.*

EJB and JSP/ SERVLET Server in One JVM

In the file FE1CMConfig.properties disable all the applications that are irrelevant for the accounts business!

The following WebLogic configuration values have been adjusted & tested with 1000 Virtual Users.

WebLogic Configuration Parameters:

1. Change the Java parameters in the start up script in setenv.sh to:

   -server -Xms1536M -Xmx1536M -XX:PermSize=128m -XX:MaxPermSize=128M -XX:NewSize=256m -XX:MaxNewSize=256m -XX:SurvivorRatio=10"

2. Change JDBC connection Pool for ABP_CORE_POOL in config.xml:

   <JDBCConnectionPool CapacityIncrement="1"

   DriverName="oracle.jdbc.driver.OracleDriver"

   InitialCapacity="35" MaxCapacity="35" Name="ABP_CORE_POOL"...

3. Change Cache Size statements for ABP_CORE_POOL:

   <JDBCConnectionPool CapacityIncrement="1"

   StatementCacheSize="300" Targets="ABPServer"

4. Enlarge "default Execute Queue" to a Thread Count of 35:

   <ExecuteQueue Name="default" ThreadCount="35" ThreadsIncrease="1"/>

5. Turn off Web Server Logging:

   <WebServer LogFileName="./config/ABP-BEFE/logs/access.log"

   LoggingEnabled="false" Name="ABPServer"/>

6. Increase JSP Performance:

   Set pageCheckSeconds to -1 in the ~ J2EEServer/config/ABP-BEFE/applications/gcc600V64/webAppRoot/WEB-INF /weblogic.xml file:

   <jsp-param>

   <param-name>pageCheckSeconds</param-name>

   <param-value>-1</param-value>

   </jsp-param>

   It improves performance due to the following reasons:

   - **pageCheckSeconds** sets the interval, in seconds, at which WebLogic Server checks to see if JSP files have changed and need recompiling. Dependencies are also checked and recursively reloaded if changed.
   - If set to 0, pages are checked on every request. This default is preset for a development environment. If set to -1, page checking and recompiling is disabled.

- In a production environment where cchanges to a JSP are rare, change the value of pageCheckSeconds to 60 or greater, according to your tuning requirements, or to -1 to disable page checking and recompiling.

Shutting Down Debug Printing for Better Performance:

Perform the same setup changes as in the **EJB Server** section (See above).

UAMS system Cache Settings

For the UAMS setting, see the "UAMS" section bellow.

EJB Quantities

The following table provides the recommendations for up to 100 concurrent users accessing each one of the EJBs. If a different number of concurrency is used, you have to change the parameters in the WebLogic console or the weblogic-ejb-jar.xml and ejb-jar.xml files.

EJB Name	Type	Initial Beans in Cache	Max Beans in Cache
AgreementDistributionConvBeanSec	Stateless	100	
AgreementSrvAgrConvBeanSec	Stateful		100
ApplManagerConvBeanSec	*		
BarringItemSessionBeanSec	Stateful		100
BarringServicePackageSessionBeanSec	*		
BillingArrangementServicesBean	Stateless	50	
BillingItemSessionBeanSec	Stateful		100
BillingPackageSessionBeanSec	Stateful		100
ChangeCustomerTypeConvBeanSec	Stateful		100
ConfigurationSessionBeanSec	*		
EventGroupItemSessionBeanSec	*		
EventGroupPackageSessionBeanSec	*		
FeatureGroupSessionBeanSec	*		
ICCManagerConvBeanSec	*		
MemoServicesBean	Stateless	50	
NewAccountConvBeanSec	Stateful		100
NewAgreementConvBeanSec	Stateful		100
NewBillingArrangementConvBeanSec	Stateful		100
NewCustomerConvBeanSec	Stateful		100
NewPayChannelConvBeanSec	Stateful		100
NewSubscrConvBeanSec	Stateful		100
ParameterSessionBeanSec	*		
PayChannelServicesBean	Stateless	50	

EJB Name	Type	Initial Beans in Cache	Max Beans in Cache
PricingItemSessionBeanSec	*		
ProvisioningItemSessionBeanSec	*		
RPL1RechargeSessionBeanSec	*		
SaleEventConvBeanSec	*		
SearchServicesBeanSec	Stateless	80	
ServicePackageSessionBeanSec	*		
SubscrMigrateConvBeanSec	*		
SubscrMoveConvBeanSec	Stateful		100
SubscrSrvAgrConvBeanSec	Stateful		100
SubscriberDistributionConvBeanSec	*		
UpdateNameAddressBean	Stateless	80	
CustomerHierarchyServicesBean.jar	Stateless	50	

* Specification for the EJB quantity is unknown since it was not included in our test scenarios.

Application Server Kernel Parameters

The following are the kernel parameters as defined in the application server to support the online execution.

The lines highlighted in yellow have been changed recently.

Parameter	Current	Dynamic	Planned
NSTRBLKSCHED	-	-	2
NSTREVENT	50	-	50
NSTRPUSH	16	-	16
NSTRSCHED	0	-	0
STRCTLSZ	1024	-	1024
STRMSGSZ	65535	-	65535
acctresume	4	-	4
acctsuspend	2	-	2
aio_listio_max	256	-	256
aio_max_ops	2048	-	2048
aio_physmem_pct	10	-	10
aio_prio_delta_max	20	-	20
allocate_fs_swapmap	0	-	0
alwaysdump	1	-	1
bootspinlocks	-	-	256
bufcache_hash_locks	128	-	128
bufpages	0	-	0
chanq_hash_locks	256	-	256
core_addshmem_read	0	Y	0
core_addshmem_write	0	Y	0
create_fastlinks	1	-	1
dbc_max_pct	10	-	10
dbc_min_pct	5	-	5
default_disk_ir	0	-	0
desfree	-	-	0
disksort_seconds	0	-	0
dmp_rootdev_is_vol	0	-	0
dmp_swapdev_is_vol	0	-	0
dnlc_hash_locks	512	-	512
dontdump	0	-	0
dskless_node	-	-	0

Parameter	Current	Dynamic	Planned
dst	1	-	1
effective_maxpid	-	-	((NPROC<=30000)?30000: (NPROC*5/4))
eisa_io_estimate	-	-	0x300
enable_idds	0	-	0
eqmemsize	15	-	15
executable_stack	1	-	1
fcp_large_config	0	-	0
file_pad	-	-	10
fs_async	0	-	0
ftable_hash_locks	64	-	64
hdlpreg_hash_locks	128	-	128
hfs_max_ra_blocks	8	-	8
hfs_max_revra_blocks	8	-	8
hfs_ra_per_disk	64	-	64
hfs_revra_per_disk	64	-	64
hp_hfs_mtra_enabled	1	-	1
hpux_aes_override	-	-	1
initmodmax	50	-	50
io_ports_hash_locks	64	-	64
iomemsize	-	-	40000
ksi_alloc_max	32000	-	(NPROC*8)
ksi_send_max	32	-	32
lotsfree	-	-	0
max_async_ports	50	-	50
max_fcp_reqs	512	-	512
max_mem_window	10	-	10
max_thread_proc	256	-	256
maxdsiz	0x80000000	-	0X80000000
maxdsiz_64bit	0x80000000	-	0X80000000
maxfiles	1024	-	1024
maxfiles_lim	5000	Y	5000
maxqueuetime	-	-	0
maxssiz	0x17000000	-	0X17000000
maxssiz_64bit	0x17000000	-	0X17000000
maxswapchunks	16384	-	16384
maxtsiz	0x18000000	Y	0X18000000

Parameter	Current	Dynamic	Planned
maxtsiz_64bit	0x40000000	Y	0X40000000
maxuprc	512	Y	512
maxusers	800	-	800
maxvgs	255	-	255
mesg	1	-	1
minfree	-	-	0
modstrmax	500	-	500
msgmap	5002	-	(2+MSGTQL)
msgmax	65535	Y	65535
msgmnb	65535	Y	65535
msgmni	1024	-	1024
msgseg	7168	-	7168
msgssz	512	-	512
msgtql	5000	-	5000
nbuf	0	-	0
ncallout	7016	-	7016
ncdnode	150	-	150
nclist	5120	-	5120
ncsize	25120	-	(NINODE+VX_NCSIZE)+ (8*DNLC_HASH_LOCKS)
ndilbuffers	30	-	30
netisr_priority	-	-	-1
netmemmax	-	-	0
nfile	20000	-	20000
nflocks	1000	-	1000
nhtbl_scale	0	-	0
ninode	20000	-	20000
nkthread	7016	-	(((NPROC*7)/4)+16)
nni	-	-	2
no_lvm_disks	0	-	0
nproc	4000	-	4000
npty	1024	-	1024
nstrpty	1024	-	1024
nstrtel	1024	-	1024
nswapdev	10	-	10
nswapfs	10	-	10

