

Retail Transaction Processor

*System Design*

Version 2.03 - DRAFT

Release History

Author: Ronald Colangelo

Date Created: 09/03/2001

Date Last Modified: 12/27/2007

**Copyright Notice**

This document (written or otherwise displayed on magnetic media or other medium) contains confidential and proprietary information of The Walt Disney Company or its affiliates.

It has been furnished only for informational purposes, and no license or permission is hereby granted to use such information in any manner.

In no event may this information be reproduced, distributed and/or publicly displayed in any form or by any means without prior expressed written permission of Walt Disney World Co.   
©Disney. All rights reserved.

Table of Contents

[1. Introduction 1](#_Toc179173244)

[1.1 Key Principles 2](#_Toc179173245)

[1.2 Document Layout 2](#_Toc179173246)

[2. Conceptual Data Flow 3](#_Toc179173247)

[2.1 Batch 3](#_Toc179173248)

[2.1.1 Physical File Structure 4](#_Toc179173249)

[2.1.1.1 Base Sequence Definition 4](#_Toc179173250)

[2.1.1.2 Oversized Transactions 4](#_Toc179173251)

[2.1.2 Logical Batch Validation 5](#_Toc179173252)

[2.1.3 Batch Transaction Submission 5](#_Toc179173253)

[2.2 Transaction 6](#_Toc179173254)

[2.2.1 Field Loader and Edits 6](#_Toc179173255)

[2.2.1.1 Edits 7](#_Toc179173256)

[2.2.1.2 Field Loader Example 7](#_Toc179173257)

[2.2.2 Check Point Management 8](#_Toc179173258)

[2.2.2.1 Check Point Hierarchy 9](#_Toc179173259)

[2.2.2.2 Transaction Modes 9](#_Toc179173260)

[2.2.2.3 Business Management Information 10](#_Toc179173261)

[2.2.3 Audit 10](#_Toc179173262)

[2.2.4 Data Integrity Management 11](#_Toc179173263)

[2.2.5 Transaction Repository 11](#_Toc179173264)

[2.3 Translation 11](#_Toc179173265)

[2.3.1 Transaction Repository 12](#_Toc179173266)

[2.3.2 Transaction Loader 13](#_Toc179173267)

[2.3.3 Transaction Analyzer 13](#_Toc179173268)

[2.3.4 Real-time Host Interface 13](#_Toc179173269)

[2.3.5 Batch Host Interface 14](#_Toc179173270)

[2.4 Batch versus Real-time Implementation 14](#_Toc179173271)

[3. Application Process Model 15](#_Toc179173272)

[3.1 Daemon 15](#_Toc179173273)

[3.1.1 Initialization 15](#_Toc179173274)

[3.1.2 Business Management 15](#_Toc179173275)

[3.1.3 Input Configuration Management 15](#_Toc179173276)

[3.1.4 Process Management 15](#_Toc179173277)

[3.1.5 Support Infrastructure 16](#_Toc179173278)

[3.2 Batch Process 16](#_Toc179173279)

[3.2.1 File Processing 16](#_Toc179173280)

[3.2.2 Transaction Processing 16](#_Toc179173281)

[3.3 Exception Management Unload 16](#_Toc179173282)

[3.4 Support Tool 16](#_Toc179173283)

[4. Application Software Structure 17](#_Toc179173284)

[4.1 RTPINIT 17](#_Toc179173285)

[4.1.1 Command Line Parameters 17](#_Toc179173286)

[4.1.2 Basic Application Flow 18](#_Toc179173287)

[4.1.3 Class Definitions 18](#_Toc179173288)

[4.1.3.1 CCmdLine Class 18](#_Toc179173289)

[4.1.3.2 CDirMgmt Class 19](#_Toc179173290)

[4.1.3.3 CSupport Class 20](#_Toc179173291)

[4.1.3.4 CNmPipe Class 21](#_Toc179173292)

[4.1.3.5 CMaster Class 22](#_Toc179173293)

[4.1.3.6 CINIDecode Class 23](#_Toc179173294)

[4.1.3.7 CDBLOps Class 23](#_Toc179173295)

[4.1.3.8 CCfgHost Class 24](#_Toc179173296)

[4.1.3.9 CCfgLocation Class 25](#_Toc179173297)

[4.2 RTPBATCH – Batch Stage 25](#_Toc179173298)

[4.2.1 Command Line Parameters 25](#_Toc179173299)

[4.2.2 Basic Application Flow 26](#_Toc179173300)

[4.2.3 Class Definitions 26](#_Toc179173301)

[4.2.3.1 CLogicalBatch Class 26](#_Toc179173302)

[4.2.3.2 CTransFile Class 27](#_Toc179173303)

[4.2.3.3 CDbLOps Class 28](#_Toc179173304)

[4.3 RTPBATCH – Transaction Stage 28](#_Toc179173305)

[4.3.1 Class Definitions 29](#_Toc179173306)

[4.3.1.1 CTransact Class 29](#_Toc179173307)