Parameter	Current	Dynamic	Planned
nsysmap	8000	-	8000
nsysmap64	8000	-	8000
o_sync_is_o_dsync	0	-	0
page_text_to_local	-	-	0
pfdat_hash_locks	128	-	128
public_shlibs	1	-	1
region_hash_locks	128	-	128
remote_nfs_swap	0	-	0
rtsched_numpri	32	-	32
scroll_lines	100	-	100
scsi_max_qdepth	8	Y	8
scsi_maxphys	1048576	-	1048576
sema	1	-	1
semaem	16384	-	16384
semmap	4098	-	4098
semmni	4096	-	4096
semmns	4096	-	4096
semmnu	512	-	512
semmsl	2048	Y	2048
semume	512	-	512
semvmx	32767	-	32767
sendfile_max	0	-	0
shmem	1	-	1
shmmax	0x60000000	Y	0X60000000
shmmni	1024	-	1024
shmseg	1024	Y	1024
st_ats_enabled	0	-	0
st_fail_overruns	0	-	0
st_large_recs	0	-	0
st_san_safe	0	-	0
streampipes	0	-	0
swapmem_on	1	-	1
swchunk	2048	-	2048
sysv_hash_locks	128	-	128
tcphashsz	0	-	0
timeslice	10	-	(100/10)

Parameter	Current	Dynamic	Planned
timezone	420	-	420
unlockable_mem	0	-	0
vas_hash_locks	128	-	128
vnode_cd_hash_locks	128	-	128
vnode_hash_locks	128	-	128
vol_checkpt_default	10240	-	10240
vol_dcm_replay_size	262144	-	(256*1024)
vol_default_iodelay	50	-	50
vol_fmr_logsz	4	-	4
vol_max_bchain	32	-	32
vol_max_nconfigs	20	-	20
vol_max_nlogs	20	-	20
vol_max_nmpool_sz	4194304	-	(4*1024*1024)
vol_max_prm_dgs	1024	-	1024
vol_max_rdback_sz	4194304	-	(4*1024*1024)
vol_max_vol	8388608	-	(8*1024*1024)
vol_maxio	256	-	256
vol_maxioctl	32768	-	32768
vol_maxkiocount	2048	-	2048
vol_maxparallelio	256	-	256
vol_maxspecialio	256	-	256
vol_maxstablebufsize	256	-	256
vol_min_lowmem_sz	524288	-	(512*1024)
vol_mvr_maxround	256	-	256
vol_nm_hb_timeout	10	-	-10
vol_subdisk_num	4096	-	4096
vol_vvr_transport	1	-	-1
vol_vvr_use_nat	0	-	0
volcvm_cluster_size	16	-	16
volcvm_smartsync	1	-	1
voldrl_max_drtregs	2048	-	2048
voldrl_min_regionsz	512	-	512
voliomem_chunk_size	65536	-	(64*1024)
voliomem_maxpool_sz	4194304	-	(4*1024*1024)
voliot_errbuf_dflt	16384	-	16384
voliot_iobuf_default	8192	-	8192

Parameter	Current	Dynamic	Planned
voliot_iobuf_limit	131072	-	131072
voliot_iobuf_max	65536	-	65536
voliot_max_open	32	-	32
volraid_rsrtransmax	1	-	1
vps_ceiling	1024	-	1024
vps_chatr_ceiling	1024	-	1024
vps_pagesize	4	-	4
vx_fancyra_enable	0	-	0
vx_ncsize	1024	-	1024
vx_ninode	0	-	0
vxfs_max_ra_kbytes	1024	-	1024
vxfs_ra_per_disk	1024	-	1024
vxtask_max_monitors	32	-	32

Table Partitioning

There is limited experience in running NFT on Customer Management partition tables.

In the Resource Management area the numbers are the expected ones.

The CM1 tables in the NFT have only one partition. However, it has no impact on the Customer Management tests and the output results.

The following table contains a list of typical Customer Management and Resource Management partitions:

Table Name	Number of Partitions	Partitioned Columns
CM1_AGREEMENT_PARAM	100	AGREEMENT_KEY
CM1_AGR_PRM_HISTORY	2400	PERIOD_KEY, AGREEMENT_KEY,
CM_USR_GRP_MEMBERS	100	ADMIN_CONTEXT_KEY, ADMIN_CONTEXT_ID_KEY
MO1_MEMO	1000	PERIOD_KEY, APP_ID, ENTITY_ID
MEMO	2400	ENTITY_KEY,PERIOD_KEY

UAMS

A UAMS cache setting depends on the following variables:

1. The application server UAMS configuration.
2. The number of user clients.

Verify that you use UAMS release jar version (with fixes after v0.50-b106 release jar

Be careful with blank spaces in variable's values.

The following cache variables values have been adjusted and tested with 1000 Virtual Users.

*In Full JVM configuration, all of the following parameters should reside on one and the same **uams.properties** file.*

UAMS Server

Set the following parameters in the ~/J2EEServer/config/ABP-UamsServer/SecServer/properties/uams.properties file:

SESS_CONSTRNT_SESS_LIMIT=1200

SESS_CACHE_TTL=1800

SESS_FACT_IDL_TIMOUT=3600

AUTHZ_CACHE_SIZE=3000

AUTHZ_CACHE_TTL=900

USER_CACHE_INIT_SIZE=1000

USER_CACHE_TTL=1800

ROLE_HIER_CACHE_SIZE=3000

ROLE_HIER_INIT_SIZE=2700

ROLE_CACHE_TTL=1800

RES_CNSTR_CACHE_SIZE=3000

RES_CNSTR_CACHE_TTL=1800

NSPACE_CACHE_NULL_TTL=1800

POLICY_CACHE_SIZE=500

SESS_FACT_EXP_TIMOUT=86400

This sets the Session Expired TIMOUT to 24 Hours

SEC_DB_RESOURCE_LIMIT=150

This is the maximum number of connections to security DB by UAMS.

SEC_DB_EXPIRATION_TIME=3600

This is the idle time for a DB connection that has already been used, before it will be removed (seconds).

SEC_DB_IDLE_TIME=300

This is the idle time for a DB connection that has NOT been used, before it will be removed (seconds).

SEC_DB_GRANTED_FREE=50

This is the minimum number (and step additions) of connections by UAMS to security database.

EJB Server

Set the following parameters in the ~/J2EEServer/config/ABP-BE/ABPServer/properties/uams.properties file:

SESS_CACHE_TTL=1800

SESS_FACT_IDL_TIMOUT=3600

AUTHZ_CACHE_SIZE=3000

AUTHZ_CACHE_TTL=900

ROLE_HIER_CACHE_SIZE=3000

ROLE_HIER_INIT_SIZE=2700

ROLE_CACHE_TTL=1800

POLICY_CACHE_SIZE=3000

POLICY_CACHE_TTL=1800

RESOURCE_CACHE_SIZE=3000

RESOURCE_CACHE_TTL=1800

TICKET_CACHE_TTL=1800

JSP/SERVLET Server

Set the following parameter in the ~/J2EEServer/config/ABP-FE/ABPServer/properties/uams.properties file:

SESS_CACHE_TTL=1800

Usage Query

The following should be taken into consideration when running Usage query:

Application Server Changes

1. For better performance of the Usage Query Java APIs operation, see the "**EJB Server**" in the Customer Management section in Chapter 3.
2. The ART Pool Size of relevant partition should be increased to 10 (the default is 1) on the WebLogic server in the ~/J2EEServer/config/ABP-FULL/ABPServer/properties/RPR1JF.properties file.

Batch Server Changes

1. The REACTOR_THREAD_NUM parameter for Usage Query Daemon should be increased to 20 (the default is 3) on the batch machine in the GN1_ART_SECTION_PARAM table in the operational account (for the following where clause: PARAM_CLASS = 'UEXT_UQ' and PARAM_NAME='REACTOR_THREAD_NUM').
2. The DB connection parameters for the Usage Query Daemon should be increased to 40 on the batch machine in the GN1_ART_SECTION_PARAM table in the operational account (for the following *where* clause: PARAM_CLASS = 'UEXT_BASE' AND SECTION_NAME like '%CONNECTION_POOL' and PARAM_NAME like '%DB_CONNECTIONS').

3. The MAX_TCP_CONN_NUM parameter for Usage Query Daemon should be increased to 50 on the batch machine in the GN1_ART_SECTION_PARAM table in the operational account (for the following where clause: PARAM_CLASS = 'UEXT_BASE' and PARAM_NAME = 'MAX_TCP_CONN_NUM').

Publisher Applications

To improve performance in the publisher applications, it is required to publish the transactions in Compress mode to the TRB area.