[4.4 RTPDTRAN 29](#_Toc179173308)

[4.5 RTPEMU 29](#_Toc179173309)

[4.6 RTPDEBUG 29](#_Toc179173310)

[4.6.1 Command Line Parameters 30](#_Toc179173311)

[4.6.2 Interactive Commands 30](#_Toc179173312)

[5. Data Structures 32](#_Toc179173313)

[5.1 Special Transactions 32](#_Toc179173314)

[5.1.1 Logical Batch Open (LBO) 32](#_Toc179173315)

[5.1.2 Logical Batch Close (LBC) 32](#_Toc179173316)

[5.2 Inter-process Communication 33](#_Toc179173317)

[5.2.1 Master Shared Memory Segment 33](#_Toc179173318)

[5.2.2 Screen FIFO Protocol 35](#_Toc179173319)

[5.2.3 Log FIFO Protocol 36](#_Toc179173320)

[5.2.4 Command FIFO Protocol 37](#_Toc179173321)

[5.3 Environment Variables 39](#_Toc179173322)

[5.4 Physical Non-database Files 39](#_Toc179173323)

[5.4.1 Non-volatile Transaction Repository 39](#_Toc179173324)

[5.4.2 Suspense Repository 39](#_Toc179173325)

[5.4.3 Initialization Files (INI) 39](#_Toc179173326)

[5.4.3.1 RTPSUPPORT – Support Infrastructure Parameters 40](#_Toc179173327)

[5.4.3.2 RTPINIT – System Management Parameters 46](#_Toc179173328)

[5.4.3.3 RTPLOGIC – Logical Batch Configuration Parameters 50](#_Toc179173329)

[5.4.3.4 RTPBATCH– Batch Configuration Parameters 51](#_Toc179173330)

[5.4.3.5 RTPDTRAN – Data Driven Translator Configuration Parameters 52](#_Toc179173331)

[5.4.3.6 RTPOPENV – OpenView Interface Parameters 53](#_Toc179173332)

[5.4.3.7 RTPMON – RTP Monitoring Parameters 54](#_Toc179173333)

[5.4.3.8 RTPMQSERIES – MQ Series Support Interface Parameters 56](#_Toc179173334)

[5.4.3.9 RTPCMMQ – Communication Manager MQ Series Parameters 58](#_Toc179173335)

[5.4.3.10 RTPWRAP – Wrap Conversion Utility Parameters 59](#_Toc179173336)

[5.4.3.11 RTPWRAP – Comma Delimited File Support 60](#_Toc179173337)

[5.4.3.12 RTPRENG – Reporting Engine Control Parameters 61](#_Toc179173338)

[5.4.3.13 RTPEXFIX – Batch Exception Fix Parameters 62](#_Toc179173339)

[5.4.3.14 RTPTLTS – TLTS Processor Control Parameters 62](#_Toc179173340)

[5.4.3.15 RTPCM – Communication Manager Control Parameters 63](#_Toc179173341)

[5.4.3.16 RTPKEYSRV – Key Service Control Parameters 64](#_Toc179173342)

[5.4.4 Boot Log 65](#_Toc179173343)

[5.4.5 Database Control Information 65](#_Toc179173344)

[5.4.6 Next Event Recovery File 65](#_Toc179173345)

[5.5 Logical Database Design 65](#_Toc179173346)

[5.5.1 Operational Log 65](#_Toc179173347)

[5.5.2 Exception Management 70](#_Toc179173348)

[5.5.3 Reporting Engine 74](#_Toc179173349)

[5.5.4 Configuration 78](#_Toc179173350)

[5.5.4.1 Input Format Definitions 78](#_Toc179173351)

[5.5.4.2 Host/Location Definitions 86](#_Toc179173352)

[5.5.4.3 System Internal Use Definitions 93](#_Toc179173353)

[5.5.4.4 Output Definitions 94](#_Toc179173354)

[5.5.4.5 Bundle Definitions 101](#_Toc179173355)

[5.5.4.6 Unload Transaction Utility 109](#_Toc179173356)

[5.5.4.7 File Delivery System Definitions 112](#_Toc179173357)

[5.5.4.8 Special Translation Table Entries 115](#_Toc179173358)

[5.5.5 Item Maintenance 115](#_Toc179173359)

[5.5.6 Mathematical Expressions 118](#_Toc179173360)

[5.5.6.1 String Results 118](#_Toc179173361)

[5.5.6.2 Numeric Results 119](#_Toc179173362)

[5.5.6.3 Filename Results 121](#_Toc179173363)

[5.5.6.4 Conditional Results 122](#_Toc179173364)

[5.5.6.5 Input Results 123](#_Toc179173365)

[5.5.6.6 Input Numeric Results 124](#_Toc179173366)

[5.5.7 User 125](#_Toc179173367)

[6. File Summary 129](#_Toc179173368)

[6.1 Source Code – Core Program Modules 129](#_Toc179173369)

[6.2 Source Code – Common Modules 129](#_Toc179173370)

[6.3 Source Code – Data Structure Definition Modules 131](#_Toc179173371)

[6.4 Source Code – Database Modules 131](#_Toc179173372)

[6.5 Code Generation 132](#_Toc179173373)

[7. Error Definitions 133](#_Toc179173374)

[7.1 Log File Module Name Definitions 159](#_Toc179173375)

# Introduction

The *Retail Transaction Processor (RTP)* is designed to provide all merchandise/food and beverage selling devices with a single point of interface to the backend systems at Walt Disney World (WDW). Figure 1 provides a high-level overview of RTP. Systems on the left are the selling devices, at time of publication, planned to feed RTP transactions. Backend systems on the right are updated by information provided to RTP. Future selling devices may include kiosks, in-room selling devices, as well as RF handheld devices used across property.



Figure 1.Retail Transaction Processor Top Level Overview

RTP is built on the assumption that transactions are processed independently and fed as required to the appropriate backend systems. Initially, the system will process data every 15 minutes as part of a batch system. By 2004, the system will be converted to handle transactions individually in real-time allowing for real-time management of financials and inventory as well as increasing the available processing capacity by handling the transactions throughout the business day. This document provides a technical design for the batch version of RTP being developed as part of the Pop Century Combined POS implementation. User interfaces are documented in a separate document titled *RTP User Interface Design*. Recommendations are included where appropriate for converting the system from this initial batch form into a real-time transaction processor.

## Key Principles

The following key principles were used as a basis for the design of this system:

* **Data Driven Input** – The input data format should be driven by metadata within a database. The system should simultaneously support multiple input formats allowing for easy rollout of new transactions and field data. Data driven input will eliminate the need for code modifications to support new data fields and/or selling device functionality.
* **Data Driven Validation** – Information should only be passed to host systems if it meets minimum standards (e.g., no single item sale over $1M, valid date information, etc…). The standards should be defined within a database to prevent the need for code modification to change validation rules. Failed transactions need to be stored in a form that the users without IT involvement can review, correct, and resubmit transactions for processing.
* **Data Driven Output** – Output record formats, content, and delivery method should be defined by information within the database. This approach allows for addition of new systems, new data, and modification of existing fields without code modifications.
* **Performance** – In order to facilitate batch load of the GSA transactions, the system will need to be capable of processing at least 150,000 transactions in less than 30 minutes. The resulting sustained throughput rate is at least 83 transactions per second.
* **Modular Structure** – Although initially developed as a batch system all software should be developed in a modular form for eventual transition to a real-time environment with minimal code loss. After review with architecture, the development framework was defined as C++ under Unix using Mastreo and shell scripts for the initial batch implementation.

## Document Layout

The remaining portions of this document define the general design of the RTP application. Chapter 2 is a conceptual overview of the operation of RTP designed for both technical and non-technical readers. It defines in general terms how and where information will be processed. Chapter 3 is a technical overview of how the system will be implemented. It provides a general structure of the processes, inter-process communication, and overall data storage required to support RTP functionality. Chapter 4 provides a detailed analysis of the software structure. It identifies anticipated class and module definitions along with basic process flows for each of the applications that make up the RTP system. Chapter 5 provides detailed information about the logical data structures within the files, databases, and inter-process communication.

Finally, the remaining chapters of this document are designed to assist with the on-going support of this application. Although introduced during design, it is anticipated that these chapters will be heavily modified throughout the development and implementation processes. Chapter 6 provides a list of source files and their content while chapter 7 contains definitions of messages produced by the system including recommended corrective course of action.

# Conceptual Data Flow

Conceptually, RTP contains three functional stages through which information is passed as shown in Figure 2. The first stage *Batch* is only used for selling devices that submit information in one or more batch files. During the initial implementation, all TLOG information will be processed through this stage. After real-time conversion, this stage will only be used for selling devices reporting through a batch process such as Special Events and Disney Cruise Line.



Figure 2. Data Flow Stages

The *Transaction* stage processes individual transactions verifying transaction sequence, structure, and key content (e.g., field is numeric, line item dollar amount less than $ 1M, etc…). Transactions are then either written to an exception database for manual review or reformatted into a standard tag-value format. The final stage of this conceptual model is the *Translation* stage. Multiple instances of the *translation* stage analyze the transactions sequentially, collect data as appropriate, and generate the necessary records to feed the corresponding host system. One instance exists for each host system potentially requiring information from the selling devices.

The following sections provide detailed information on each of the conceptual stages. Sections are organized from the start of the data flow through delivery to the host systems.

## Batch

The *Batch* stage of the conceptual system data flow is primarily responsible for accepting information from systems that are not capable of real-time transaction submissions and feeding them into the real-time environment as individual transactions. Figure 3 provides a basic overview of this stage which is initiated either by the successful receipt of an input file or based on a planned time interval (e.g., every 15 minutes).