Each application publisher (for example: CM, CRM, and the like) should publish compressed transactions to the TRB tabled.

How to configure the compress mode:

Change in the UTL1_FILE_LOADER application table for the following where clause: FILE_NAME like '%PUBLISHER_REGISTER%' – in all the XMLs of the column FILE_DATA, the **Test-Mode** variable should be "**false**"

4. EXTERNAL TOOLS

This chapter provides guidelines for configuring various tools and the environment for employment in NFT for running further optimization testing for the Amdocs Billing Platform 6.0.

AMC

Due to an AMC limitation, this application cannot send a message to start dependent applications that are connected to a different Message Router (MRO). Jobs that are instantiated by another job should connect to the initial job's MRO.

Oracle

1. Oracle Patch #4130829 must be installed on all platforms (Sun, HP, AIX) in order to solve the following bugs:
   - Bug #3853332 (OCCI memory leak).
   - Bug #3992072 (use of setNull and addIteration)
2. The NFT database environment should be created to support 5M customers. This was implemented in a sizing model that extrapolates database volume according to the various NFT activities that run in a 5 MB environment.
3. The Oracle tnsnames' definition should include only TCP protocol connections. No BEQ protocol for local connections should be defined in the tnsnames file.

   The reason for this is that the Java 64-bit process does not work when JNI (Java calling a C/C++ functions) is used. Local connections should be optimized to connect directly via a pipe bypassing the listener module.
4. The NFT database is built on top of raw devices. The advantages of raw devices are that the file system cache can be bypassed and the I/O can be configured to work in asynchronous mode. According to Amdocs' experience raw device setup provides a throughput improvement of 20-30%.
5. The raw device setup is done, side-by-side, by system and database administrators. A side effect of working in raw devices is that the administration needs to be much more organized and planned ahead. For convenience, the following conventions and approaches have been used:
   - Raw devices are gathered into several types in the /dev/vg* directory: DATA, INDEX, TOOLS, UNDO, TEMP, REDO and CTRL. For each type several standard space sizes are allocated (plus an overhead space for the raw device handling), i.e., 10 GB, 5 GB, and 1 GB. The idea is to have a bulk of free space that can be allocated on demand.

- Symbolic links located in the /oravl01/ORACLE/<instance name>/dbf directory point to the raw devices. Those symbolic names in which their space is not allocated have the suffix: **.unused**. Others whose space is allocated follow the convention: <instance_name>_<tablespace_name>_<#>_<tablesapce type>_<size>_<#>.

6. Multiple buffer pools

   For faster access and physical I/O elimination, Oracle caches data blocks in a shared memory area. When it comes to how long Oracle should keep objects in the cache, Oracle segments have varying usage patterns and segments should be treated differently. Hence, to speed up and save disk reads, frequently accessed objects should be kept in the memory and rarely accessed objects should be prevented from taking critical memory space. Recommendations, on the object level, are:

Instance Type	Buffer Pool Type	Object Name	Object Type
Customer	Keep	Address_name_link	Tables and indexes
Customer	Keep	Name_data	Tables and indexes
Customer	Keep	Ch_arcs	Tables and indexes
Customer	Keep	Ch_arcs_attributes	Tables and indexes
Customer	Keep	Ch_objects	Tables and indexes
Customer	Keep	Subscriber	Tables and indexes
Customer	Recycle	Memo	Tables and indexes
TRB	Recycle	Trb1_mst_log	Tables and indexes
Usage	Keep	Performance_ind	Tables and indexes
Recycle	Recycle	Rated_event	Tables and indexes
Recycle	Recycle	Rejected_event	Tables and indexes

7. Parameters in the Init file are compliant with Amdocs database core team recommendations and configured for raw device layout. The following table focuses on certain, but not all, of the parameters that are define in the Init file:

Parameter	Value	Comment
Compatible	9.2.0.5	
Undo_management	AUTO	
Undo_tablespace	UNDOTBS	
Undo_retention	3600	Time, in seconds, data kept for rollback. Prevent snapshot too old errors
Db_block_size	8192	
Db_cache_size	1000M	Size of the default buffer pool. Replace DB_BLOCK_ BUFFERS

Parameter	Value	Comment
Db_chace_advice	ON	Dynamically calculate, based on views, Db_cache_size
Db_recycle_chace_size	< Size - based on Recycle objects sizes in bullet 5 >	Release data blocks out of the memory, as soon as they are redundant.
Db_keep_chace_size	< Size - based on Keep objects sizes in bullet 5 >	Retains data blocks in the memory
Shared_pool_size	209715200	
Sga_max_size		Formula: 16M+db_cache_size+ shared_pool_size+ large_pool_size+ java_pool_size
Disk_async_io	True	supports raw device layout
Db_file_multiblock_read_ count	8	
Db_writer_processes	1	For raw device one. Otherwise #CPU / 8
Log_checkpoint_interval	999999999	
Log_checkpoints_to_alert	True	
Log_buffer	1048576	
Processes	500	
Open_cursors	2000	
Job_queue_processes	4	
Parallel_max_servers	16	
Parallel_min_servers	0	
Max_enabled_roles	100	
Open_links	30	
Optimizer_mode	Choose	
Optimizer_index_cost_adj	1	
Session_chached_cursors	300	
Hash_join_enabled	True	
Workarea_size_policy	Auto	Obsoletes all the area_size parameters
Pga_aggregate_target	500M	Total instance memory – SGA size

Parameter	Value	Comment
Hash_area_size	-	Replaced by Workarea_size_policy and Pga_aggregate_target
Sort_area_size	-	Replaced by Workarea_size_policy and Pga_aggregate_target
Sort_area_retained_size	-	Replaced by Workarea_size_policy and Pga_aggregate_target
Query_rewrite_enabled	True	
Resource_limit	False	
Plsql_native_make_utility	/usr/bin/make	
Db_block_checksum	False	

8. Extensions are managed locally and allocated automatically (using the AUTOALLOCATE option).
9. When building an Oracle instance on the Linux platform, the following kernel parameters must be set in the /etc/sysctl.conf file:

   kernel.shmmax = 2147483648

   fs.file-max=65535

   Otherwise, the DB creation will fail with following error: ORA-27125: unable to create shared memory segment
10. All partitioned tables should be partitioned according to the recommendations provided in Appendix A. Failing to do so will result in significant performance degradation.

TimesTen

The following settings should be used for TimesTen:

- **Driver:** /opt/TimesTen/<TimesTen instance name>/lib/libtten.sl
- **DataStore:** <The full path name of the data store>
- **LogDir:** <The directory in which the log files reside>
- **PermSize:** <The permanent segment size should be set according to the Datastore proposed Tables content>

  To see the current capacity, run the **dssize** command from the **ttIsql** utility.
- **TempSize:** <The temporary segment size should be set according to the Datastore proposed operation>

  To see the current capacity, run the **dssize** command from the **ttIsql** utility.

If not specified, TimesTen size calculation is as follows:

If PermSize is less than 64 MB, TempSize = TempSizeMinimum + (PermSize / 4); otherwise, TempSize = TempSizeMinimum + 8 MB + (PermSize / 8). The TempSizeMinimum is 2 MB.

- **LogBuffSize:** <The LogBuffSize attribute specifies the size of the internal log buffer in kilobytes>

  The default log buffer size is 8192 KB. We recommend setting the size to **65536 (64 MB).**

- **LogFileSize:** <The LogFileSize attribute specifies the maximum size of log files>

  The default value is 8 MB. We recommend setting the size to **64** (same as the LogBufferSize).

- **DurableCommits:** 0

- **SMPOptLevel:** 1

- **MemoryLock:** <TimesTen allows applications that connect to a shared Datastore to specify whether or not the real memory should be locked during Datastore loading>

  If locked, the memory cannot be paged and is not available to other applications in the system. While memory locking can improve Datastore load performance, it may impede other applications on the same machine.

  Our recommendation is to use a value of **4**. This value places the Datastore in a continuous resident memory.

- **AutoCreate:** 0
- **Authenticate:** 0
- **UID:** <Owner of the tables (serves only for identification and not for security)>

1. Currently we use a Direct connection instead of Client/Server on the same machine. For Remote connection environments, you will need to create Server entries in sys.ttconnect.ini on the Client machine:

   [<Server logical name>]

   - **Description= TimesTen Client/Server**
   - **Network_Address=**<Server physical name >
   - **TCP_PORT=**<TimesTen Server port on the server>

2. For a Remote connection environment, create Client DSN entries in $ODBCINI file on the Client machine:

   [ODBC Data Sources]

   - < Client_DSN_name >**=TimesTen 5.1 Client Driver**
   - **[<Client_DSN_name>]**
   - **TTC_Server=**< TimesTen Server logical name>
   - **TTC_Server_DSN=**< TimesTen Server DSN name >
   - **UID=**<UNIX user who connects to the server DSN machine>

   When working with Authenticate=0, this attribute is not relevant.

3. Calculate table and columns statistics, this is important after major Datastore changes:

   Run the following command from **ttIsql** for all the tables in the Datastore:

   Call ttOptUpdateStats(<table_name>, 1);

   If you are not running in the Autocommit mode, then perform the commit at the end of the commands.

4. Run a checkpoint program (the TimesTen **ttMonitor** application) for each TimesTen Datastore. This is required not only for the Rater Datastores, but for the A & F Datastores as well. The interval should not exceed 300 seconds, and on highly active environments it is recommended to set it as 180 seconds:

   - **ttMonitor** is activated through the ttchkmon script by **cron**:

     ttMonitor -dsn $DSN -monitor 3 -ckpt 300 –durable 60 -compact 30 -permsize 80 -tempsize 80 -v &

   - The **Checkpoint** application uses the pm1checkPointProcess.xml configuration file:

     <WaitInterval interval="300"/>

5. Currently there is a Unix limit of 2 GB for a single file on a file system. For Datastores that exceed 2 GB (Perm + Temp size), it is important to change the checkpoint file systems to support large files.
6. The TimesTen Datastores are sensitive to abrupt session termination. Performing a **kill -9** on an application that is connected to a TimesTen Datastore, especially in the Direct mode, may cause a Datastore invalidation and long-time recovery. Try to avoid a harsh termination without signal handling on TimesTen applications.
7. When running the Rater, not working with mem2file, on the TimesTen Datastores, it is important that the XLA application be running as well; otherwise, the XLA will hold a Bookmark on the previous log files. The Checkpoint application will not be able to delete these files and once log file system gets full, the Datastore will freeze and the Rater will not be able to continue the run.

5. HARDWARE CONFIGURATIONS

This chapter provides guidelines for platform oriented configuration that will ensure a better operation of the applications and tools.

Disk Setup

The physical disk setup is critical for the performance results. Tests have shown an up to 50% gap, depending on the application characteristics. It should be taken into consideration that the write activity has more of an effect than the read activity. Disk allocation should be planned ahead, with emphasis on the applications priorities and their I/O intensiveness.

The following two factors are crucial for the disk setup:

- RAID-1 (mirrored disks) versus RAID-S (parity)
- Data on external tracks on the spindle achieve higher throughput.

LINUX Suse SLES9 OS

On the Linux Suse SLES9 OS the BIOS does not react correctly to ACPI instructions, causing clock to misbehave and sometimes move backwards due to lowering the processors' clock speed.

To solve this problem - on kernel 2.6 (SuSE SLES9) based systems we disable the ACPI mechanism in order to avoid the clock gaps. We boot the system with 'acpi=off' boot parameter.

6. MEMORY ALLOCATION PATTERNS

This chapter provides guidelines for more effective handling of memory allocation to improve the overall performance of the applications.

Memory Allocation Problem

A lot of calls to *malloc()* can cause more contentions between them and bottlenecks when they all try to access the same part of memory (the heap).

The Unix operating system sometimes does not handle the memory for dynamic allocation correctly and needs tuning.

The above problems cause degradation in performance.

Proposed Solution

Reduce the number of unneeded calls to *malloc* in the code:

- Some dynamic allocations could be done statically instead.
- De-allocated objects in a pool for reuse.

  Objects in the pool will be used preferably over freshly allocated objects and time is gained on the saved *malloc* calls.
- Reallocations could be omitted if enough memory is allocated to begin with.
- Improve the way HP UNIX handles the memory for *malloc*. Three environment variables can be set to configure the *malloc* behavior:
  - Bottlenecks due to contentions between multiple threads, while trying to allocate memory from the same arena, degrades performance, the _M_ARENA_OPT variable can help:

    _M_ARENA_OPTS=#arenas:#extensionpages.

    Arena is piece of memory that is given to the application for dynamic allocations. Each arena can be extended by the extension (*extensionpages*). In general, the more threads in an application that have a lot of *malloc* activity, the more arenas should be used for better performance. In this way threads are distributed among different arenas and contention will not happen.

    The expansion factor controls how many pages to expand each time. The default values are: _M_ARENA_OPTS=8:32
  - Small Block Allocator (SBA) will allocate small memory units more efficiently.

    The a C++ runtime now automatically enables *malloc*'s Small Block Allocator (SBA). This improves heap performance for *malloc()*.

    _M_SBA_OPTS= MXFAST: NLBLKS: GRAIN

    Where:

- MXFAST: The algorithm allocates all blocks below this size in large groups, then doles them out very quickly for *malloc()*.
- NLBLKS: The above mentioned "large groups" each contains NLBLKS blocks.
- GRAIN: The sizes of all blocks smaller than MXFAST are considered to be rounded up to the nearest multiple of grain. The default values are: _M_SBA_OPTS= 512:100:8

- _M_CACHE_OPTS can improve speed performance for some threaded applications, by reducing contention among threads. This environment variable configures a thread-private cache for "malloc'ed" blocks. Malloc'ed blocks are freed into a thread's private cache, and may thereafter be allocated from there when *malloc* is called.

  By default cache is not active and must be activated by setting _M_CACHE_OPTS to a legal value.

  _M_CACHE_OPTS=bucket_size:buckets:retirement_age

  Where:

  - *bucket_size:* The number of cached pages per bucket that will be held in the ordinary block cache.
  - *buckets:* The number to the power of 2 buckets that will be maintained per thread.

    <bucket_size><buckets>* is the maximum number of ordinary blocks that will be cached per thread.
  - *retirement_age*: Number of minutes for which unused caches are retained for possible reuse.

- Limitations and downside for the environment variables setting:
  - Threaded applications experience increased heap storage usage.
  - Each Arena has its own SBA. If enabled it increases the memory that is assigned to the application.

  Each thread gets private cache with _M_CACHE_OPTS. Therefore, activating a modest-sized cache is needed to decrease memory usage.

Appendix A. Recommended Table Partitioning

This appendix provides the recommended partitioning of all tables. The recommendations are based upon Production experience, NFT knowledge and tests.

Module	Table	Partition Key	Method	Number of Partitions
AC	AC1_AUDIT_ BALANCE	TABLE_ISSUE_ CODE	Static	Currently defined as static but should be changed to dynamic. TABLE_ISSUE_CODE - according to reference table AC_PROGRAMS. TABLE_ISSUE_CODE. Each Program is directed to different value of TABLE_ISSUE_CODE.
AC	AC1_CONTROL	TABLE_ISSUE_ CODE	Dynamic	TABLE_ISSUE_CODE - according to reference table AC_PROGRAMS. TABLE_ISSUE_CODE. Each program is directed to a different value of TABLE_ISSUE_CODE.
AC	AC1_CONTROL_ HIST	IMP_PERIOD	Dynamic	Partition per week: Current Date – Period Base (Period Base parameter, a date defined in the TIMER Thread in AC_Configuration.xml.ksh) / 7) modulo cycle_size, which is defined in the TIMER Thread in AC_Configuration.xml.ksh
AC	AC1_PHY_FILES	IMP_PERIOD	Dynamic	Partition per week: Current Date - Base Date (Base Date parameter, a date defined in TIMER Thread in AC_Configuration.xml.ksh) / 7) modulo cycle_size which is defined in TIMER Thread in AC_Configuration.xml.ksh
AC	AC_AUDIT_ SUMMARY	IMP_PERIOD	Dynamic	Partition per week (Current Date - Base Date (Base Date parameter, a date defined in TIMER Thread in AC_Configuration.xml.ksh) / 7) modulo cycle_size, which is defined in TIMER Thread in AC_Configuration.xml.ksh
AMC	AMC1_API_ TRAN_CELLS	GROUPID	Static	Default is 7 partitions, one per day. Less than 1, less than 2… less than 7 Formula: Calendar.DAY_OF_MONTH modulo mcSystem.properties viz.amc.apiHistory.partitionsModulo
BL	BL1_ACTIVITY_ HISTORY	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_BACKDATE _ REQUESTS	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo Property CustomerPartitionDimention
BL	BL1_BILL_ADD_ COMPS	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_BILL_ADD_ COMPS	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property.