Figure 3. Batch Stage Conceptual Data Flow

### Physical File Structure

Fundamentally, each input file consists of a set of one or more individual transactions. Utilizing ASCII characters only, each transaction consists of a standard header, known as a *base sequence*, followed by a data block whose format and content is specific to the type of transaction. Analysis done within this stage is based only on the *base sequence* information except in the case of the two *Logical Batch Validation* transactions as described in section 2.1.2. This design approach allows the modification and addition of new transactions without requiring code changes to any software within this stage of the system. Detailed transaction specifications are contained within the document labeled *Retail Transaction Processor – Batch File Specification*.

#### Base Sequence Definition

The *base sequence* formatis standard for all transactions accepted by RTP for processing. The following table defines the *base sequence* format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Byte Position | Data Type | Name | Description | Case Sensitive |
| 0 – 0 | Char(1) | Start Marker | Must always contain ‘@’ marking the start of a new transaction. | Y |
| 1 – 3 | Char(3) | Transaction Code | Three character alphanumeric code that uniquely defines the transaction being executed. The first position of this field must be a value A-Z. | N |
| 4 – 7 | Hex  0xFFFF | Data Length | Contains the number of bytes in this transaction including the 10-byte *Base Sequence*. A value of 0xffff indicates that the size is greater than 65,534 bytes. Additional information on transactions 64K or larger is documented in section 2.1.1.2. | N |
| 8 – 9 | Char(2) | Version | A value that uniquely identifies the format of the transaction. This field will allow RTP the ability to simultaneously support multiple transaction formats. | Y |

Transactions are pulled from this file by first reading the fixed length *base sequence* and then using the information contained within the header to read the data block as a set of bytes.

#### Oversized Transactions

In version 00 of the batch file specification, it is not anticipated that very large transactions will be used. They are, however, considered during the system design to ensure that the system can support large transactions in the future without redevelopment of the core software. If a transaction *base sequence* contains 65,535 in the data length field, the system reads an additional six bytes that make up the remaining components of the *extended base sequence* as defined in the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Byte Position | Data Type | Name | Description | Case Sensitive |
| 0 – 0 | Char(1) | Start Marker | Must always contain ‘@’ marking the start of a new transaction. | Y |
| 1 – 3 | Char(3) | Transaction Code | Three character alphanumeric code that uniquely defines the transaction being executed. The first position of this field must be a value A-Z. | N |
| 4 – 7 | 0xFFFF | Data Length | Field always contains the fixed value of 0xffff. | N |
| 8 – 9 | Char(2) | Version | A value that uniquely identifies the format of the transaction. This field will allow RTP the ability to simultaneously support multiple transaction formats. | Y |
| 10 – 11 | Hex 0xFF | Buffer Count | A value that is multiplied by 65,536 and added to the data length and the adjustment field to determine the overall byte size of the transaction. | N |
| 12 – 15 | Hex 0xFFFF | Adjustment | A value that is used to determine the exact size of a transaction that is 64K or larger. | N |

The calculation used to determine the overall size of this transaction, including the 16 bytes for the *extended base sequence*, is as follows:

*Transaction Size* = 65,535 + [Buffer Count] \* 65,536 + [Adjustment]

Using this formula and based on the initial implementation, RTP can support a single transaction containing a data block and extended base sequence of up to 16M bytes.

### Logical Batch Validation

The *Logical Batch Validation* component of the *Batch* stage data flow is designed to ensure that the information being received within the file is complete prior to processing any individual transactions. This function is accomplished through the implementation of a pair of special transaction entitled *Logical Batch Open (LBO)* and *Logical Batch Close (LBC)[[1]](#footnote-1)*. These transactions alone are unique in that they have the following characteristics:

* Format is not driven by metadata within the database
* Transactions are only recognized by the batch process
* Transactions do not have a real-time equivalent
* Transactions are not passed to any other stages of the conceptual data flow model

Each *LBO* transaction marks the beginning of a set of transactions all of which occurred on a single business date. Minimal information is contained within the data block of this transaction. Once received, the *Logical Batch Validation* component begins sequentially counting transactions recording the number overall, number of *Store Open*, and number of retail transactions contained within this logical batch. This process continues until a matching *LBC* transaction is encountered.

The *LBC* transaction marks the end of the set of transactions identified by the corresponding *LBO*. The data block for the *LBC* transaction provides validation information. Each submitting selling device provides corresponding record counts that can be compared to the above-calculated values to validate the integrity of transaction set. If the validation is successful, then the transactions between the *LBO* and *LBC (non-inclusive)* are passed on to the *Batch Transaction Submission* component for further processing. If validation fails, then the batch is written to a support directory for further review. In either case, the operational log is updated to record the processing of this information.

### Batch Transaction Submission

The set of transactions received from the *Logical Batch Validation* component are sequentially processed. Each one is read and submitted to the *Transaction* stage of the data flow for verification and acceptance. The following results can occur with each transaction submission:

* Transaction is accepted fully – no further action is required by this component.
* Transaction is rejected as a duplicate – no further action is required by the component.
* Transaction is rejected as out of sequence – either the system waits and tries resubmitting the transaction after a specified time interval or the transaction is rejected as described in the next result description. Configuration or command line parameters dictate the selected action.
* Transaction is rejected – this information is logged in the operational log and the transaction is written to the exception management system.

The successful processing of all transactions within this logical set is recorded into the system’s operational log. This information is used for support and can be used to prevent this stage from reprocessing entire batches that have previously been processed through the system.

## Transaction

The *Transaction* stage of the conceptual system data flow is primarily responsible for accepting individual transactions, validating basic format, ensuring transactions are only processed once, validating audit rules, and finally storing the transactions in a safe recoverable format. This stage of the system is completely driven by information contained within the configuration database. This design allows the system to accept new data formats without code changes while simultaneous providing support for multiple input formats. Figure 4 provides a basic overview of this stage that is initiated upon receipt of a transaction.



Figure 4. Transaction Stage Conceptual Data Flow

### Field Loader and Edits

The purpose of the field loader is to receive a byte buffer that represents a transaction starting with the *base sequence* information defined in section 2.1.1 and convert it into the link list based information format used throughout the remaining components of RTP. Data within the transaction buffer is analyzed based on input format definitions within the configuration database. Figure 5 shows a potential multi-chain link list that could result from a *Field Loader* conversion. The first list, represented by the blue/cyan colored blocks on the left, describe each of the record types within the transaction. All transactions have a “@0” record type that contains the top-level field information. Multiple record transactions would then include a single entry for each additional record type in alphabetical order.

The second list, represented by the pink second column of blocks, describes the number of individual instances for each record type. For example, if the above figure represented a retail transaction where the “AA” record type described line items then instance 01 and 02 would describe the first and second item on the ticket, respectively. The remaining green boxes represent the third and final list. One entry exists in each of these lists for each individual field defined in the configuration database for the associated record type. During processing, these lists are stored in a byte buffer within a class. Methods will be provided within the class to quickly locate and retrieve the data associated with a given field name.



Figure 5. Internal Link List Structure

#### Edits

During the *Field Loader* conversion process, field information is verified against a series of edits. Edits are field specific definitions designed to ensure the integrity of the transaction being submitted to the host systems. Edit definitions describe characteristics such as number of bytes, numeric versus alphanumeric, minimum/maximum values, field justification, or padding. All edits are enforced at the individual field level without any reference to any other field within the transaction and/or system. Edit failures do not prevent the field **tag/data** information from being passed on to subsequent components. Instead, the *Field Loader and Edits* component sends one or more **control** records with the field **tag/data** to the *Check Point Management* component. This design approach is used to prevent multiple submissions of a single faulty transaction from being stored in the exception management database since the *Check Point Management* component handles making sure transactions are processed only once.

#### Field Loader Example

The following is an example of how data would flow through the *Field Loader and Edits* component in the RTP *Transaction* stage. Assume the following is an input buffer for a transaction coded “AAA” which simply records a single operator ID.

Buffer Position

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  | 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| @ | A | A | A | 0 | 0 | 2 | 6 | 0 | 0 | 1 | 2 | 1 | 6 | 2 | 0 | 0 | 1 | 5 | R | 5 | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 2 |  |  |  |  |

Example Data

The following table defines the transaction structure along with the necessary field level edits. This information would be contained within the *Configuration* database table as defined in section 5.5.4.1 of this document.

|  |  |  |  |
| --- | --- | --- | --- |
| Byte Position | Name | Description | Edit Definitions |
| 0 – 9 | Base Sequence | Fixed Value “@AAA002600”. | N/A |
| 10 – 17 | Business Date | Business date associated with this transaction. Business date is typically updated at each nightly close to the next valid date. Business date does not necessarily rollover at mid-night. The format is MMDDYYYY. | Date Format MMDDYYYY must be valid |
| 18 – 22 | Location Number | A five character alphanumeric identifier that uniquely identifies the selling location. Valid values are defined as selling locations open. | $$$$$ - Five alphanumeric characters no spaces allowed |
| 23 – 26 | Register/ Terminal ID | A numeric value that uniquely identifies the physical terminal at a location used to capture this data. | ### - Four numeric values zero filled |
| 27 – 31 | Sequence Number | A number that uniquely identifies a transaction for a given location and register combination. The value must be sequential and concurrent starting with one for a given location/register on a single business day. | ##### - Five numeric values zero filled |
| 32 – 37 | Operator ID | A numeric field that uniquely identifies a user within a location. | %%%%%% - Six numeric values left justified padded with spaces |

The sample input buffer would pass all edits defined for this transaction. Consequently, the component would convert the information into a series of linked lists in the following **tag/data** structure without any **control** information. Record label “@0” is always used to identify the header record that always has one and only one instance identified as “00”.

|  |  |  |
| --- | --- | --- |
| Record “@0” | Instance “00” | Field [Business Date] = “20011216” |
|  |  | Field [Location Number] = “5R520” |
|  |  | Field [Register/Terminal ID] = 1 |
|  |  | Field [Sequence Number] = 10 |
|  |  | Field [Operator ID] = 12 |