Module	Table	Partition Key	Method	Number of Partitions
BL	BL1_BILL_ FINANCE_ACT	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property.
BL	BL1_BILL_ FINANCE_ACT	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_BILL_ STATEMENT	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_BILL_ STATEMENT	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_CHARGE	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_CHARGE	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_CHARGE_A CC	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_CHARGE_A CC	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_CHARGE_A DJ	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_CHARGE_ RECALC	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_CHARGE_ RECALC	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_CHARGE_ REQUEST	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property

Module	Table	Partition Key	Method	Number of Partitions
BL	BL1_CUSTOMER_CHARGES	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_CYCLE_CUSTOMERS	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_CYCLE_CUSTOMERS	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_CYC_PAYER_POP	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_CYC_PAYER_POP	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_DOCUMENT	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_DOCUMENT	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_HIST_CHARGE_REQUEST	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_HIST_RC_RATES	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_INVOICE	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_INVOICE	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_INV_CHARGE_REL	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_INV_CHARGE_REL	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property

Module	Table	Partition Key	Method	Number of Partitions
BL	BL1_INV_ STATEMENT	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_INV_ STATEMENT	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_NUMBER	PARTITION_KEY	Dynamic	Table purpose is to receive the sequences as a bulk. In each partition 10,000 records to support retrieve of maximum 10,000 sequences in bulk. The partition is for distributing the parallel access to table. Each process retrieves sequences using random partition key. The partitioning is redundant. At the time the thought was that partitions are good also for parallel read access. Therefore number of partitions should be set to 1. Number of partitions is set according to reference table BL1_CONF_SECTION_PARAM. Record: PARAM_CLASS = 'BILLING' , SECTION_NAME = 'config' and PARAM_NAME= 'NumberTablePartitions'. Value of parm in record should be set to 1: PARAM_VALUE='1'
BL	BL1_PAYER_CU ST_REL	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_PREPAID_ STATEMENT	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_PREPAID_ STATEMENT	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_RC_FREQ_ CREATION	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_RC_RATES	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
BL	BL1_SAM_ CONTROL	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_TAX	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_TAX	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property

Module	Table	Partition Key	Method	Number of Partitions
BL	BL1_TAX_ITEM	PERIOD_KEY	Dynamic	Partition per month (<cycle close month> + 12*<cycle close year>) modulo parametersPeriodPartitionDimention property
BL	BL1_TAX_ITEM	CUSTOMER_KEY	Dynamic	CUSTOMER_ID modulo CustomerPartitionDimention property
CCS	CCS1_BATCH_CTRL_HIST	PERIOD_KEY	Dynamic	Partition per month: (Calendar.YEAR *12 + (Calendar.MONTH + 1)) modulo (amdocs.ccs.db.periodKey.partitionCount property + 1). The amdocs.ccs.db.periodKey.partitionCount property default is 10
CCS	CCS1_BATCH_FILE_CTRL_HIST	PERIOD_KEY	Dynamic	Partition per month (Calendar.YEAR *12 + (Calendar.MONTH + 1)) modulo (amdocs.ccs.db.periodKey.partitionCount property + 1). The amdocs.ccs.db.periodKey.partitionCount property default is 10
CCS	CCS1_REQUEST_HIST	REQUEST_KEY	Dynamic	REQUEST_ID modolu amdocs.ccs.db.requestKey.partitionCount property. The amdocs.ccs.db.requestKey.partitionCount property default is 10
CCS	CCS1_REQUEST_HIST	PERIOD_KEY	Dynamic	Partition per month (Calendar.YEAR *12 + (Calendar.MONTH + 1)) modulo (amdocs.ccs.db.periodKey.partitionCount property + 1). The amdocs.ccs.db.periodKey.partitionCount property default is 10
CCS	CCS1_REQ_ACTIVITY_HIST	REQUEST_KEY	Dynamic	REQUEST_ID modulo amdocs.ccs.db.requestKey.partitionCount property
CCS	CCS1_REQ_ACTIVITY_HIST	PERIOD_KEY	Dynamic	Partition per month (Calendar.YEAR *12 + (Calendar.MONTH + 1)) modulo (amdocs.ccs.db.periodKey.partitionCount property + 1). The amdocs.ccs.db.periodKey.partitionCount property default is 10
CCS	CCS1_REQ_DATA_SGMT_HIST	REQUEST_KEY	Dynamic	REQUEST_ID modulo amdocs.ccs.db.requestKey.partitionCount property
CCS	CCS1_REQ_DATA_SGMT_HIST	PERIOD_KEY	Dynamic	Partition per month (Calendar.YEAR *12 + (Calendar.MONTH + 1)) modulo (amdocs.ccs.db.periodKey.partitionCount property + 1). The amdocs.ccs.db.periodKey.partitionCount property default is 10

Module	Table	Partition Key	Method	Number of Partitions
CCS	CCS1_REQ_PARAM_HIST	REQUEST_KEY	Dynamic	REQUEST_ID modulo amdocs.ccs.db.requestKey.partitionCount property
CCS	CCS1_REQ_PARAM_HIST	PERIOD_KEY	Dynamic	Partition per month (Calendar.YEAR *12 + (Calendar.MONTH + 1)) modulo (amdocs.ccs.db.periodKey.partitionCount property + 1). The amdocs.ccs.db.periodKey.partitionCount property default is 10
CCS	CCS1_REQ_XTRCT_CTRL_HIST	REQUEST_KEY	Dynamic	REQUEST_ID modulo amdocs.ccs.db.requestKey.partitionCount property
CCS	CCS1_REQ_XTRCT_CTRL_HIST	PERIOD_KEY	Dynamic	
CL	CL1_ACTIVITY_PARM	ENTITY_KEY	Dynamic	ENTITY_ID column modulo amdocs.cl.db.collectionEntity.partition property
CL	CL1_ACTIVITY_PARM_HIS	ENTITY_KEY	Dynamic	ENTITY_ID column modulo amdocs.cl.db.collectionEntity.partition property
CL	CL1_ACTIVITY_PARM_HIS	END_TREATMENT_KEY	Dynamic	Partition per year. Calendar.YEAR modulo amdocs.cl.db.collectionEntity.partition property
CL	CL1_COLL_ENTITY	ENTITY_KEY	Dynamic	ENTITY_ID column modulo amdocs.cl.db.collectionEntity.partition property
CL	CL1_COLL_ENTITY_HISTORY	ENTITY_KEY	Dynamic	ENTITY_ID column modulo amdocs.cl.db.collectionEntity.partition property
CL	CL1_COLL_ENTITY_HISTORY	END_TREATMENT_KEY	Dynamic	Partition per year Calendar.YEAR modulo amdocs.cl.db.collectionEntity.partition property
CL	CL1_DEBT_BUCKET	ENTITY_KEY	Dynamic	ENTITY_ID column modulo amdocs.cl.db.collectionEntity.partition property
CL	CL1_TREATMENT_ACTIVITY	ENTITY_KEY	Dynamic	ENTITY_ID column modulo amdocs.cl.db.collectionEntity.partition property
CL	CL1_TREAT_ACT_HISTORY	ENTITY_KEY	Dynamic	ENTITY_ID column modulo amdocs.cl.db.collectionEntity.partition property
CL	CL1_TREAT_ACT_HISTORY	END_TREATMENT_KEY	Dynamic	Partition per year Calendar.YEAR modulo amdocs.cl.db.collectionEntity.partition property

Module	Table	Partition Key	Method	Number of Partitions
CM	AGREEMENT_ RESOURCE	AGREEMENT_KE Y	Dynamic	Column AGREEMENT_NO modulo amdocs.cm.partitionKey.entityID.agreement_resource. agreement_key property OR 100 if property is not defined
CM	AGR_RES_ HISTORY	PERIOD_KEY	Dynamic	Partition per month Formula based on date when the history record was created (YEAR of logical_date * 12 + (MONTH of logical_date + 1)) modulo amdocs.cm.partitionKey.period.agr_res_history.period_key property OR 12 if property is not defined + 1
CM	AGR_RES_ HISTORY	AGREEMENT_ KEY	Dynamic	Column AGREEMENT_NO modulo amdocs.cm.partitionKey.entityID.agreement_resource.agreement_key property OR 100 if property is not defined
CM	CM1_AGREEME NT_PARAM	AGREEMENT_ KEY	Dynamic	Column AGREEMENT_NO modulo amdocs.cm.partitionKey.cm1_agreement_param.agreement_key property OR 100 if property is not defined
CM	CM1_AGR_PRM_ HISTORY	PERIOD_KEY	Dynamic	Partition per month Formula based on date when the history record was created: (YEAR of logical_date * 12 + (MONTH of logical_date + 1)) modulo (amdocs.cm.historyMonthsToKeep property + 1)
CM	CM1_AGR_PRM_ HISTORY	AGREEMENT_ KEY	Dynamic	Column AGREEMENT_NO modulo amdocs.cm.partitionKey.cm1_agreement_param.agreement_key property OR 100 if property is not defined
CM	CM_USR_GRP_ MEMBERS	ADMIN_ CONTEXT_KEY	Dynamic	GROUP_ID column OR AGREEMENT_NO column modulo amdocs.cm.partitionKey.cm_usr_grp_members.group_id.partition_key property OR 100 if property is not defined
CM	CM_USR_GRP_ MEMBERS	ADMIN_CONTEX T_ID_KEY	Dynamic	Indicates if the partition key should be according to: GROUP_ID column OR AGREEMENT_NO column Range of values 1 or 0.