### Check Point Management

The purpose of the *Check Point Management* component is to ensure that all transactions are processed in the order in which they physically occurred, they are processed once and only once, and they are only processed if operationally appropriate (e.g., we can not sell merchandise before a store is open). This component is only responsible for handling the verification of the transaction and either storage within the *Exception Management* database or passing **tag/data** information to the next component, as appropriate. All other control updates and responses are handled by the *Data Integrity Management* component. The specific checks handled within *Check Point Management* are as follows:

* Validate that the transaction has not previously[[2]](#footnote-2) been processed
* Validate that the transaction is the next transaction expected for a given physical selling device unless the transaction represents an *Exception Management* resubmission[[3]](#footnote-3).
* Validate that all *Exception Management* resubmissions are processed once and only once on a given posting/processing date.
* Submit control records to the *Data Integrity Management* component for any edit failures.
* Validate that the “transaction mode”, as defined in section 2.2.2.2, is appropriate for the transaction being processed.

The purpose of these checks is to ensure data integrity and facilitate easier system restarts. The remaining subsections under *Check Point Management* conceptually define the controls used during the validation processes described within this section.

#### Check Point Hierarchy

In order to provide maximum flexibility, the structure of the check point information is driven by four distinct definitions within the configuration database. These definitions control which fields are used for grouping information, defining the hierarchal associate between groupings, and specification of which field(s) control the order in which transactions are processed. The four definitions in hierarchical order are as follows:

* **Chronological Grouping** – the field(s) associated with this grouping are used to combine transactions that occurred during a specific business time period (e.g., a single business date). The content of these fields can be any valid alphanumeric characters.
* **Physical Location Grouping** – the field(s) associated with this grouping are used to combine transactions that occurred at a single physical selling location. The content of these fields can be any valid alphanumeric characters.
* **Physical Device Grouping** – the field(s) associated with this grouping are used to combine transactions from a single physical selling device (e.g., POS terminal). The content of these fields can be any valid alphanumeric characters.
* **Sequence Control** – the field or fields associated with this definition are used to ensure that all transactions are processed in the order in which they physically occurred. All data within these fields must be numeric with an initial value containing all zeros. Subsequent transactions will have sequential values until the specific business period as defined by the **Chronological Grouping** is complete.

The first three values described above constitute the validation grouping used in check point management. The combination of all four values constitutes a transaction key. In a real-time environment, data integrity will be insured through the use of this transaction key. Each key value will uniquely identify a specific physical event being recorded (e.g., store open, sale, etc…). Each transaction key **must** be unique across all of property. The key fields in hierarchical order for the initial implementation are expected to be as follows:

**Chronological Grouping**

* Business Date

**Physical Location Grouping**

* Location Number

**Physical Device Grouping**

* Register/Terminal ID

**Sequence Control**

* Sequence Number

#### Transaction Modes

The “transaction mode" concept within RTP is designed to allow configuration-based control over the sequence in which transactions are accepted for processing. For example, the system can be configured in such a way that a store open transaction must always be processed before any sales transactions are accepted. In this case, the store open transaction might switch the “transaction mode” from a value of “A” to a value of “B”. The sales transaction would then be configured as a mode “B” transaction. The “transaction mode” would be transitioned to level “B” for the given chronological/physical location combination by the *Data Integrity Management* component when the store open transaction is received.

In general, the “transaction mode” defaults to “A” for all newly identified chronological/physical location combinations. This level can then be adjusted to any valid upper case alpha character. Any new level setting includes not only the specified setting but also all values below the specified setting. For example, a setting of level “B” will process all transactions identified as valid for either level “B” or “A”. In this example, however, transactions identified as “C” mode transactions will be rejected and stored within the *Exception Management* database.

#### Business Management Information

The *Business Management Information* is used by RTP for two distinct purposes. The first purpose is to maintain information that clearly defines the current “transaction mode” and what transaction is expected next from each individual physical device. Figure 6 shows the overall structure for the multiply linked list that is used to maintain control information. The left most linked list, in cyan, is used to keep chronological information. There is no immediate control information contained within this list. The next linked list is extended off each chronological group, in pink, and contains an entry for each physical location along with the associated “transaction mode”. The final linked list, in gray, is extended off each of the physical locations to identify each physical device along with the next expected transaction sequence number. This complete information structure addresses the first purpose of the *Business Management Information,* which is to control the order in which transactions are processed.



Figure 6. Business Management Information Model

The second purpose of *Business Management Information* is to ensure that *Exception Management* resubmissions are processed once and only once on a given posting date. In order to provide this service, the system will maintain a list and/or flag for each transaction within the exception system that has been submitted on a given posting date. Any resubmission will be rejected as a duplicate if it is contained within this verification list.