Module	Table	Partition Key	Method	Number of Partitions
CM	MEMO	PERIOD_KEY	Dynamic	Partition per month Formula based on date when the memo was created (YEAR of logical_date * 12 + (MONTH of logical_date + 1)) modulo amdocs.cm.partitionKey.period.memo.period_key property OR 12 if property is not defined + 1)
CM	MEMO	ENTITY_KEY	Dynamic	ENTITY_ID column modulo amdocs.cm.partitionKey.entityID.memo.entity_key property OR 100 if property is not defined
MF	MF1_BATCH_CONTROL	BATCH_TYPE	Static	Less than 2, less than 3, less than 4, less than 5 Total 4 partitions
MF	MF1_BATCH_RET_RSN	BATCH_TYPE	Static	Less than 2, less than 3 Total 2 partitions
MF	MF1_DUPCHECK_KEYS	SUB_PARTITION_ID	Dynamic	
MF	MF1_DUPCHECK_KEYS	PERIOD	Dynamic	
MF	MF1_OUTCOL_DAILY_SCHG	SERVE_SID_KEY	Dynamic	
MF	MF1_OUTCOL_DAILY_SCHG	PERIOD_KEY	Dynamic	
MF	MF1_OUTCOL_RATED_EVENT	PERIOD	Dynamic	
MMO	MO1_MEMO	PERIOD_KEY	Dynamic	Partition per month (YEAR of MEMO_DATE column)*12 + (MONTH of MEMO_DATE column +1)) modulo amdocs.mo.partitionKey.period.num_months_to_keep property (MMO1App.properties) + 1)
MMO	MO1_MEMO	ENTITY_KEY	Dynamic	ENTITY_ID column modulo amdocs.mo.partitionKey.entity.num_entity_partitions property (MMO1App.properties)

Module	Table	Partition Key	Method	Number of Partitions
MMO	MO1_MEMO	APP_ID	Dynamic	Application ID of the calling application. Range of values according to reference table MO1_APPLICATION column APP_ID
OLC	OLC1_PROCESSED_TRX	PARTITION_KEY	Dynamic	Column RATING_PARTITION modulo reference table GN1_ART_SECTION_PARM. Relevant record in GN1_ART_SECTION_PARM is: SECTION_NAME = 'SERVER_PARAMETERS' and PARAM_NAME = 'MAX_PARTITION_NUM' and PARAM_CLASS is relevant daemon name (XLA_OLC_BASE, RB_BASE, RBAOC_BASE , RBF_BASE, SYNC_BASE, OXA_BASE)
OLC	OLC1_PROCESSED_TRX	PERIOD_KEY	Dynamic	Partition per hour of day. Range of values between 0 - 23 When inserting record, PERIOD_KEY is calculated according to hour of current time of day. Although table's partitions are defined as dynamic, this partition key must be defined 0 - 23 (Less than 1, less than 2...less than 24).
PM	PERFORMANCE_IND	CYCLE_MONTH	Dynamic	According to the applicative table the PM1_CYCLE_STATE table defines the CYCLE_INSTANCEs for specific CYCLE_CODE. From V6.0 CYCLE_MONTH column in RATED_EVENT is actually CYCLE_INSTANCE
PM	PERFORMANCE_IND	CYCLE_CODE	Dynamic	According to reference table CYCLE_DEFINITION
PM	RATED_EVENT	SUB_PARTITION_ID	Dynamic	CUSTOMER_ID modulo reference table CYCLE_DEFINITION column NUMBER_PARTITIONS for specific column Although can set a different number of partition per cycle in reference tables, the Rater processes do not support this. Number of partitions should be equal to all cycles
PM	RATED_EVENT	CYCLE_MONTH	Dynamic	According to applicative table the PM1_CYCLE_STATE table defines the CYCLE_INSTANCEs for specific CYCLE_CODE. From V6.0 CYCLE_MONTH column in RATED_EVENT is actually CYCLE_INSTANCE
PM	RATED_EVENT	CYCLE_CODE	Dynamic	According to reference table CYCLE_DEFINITION
PM	REJECTED_EVENT	CYCLE_MONTH	Dynamic	According to applicative table the PM1_CYCLE_STATE table defines the CYCLE_INSTANCEs for specific CYCLE_CODE. From V6.0 CYCLE_MONTH column in RATED_EVENT is actually CYCLE_INSTANCE
PM	REJECTED_EVENT	CYCLE_CODE	Dynamic	According to reference table CYCLE_DEFINITION

Module	Table	Partition Key	Method	Number of Partitions
PV	DVC_TRX_CNT	PARTITION_ID	Static	Less than 2, less than 3… less than 11 Total 10 partitions
PV	DVC_TRX_REPOS	PARTITION_ID	Static	Less than 1, less than 2..less than 10
PV	DVC_TRX_REPOS	SRV_TRX_S_MOD	Static	Less than 2, less than 3…less than 6 Total partitions for table 10*5=50
PV	Q1	SPF_NO	Static	Less than 2, less than 3…less than 21 Total partitions for table 20
PV	Q2	MNG_NO	Static	Less than 2, less than 3…less than 21 Total partitions for table 20
PV	Q3	MNG_NO	Static	Less than 2, less than 3…less than 21 Total partitions for table 20
PV	Q4	MNG_NO	Static	Less than 1, less than 2…less than 21, Less than 99 Total partitions for table 22
PV	SRVTRX_GEN_DATA	PARTITION_ID	Static	Less than 1, less than 2..less than 10
PV	SRVTRX_GEN_DATA	SRV_TRX_S_MOD	Static	Less than 2, less than 3…less than 6 Total partitions for table 10*5=50
PV	SRV_SEARCH_REPOS	PARTITION_ID	Static	Less than 1, less than 2…less than 10
PV	SRV_SEARCH_REPOS	SRV_TRX_S_MOD	Static	Less than 2, less than 3…less than 6 Total partitions for table 10*5=50
PV	SRV_TRX_REPOS	PARTITION_ID	Static	Less than 1, less than 2..less than 10
PV	SRV_TRX_REPOS	SRV_TRX_S_MOD	Static	Less than 2, less than 3…less than 6 Total partitions for table 10*5=50
RPL	RPL1_RECOVERY	PERIOD_KEY	Dynamic	Partition per day (Calendar.YEAR * 366 + Days from beginning of the year) modulo (Property DAY_PAR + 1)

Module	Table	Partition Key	Method	Number of Partitions
RPL	RPL1_RECOVERY	RCG_KEY	Dynamic	RCG_ID modulo RCG_MO_PAR property
RPL	RPL1_RECOVERY_DET	PERIOD_KEY	Dynamic	Partition per day (Calendar.YEAR * 366 + Days from begging of the year) modulo (DAY_PAR property + 1)
RPL	RPL1_RECOVERY_DET	RCG_KEY	Dynamic	RCG_ID modulo RCG_MO_PAR property
RPL	RPL1_RECOVERY_ERROR	PERIOD_KEY	Dynamic	Partition per day (Calendar.YEAR * 366 + days from beginning of the year) modulo (DAY_PAR property + 1)
RPL	RPL1_RECOVERY_ERROR	RCG_KEY	Dynamic	RCG_ID modulo RCG_MO_PAR v
RPL	RPL1_RECOVERY_HISTORY	PERIOD_KEY	Dynamic	Partition per day (Calendar.YEAR * 366 + days from beginning of the year) modulo (DAY_PAR property + 1)
RPL	RPL1_RECOVERY_HISTORY	RCG_KEY	Dynamic	RCG_ID modulo RCG_MO_PAR property
RPL	RPL1_RECOVERY_HISTORY_DET	PERIOD_KEY	Dynamic	Partition per day (Calendar.YEAR * 366 + days from beginning of the year) modulo (DAY_PAR property + 1)
RPL	RPL1_RECOVERY_HISTORY_DET	RCG_KEY	Dynamic	RCG_ID modulo RCG_MO_PAR property
RPL	RPM_RCG_ERROR	PERIOD_KEY	Dynamic	Partition per month (Calendar.YEAR * 12 + MONTH of Calendar) modulo (DAY_PAR property + 1)
RPL	RPM_RECHARGES	PERIOD_KEY	Dynamic	Partition per month (Calendar.YEAR * 12 + MONTH of Calendar) modulo (MONTH_PAR property + 1)
RPL	RPM_RECHARGES	BUCKET_KEY	Dynamic	column BUCKET_ID modulo BKT_MO_PAR property
RPL	RPM_RECHARGES_DET	PERIOD_KEY	Dynamic	Partition per month (Calendar.YEAR * 12 + MONTH of Calendar) modulo (MONTH_PAR property + 1)
RPL	RPM_RECHARGES_DET	BUCKET_KEY	Dynamic	column BUCKET_ID modulo BKT_MO_PAR property