### Audit

The *Audit* component is similar to the previously described *Edits* in that it verifies information within a transaction prior to its submission to the host systems. Unlike edits, an audit can include complex comparisons between fields throughout a transaction. Comparison elements can include data such as fields within separate records, summation of fields within a group of records (e.g., all item extended cost must equal total of tender records), or calculations using system data such as posting date minus two days. Audits are defined within the configuration database as described in section **Error! Reference source not found.**.

All audits are enforced within an individual transaction without reference to any other transactions and/or system summary data. Audit failures prevent the field **tag/data** information from being passed on to subsequent components. Instead, transaction information is stored in the *Exception Management* system and one or more **control** records are sent to the *Data Integrity Management* component to prevent multiple processing of a single faulty transaction.

### Data Integrity Management

The *Data Integrity Management* component handles the final processing of data within the *Transaction* stage. Typically, this component is responsible for re-verification of the transaction check point, storing the verified transaction in non-volatile storage[[4]](#footnote-4), and simultaneously updating the business management information[[5]](#footnote-5) assuring future transactions are properly handled by the *Check Point Management* component. Re-verification is introduced to prevent a second instance of the transaction processor from attempting to update a second copy of the transaction while the first instance is processing audit information. Although highly unlikely by design, an uncaught event of this type could result in unacceptable duplication of data.

In contrast, *Data Integrity Management* may receive a control record without matching **tag/data** for edit or audit failures from the *Check Point Management* or *Audit* components respectively. Receipt of an independent control record updates business management information only. This update allows the system to accept subsequent transactions from the selling device and prevents processing faulty transactions multiple times.

In either case, this component has the final responsibility of notifying the prior stage or submitting system the disposition of the processing of this transaction. Anticipated dispositions would generate responses such as processed successfully, duplicate transaction, transaction out of sequence, or unknown transaction.

### Transaction Repository

The *Transaction Repository* is non-volatile storage for transaction information that has been completely verified and is awaiting delivery to the backend host systems. The repository storage is broken into three distinctive areas:

* **Control and Recovery Information** – This is a series of data points used to verify the integrity of the business management information and improve the reconstruction of internal control information during a system restart and/or recovery.
* **Transactions** – Transactions are stored within this area in the order in which they were processed. Information is stored in the internal linked list based **tag/data** format. This storage structure allows the translators a method to quickly load the data into the same class used by the *Transaction* stage improving code reuse and overall system performance.
* **Operational Statistics** – The operational statistics information is used for system monitoring and trend analysis. Information contained within this block includes data such as average transaction processing time, number of transactions processed, number of rejected transactions, etc… This information is specific to the processing day it represents and can be accessed through the debugging tool.

## Translation

The *Translation* stage of the conceptual system data flow is responsible for creating the appropriate records for each of the configured host systems based on the validated transactions and the configuration[[6]](#footnote-6) database, thereby, completing the RTP process. During system initialization, one instance of the translator is created for each receiving host system as shown in Figure 7. Each instance is functionally identical deriving the appropriate output requirements from the configuration database using the supplied system code (e.g., RS, CS, MI, …) as a key. The output requirements driven by the configuration database include the output record sequence, output record format, frequency of data delivery, and method of data delivery (e.g., middleware versus ftp).



Figure 7. Translator Multiple Instance Structure

Once configured, each translator independently reads sequentially through the master transaction file either creating the appropriate host system specific output records or updating control information such as record counts that will eventually be used in header or trailer records. Control and recovery information is updated within a system specific status area to prevent the reprocessing of transactions. The basic components involved in this process are outlined in Figure 8.



Figure 8. Basic Translator Conceptual Data Flow

### Transaction Repository

Initially introduced in section 2.2.5, the non-volatile storage based *Transaction Repository* is the central point within the system in which data flow is transition from being driven by a physical event at a selling device to being processed as required by the host system (e.g., once daily, at store close, every hour, etc…). The two key elements within the repository fundamental to this process are as follows:

* **Transactions** – A complete set of transaction data is stored within the single repository in the order in which they were verified.
* **Control and Recovery Information** – Within this area, a set of data points are stored for each host system instance. These data points clearly define which transactions have been transmitted to the corresponding host system. Each translator reads and updates the appropriate set of data points based on the assigned system code key. Use of this information allows the translator to process and submit each transaction to the host system once and only once. All control information is validated during restart of the individual instances of the translators against redundant control information.

### Transaction Loader

The *Transaction Loader* component handles locating and loading the next transaction from the *Transaction Repository* for the host system associated with this instance of the translator. This component is designed to assure that all appropriate transactions are passed to the host once and only once. Transactions are evaluated for appropriateness based on their originating location type (e.g., merchandise WDW, merchandise DCL, food & beverage, DVC, etc…) and the target host. For example, since DCL settles credit cards through their own process the credit settlement instance of the translator will ignore any transactions from DCL locations. After a transaction is deemed inappropriate or delivered successfully, this component updates information within the **Control and Recovery Information** structure to prevent duplicate processing/delivery of transactions.