Module	Table	Partition Key	Method	Number of Partitions
RPL	RPM_SPLT_RCG_ERR	PERIOD_KEY	Dynamic	Partition per month (Calendar.YEAR * 12 + MONTH of Calendar) modulo (MONTH_PAR property + 1)
RPR	RPR1_SUBS_RERATE	CYCLE_CODE	Dynamic	According to reference table CYCLE_DEFINITION
RPR	RPR1_SUBS_RERATE	CYCLE_INST	Dynamic	According to applicative table the PM1_CYCLE_STATE table defines the CYCLE_INSTANCEs for specific CYCLE_CODE
SEC	SEC1_LOG	PARTITION_ID	Static	Less than 1, less than 2…less than 12 Partition per month: value 0 for January, 1 for February…Value 11 for December Total 12 partitions
TRB	TRB1_MST_LOG	IMP_PERIOD	Dynamic	Partition per week or month Current DATE - BASE DATE (defined in TRB1_PARAM_CONFIG.PARAM_NAME='TRB1_BASE_TIME') / (7 OR 30 – depending on value of TRB1_PARAM_CONFIG.PARAM_NAME='TRB1_IMP_PERIOD_TYPE') modulo PERIOD_CYCLE (defined in TRB1_PARAM_CONFIG.PARAM_NAME='TRB1_IMP_PERIOD_CYCLE)
TRB	TRB1_PUB_LOG	SOURCE_COMP_ID	Dynamic	According to TRB1_MEMBERS.MEMBER_ID reference table Each publisher (defined with TRB1_MEMBERS.MEMBER_MODE='PU') has different MEMBER_ID and directed to different partition Publisher MEMBER_ID values can range from 2000 to 2099
TRB	TRB1_SUB_ERRS	SUB_APPL_ID	Dynamic	According to TRB1_MEMBERS.MEMBER_ID reference table Each subscriber (defined with TRB1_MEMBERS.MEMBER_MODE='SU') has different MEMBER_ID and directed to different partition. Subscriber MEMBER_ID values can range from 3000 to 3099
TRB	TRB1_SUB_LOG	SUB_APPL_ID	Dynamic	According to TRB1_MEMBERS.MEMBER_ID reference table Each subscriber (defined with TRB1_MEMBERS.MEMBER_MODE='SU') has different MEMBER_ID and directed to different partition. Subscriber MEMBER_ID values can range from 3000 to 3099

Module	Table	Partition Key	Method	Number of Partitions
URM	RM1_PACKAGE	PACKAGE_TYPE_ KEY	Dynamic	According to RM_PACKAGE_TYPE reference table RPT_PARTITION_KEY column. At the time of resource type creation, the default value assigned to RM_PACAKGE_TYPE.RPT_PARTITION_KEY is calculated by the formula: (2 * ROWNUM) modulo 100 The default value can be manually overwritten.
URM	RM1_PACKAGE	PACKAGE_ VALUE_KEY	Dynamic	The 8 right-most characters of the PACKAGE_VALUE column The number of the position of each character in this new 8-character string (from left to right) is multiplied by the ASCII code of the character, and the products are summed Example: PACKAGE VALUE: 'LLLABC123456789.ABC' Substring: '6789.ABC' Value key: ASCII ('6') * 1 + ASCII ('7') * 2 +….+ ASCII ('C') * 8
URM	RM1_PACKAGE_ TTRIBUTES	PACKAGE_TYPE_ KEY	Dynamic	According to the RM_PACKAGE_TYPE reference table RPT_PARTITION_KEY column. At the time of resource type creation, the default value assigned to RM_PACAKGE_TYPE.RPT_ PARTITION_KEY is calculated by the formula: (2 * ROWNUM) modulo 100 Default value can be manually overwritten.
URM	RM1_PACKAGE_ ATTRIBUTES	PACKAGE_ VALUE_KEY	Dynamic	The 8 right-most characters of the PACKAGE_VALUE column The number of the position of each character in this new 8-character string (from left to right) is multiplied by the ASCII code of the character, and the products are summed Example: PACKAGE VALUE: 'LLLABC123456789.ABC' Substring: '6789.ABC' Value key: ASCII ('6') * 1 + ASCII ('7') * 2 +….+ ASCII ('C') * 8
URM	RM1_PACKAGE_ CONTENT	PACKAGE_TYPE_ KEY	Dynamic	According to the RM_PACKAGE_TYPE reference table RPT_PARTITION_KEY column. At the time of resource type creation, the default value assigned to RM_PACAKGE_TYPE.RPT_ PARTITION_KEY is calculated by the formula: (2 * ROWNUM) modulo 100 Default value can be manually overwritten.

Module	Table	Partition Key	Method	Number of Partitions
URM	RM1_PACKAGE_CONTENT	PACKAGE_VALUE_KEY	Dynamic	The 8 right-most characters of the column PACKAGE_VALUE The number of the position of each character in this new 8-character string (from left to right) is multiplied by the ASCII code of the character, and the products are summed Example: PACKAGE VALUE: 'LLLABC123456789.ABC' Substring: '6789.ABC' Value key: ASCII ('6') * 1 + ASCII ('7') * 2 +….+ ASCII ('C') * 8
URM	RM1_PACKAGE_HISTORY	PACKAGE_HISTORY_PERIOD_KEY	Dynamic	Partition per month Formula is based on date when activity was issued ((YEAR of logical_date * 12) + (MONTH of logical_date + 1)) modulo (amdocs.rm.partitionKey.period.rm1Package property OR 0 if property is not defined + 1)
URM	RM1_PACKAGE_HISTORY	PACKAGE_TYPE_KEY	Dynamic	According to the RM_PACKAGE_TYPE reference table RPT_PARTITION_KEY column. At the time of resource type creation, the default value assigned to RM_PACAKGE_TYPE.RPT_PARTITION_KEY is calculated by the formula: (2 * ROWNUM) modulo 100 Default value can be manually overwritten.
URM	RM1_PACKAGE_HISTORY	PACKAGE_VALUE_KEY	Dynamic	The 8 right-most characters of the PACKAGE_VALUE column The number of the position of each character in this new 8-character string (from left to right) is multiplied by the ASCII code of the character, and the products are summed Example: PACKAGE VALUE: 'LLLABC123456789.ABC' Substring: '6789.ABC' Value key: ASCII ('6') * 1 + ASCII ('7') * 2 +….+ ASCII ('C') * 8
URM	RM1_REQUEST	REQUEST_PERIOD_KEY	Dynamic	Partition per month Formula is based on date when request was issued. ((YEAR of logical_date * 12) + (MONTH of logical_date + 1)) modulo (amdocs.rm.partitionKey.period.rm1Request.periodkey property OR 0 if property is not defined + 1)
URM	RM1_REQUEST	ENTITY_TYPE_KEY	Dynamic	Column value depends on the type of the entity for which the request is created. If the REQUEST_ID is related to resource, the column value is populated with the value of RM1_RESOURCE.RESOURCE_TYPE_KEY of resource record to which the request is related. If the REQUEST_ID is related to package, the column value is populated with the value of RM1_PACKAGE.PACAKGE_TYPE_KEY of package record to which the request is related .

Module	Table	Partition Key	Method	Number of Partitions
URM	RM1_REQUEST	ENTITY_VALUE_ KEY	Dynamic	Column value depends on the type of the entity for which the request is created. If the REQUEST_ID is related to resource, the column value is populated with the value of RM1_RESOURCE.RESOURCE_VALUE_KEY of resource record to which the request is related. If the REQUEST_ID is related to package, the column value is populated with the value of RM1_PACKAGE.PACAKGE_VALUE_KEY of package record to which the request is related.
URM	RM1_RESOURCE	RESOURCE_TYPE _KEY	Dynamic	According to reference table RM_RESOURCE_TYPE RRT_PARTITION_KEY column. At time of resource type creation the default value assigned to RM_RESOURCE_TYPE.RRT_PARTITION_KEY is calculated by the formula: (2 * ROWNUM) modulo 100 Default value can be manually overwritten.
URM	RM1_RESOURCE	RESOURCE_ VALUE_KEY	Dynamic	The 8 right-most characters of the column RESOURCE_VALUE The number of the position of each character in this new 8-character string (from left to right) is multiplied by the ASCII code of the character, and the products are summed Example: RESOURCE VALUE: 'LLLABC123456789.ABC' Substring: '6789.ABC' Value key: ASCII ('6') * 1 + ASCII ('7') * 2 +….+ ASCII ('C') * 8
URM	RM1_RESOURCE _ ATTRIBUTES	RESOURCE_TYPE _KEY	Dynamic	According to the RM_RESOURCE_TYPE reference table RRT_PARTITION_KEY column. At time of resource type creation, the default value assigned to RM_RESOURCE_TYPE.RRT_PARTITION_KEY is calculated by the formula: (2 * ROWNUM) modulo 100 Default value can be manually overwritten.
URM	RM1_RESOURCE _ ATTRIBUTES	RESOURCE_ VALUE_KEY	Dynamic	The 8 right-most characters of the RESOURCE_VALUE column The number of the position of each character in this new 8-character string (from left to right) is multiplied by the ASCII code of the character, and the products are summed Example: RESOURCE VALUE: 'LLLABC123456789.ABC' Substring: '6789.ABC' Value key: ASCII ('6') * 1 + ASCII ('7') * 2 +….+ ASCII ('C') * 8
URM	RM1_RESOURCE _ HISTORY	RESOURCE_ HISTORY_PERIOD _KEY	Dynamic	Partition per month Formula is based on date when the activity was issued ((YEAR of logical_date * 12) + (MONTH of logical_date + 1)) modulo (Property amdocs.rm.partitionKey.period.rm1Resource.periodkey OR 0 if property is not defined + 1)

Module	Table	Partition Key	Method	Number of Partitions
URM	RM1_RESOURCE_HISTORY	RESOURCE_TYPE_KEY	Dynamic	According to reference table RM_RESOURCE_TYPE column RRT_PARTITION_KEY. At resource type creation default value assign to RM_RESOURCE_TYPE.RRT_PARTITION_KEY by formula: (2 * ROWNUM) modulo 100 Default value can be manually overwritten.
URM	RM1_RESOURCE_HISTORY	RESOURCE_VALUE_KEY	Dynamic	8 right-most characters of the column RESOURCE_VALUE The number of the position of each character in this new 8-character string (from left to right) is multiplied by the ASCII code of the character, and the products are summed Example: RESOURCE VALUE: 'LLLABC123456789.ABC' Substring: '6789.ABC' Value key: ASCII ('6') * 1 + ASCII ('7') * 2 +….+ ASCII ('C') * 8
VM	VM1_PACKAGE	EXP_DATE_PARTITION	Dynamic	Based on the EXP_DATE column. Each partition group records of a PERIOD that can be a month, 2 months, quarter, etc. Number of months grouped together to one partition is determined by the amdocs.vm.packageExpDate PartitionMonthGroupSize property in VM1App.Properties. The amdocs.vm.packageExpDate PartitionCycle property sets the number of years packages remain in the system.
VM	VM1_PACKAGE_HISTORY	EXP_DATE_PARTITION	Dynamic	Based on the EXP_DATE column. Each partition group records of a PERIOD that can be month, 2 months, quarter, etc. Number of months grouped together to one partition is determined by the amdocs.vm.packageExpDate PartitionMonthGroupSize propertyin VM1App.Properties. The amdocs.vm.packageExpDate PartitionCycle property sets the number of years packages remain in the system.

Module	Table	Partition Key	Method	Number of Partitions
VM	VM1_TENT_VCH	ID_PARTITION	Dynamic	Formula is modulo on column SERIAL_NO or PIN. The amdocs.vm.voucherId PartitionSource property in VM1App.properties defines on which column the modulo is calculated. Valid values for this property are SERIAL_NO / PIN Since the VOUCHER_PIN_CODE is saved in hexadecimal, the characters will be translated to numbers before the mathematic calculation. A=1, B=2, C=3, D=4, E=5, F=6. Whether formula is based on SERIAL_NO or PIN, first the significant last digits are extracted Number of significant digits is length of the modulo + 1 (for value of 1 the result is 2, for value of 10 the result is 3, for value 100 the result is 4) Then the calculation is significant last digits modulo (amdocs.vm.voucherIdPartitionModule property in VM1App.properties + 1)
VM	VM1_TENT_VCH	EXP_DATE_ PARTITION	Dynamic	Based on column EXP_DATE. Each partition group records of a PERIOD that can be month, 2 months, quarter, etc. Number of months grouped together to one partition is determined by the amdocs.vm.tentVchExpDate PartitionMonthGroupSize property in VM1App.Properties. The amdocs.vm.tentVchExpDatePartitionCycle property sets the number of years tentative vouchers remain in the system.
VM	VM1_TENT_VCH _ HISTORY	ID_PARTITION	Dynamic	Formula is modulo on column SERIAL_NO or PIN. The amdocs.vm.voucherIdPartition Source property in VM1App.properties defines on which column the modulo is calculated. Valid values for this property are SERIAL_NO / PIN Since the VOUCHER_PIN_CODE is saved in hexadecimal, the characters will be translated to numbers before the mathematic calculation. A=1, B=2, C=3, D=4, E=5, F=6. Whether formula is based on SERIAL_NO or PIN, first the significant last digits are extracted Number of significant digits is length of the modulo + 1 (for value of 1 the result is 2, for value of 10 the result is 3, for value 100 the result is 4) Then the calculation is significant last digits modulo (amdocs.vm.voucherIdPartitionModule property in VM1App.properties + 1)

Module	Table	Partition Key	Method	Number of Partitions
VM	VM1_TENT_VCH_HISTORY	EXP_DATE_PARTITION	Dynamic	Based on column EXP_DATE. Each partition group records of a PERIOD that can be month, 2 months, quarter, etc. Number of months grouped together to one partition is determined by the amdocs.vm.tentVchExpDate PartitionMonthGroupSize property in VM1App.Properties. The amdocs.vm.tentVchExpDatePartitionCycle property sets the number of years tentative vouchers remain in the system.
VM	VOUCHER	ID_PARTITION	Dynamic	Formula is modulo on column VOUCHER_SERIAL_NO or VOUCHER_PIN_CODE. The amdocs.vm.voucherId PartitionSource property in VM1App.properties defines on which column the modulo is calculated. Valid values for this property are: VOUCHER_SERIAL_NO / VOUCHER_PIN_CODE Since the VOUCHER_PIN_CODE is saved in hexadecimal, the characters will be translated to numbers before the mathematic calculation. A=1, B=2, C=3, D=4, E=5, F=6. Whether the formula is based on VOUCHER_SERIAL_NO or VOUCHER_PIN_CODE, first the significant last digits are extracted Number of significant digits is length of the modulo + 1 (for value of 1 the result is 2, for value of 10 the result is 3, for value 100 the result is 4) Then the calculation is significant last digits modulo (amdocs.vm.voucherIdPartitionModule property in VM1App.properties + 1)
VM	VOUCHER	EXP_DATE_PARTITION	Dynamic	Based on column EXP_DATE. Each partition group records of a PERIOD that can be month, 2 month, quarter etc. Number of months grouped together to one partition is determined by property: amdocs.vm.voucherExpDate PartitionMonthGroupSize in VM1App.Properties. The property amdocs.vm.voucherExpDatePartitionCycle property sets the number of years vouchers remain in the system.

Module	Table	Partition Key	Method	Number of Partitions
VM	VOUCHER_ HISTORY	ID_PARTITION	Dynamic	Formula is modulo on column VOUCHER_SERIAL_NO or VOUCHER_PIN_CODE. The amdocs.vm.voucherIdPartitionSource property in VM1App.properties defines on which column the modulo is calculated. Valid values for this property are VOUCHER_SERIAL_NO / VOUCHER_PIN_CODE Since the VOUCHER_PIN_CODE is saved in hexadecimal, the characters will be translated to numbers before the mathematic calculation. A=1, B=2, C=3, D=4, E=5, F=6. Whether formula is based on VOUCHER_SERIAL_NO or VOUCHER_PIN_CODE, first the significant last digits are extracted Number of significant digits is length of the modulo + 1 (for value of 1 the result is 2, for value of 10 the result is 3, for value 100 the result is 4) Then the calculation is significant last digits modulo (amdocs.vm.voucherIdPartition Module property in VM1App.properties + 1)
VM	VOUCHER_ HISTORY	EXP_DATE_ PARTITION	Dynamic	Based on column EXP_DATE. Each partition group records of a PERIOD that can be month, 2 months, quarter, etc. Number of months grouped together to one partition is determined by the amdocs.vm.voucherExpDate PartitionMonthGroupSize property in VM1App.Properties. The amdocs.vm.voucherExpDate PartitionCycle property sets the number of years vouchers remain in the system.