### Transaction Analyzer

The *Transaction Analyzer* component handles applying the necessary business rules against each individual transaction. The Configuration Database defines one or more of the following events as the appropriate processing for an individual transaction[[7]](#footnote-7) given the identified destination host system:

* **Summary Information Update** – results in updating information collected for a host system that requires a special transaction such as a header or trailer. Transactions that impact summary information may apply basic math functions such as increment count, decrement count, add a value into a total, save the maximum value, etc… Multiple values may be maintained for each individual host system.
* **Output Record Creation** – results in the immediate creation of one or more output records based on information contained within the loaded transaction. This event may simply be the transmission of a single copy of all transaction data or delivery of one inventory adjustment record for each item sold.
* **Event Signal Creation** – results in an event being triggered for a host that receives information in batch as defined in section 2.3.5. For example, a host system may only deliver information when a “store close” transaction is received. This event is the trigger for that transmission process.

For example, a sales transaction going to MERLIN may result in **output record creation** generating one inventory sales adjustment for each line item within the transaction. In contrast, a store close transaction going to the data warehouse may result in both an **output record creation** containing all the transaction information and an **event signal creation** which would cause the output file for the associated location to be sent to the data warehouse for immediate processing. Specific business rules definition capabilities are defined within the Configuration Database as detailed in section 0 of this document.

### Real-time Host Interface

The *Real-time Host* version of the translator is designed to feed output records to systems that accept information updates in real-time. Initially, this is limited to EDS for deposits although it is expected that it will be extended to include the financial and MERLIN systems in the near future. The final step in this interface will feed the appropriate transactions to designated middleware for submission to the host system. Middleware support will be developed based on WDW architecture standards. Successful delivery of a transaction to the middleware will result in the appropriate updates to the **Control and Recovery Information** through the *Transaction Loader* component. In addition, operational information such as number of transactions processed will be updated in the operational log for system monitoring.

### Batch Host Interface

The *Batch Host* version of the translator is designed to collect and subsequently feed a set of output records to a host system that only accepts data in batch. The data flow of this type of interface is identical to a real-time interface with the addition of an *Outbound Interface* component as shown in Figure 9.



Figure 9. Batch Host Translator Conceptual Data Flow

If configured as a batch host, the translator instance will temporarily store output records in non-volatile storage until the appropriate **event signal** is received. In the initial release, the *Outbound Interface* will transmit data based on the following **event signals**:

* Repeating Time Interval (e.g., every 15 minutes)
* Specific Time of Day (e.g., 3:30 am)
* Transaction Receipt (e.g., receive Store Close)

The content of the transmitted data will either include all output records since the last transmission or output records since the last transmission for a given physical location. Each transmission will be recorded within the operational log for support and system monitoring.

## Batch versus Real-time Implementation

During initial batch only implementation, it is anticipated that the *Batch* and *Transaction* stages be implemented as a single program and/or process. Clear lines of division, however, should be maintained throughout the code in order to easily facilitate the separation of these two components into independent modules. In the real-time environment, *Batch* will continue as a standalone program while the *Transaction* functionality will be moved into the middleware software.

In contrast, the *Translation* stage along with its inbound interfaces will not change during the transition to real-time operation. The outbound interfaces, however, will be integrated into existing middleware strategies for directly feeding host systems that can accept data in real-time. Batch feed systems will continue with their batch-ftp oriented delivery.

# Application Process Model

This chapter is designed to provide a structural overview of the processes and their related inter-process communication.

Figure 10. Process Model Overview – Business

## Daemon

### Initialization

### Business Management

### Input Configuration Management

### Process Management

### Support Infrastructure

## Batch Process

### File Processing

### Transaction Processing

## Exception Management Unload

## Support Tool

# Application Software Structure

The following chapter defines the internal structure of each of the executable programs that make up the RTP System. Documentation includes a general description, command line parameters, basic application flow, and class definitions.

## RTPINIT

The *Daemon Initialization (RTPINIT)* program should be the first program executed as part of the RTP daemon. This program by design has two modes of operation. The first mode is based on its execution from the command line either by a user or through Maestro. In this mode, the program executes commands such as start the daemon, terminate the daemon, recycle the daemon, or show status.

If the “start the daemon” command is executed the program forks a second process which becomes the daemon. The original process verifies a successful start and then returns control to either the user or the Maestro script. The following basic functionality is provided by the daemon component of this executable:

* Initialization of the shared memory control block used by all applications to monitor the status of the system, a particular translator, a physical location, etc…
* Recovery of control information such as last transaction processed after an abnormal termination
* Preload of input file configuration information into a shared memory block
* Load on demand additional input file configuration information as requested
* Storage and maintenance of STDOUT and log information from all other programs including any required interface to service center or SNMP for notifications
* Initialization of the appropriate instances of the Data Driven Translators for the host systems
* Process management through the failure of a Data Driven Translator or the detection of any data integrity issues due to batch program failures

### Command Line Parameters

The *RTPINIT* program is executed from the command line by a user or through the Maestro scheduling system. In either case, the following syntax should be followed:

RTPINIT {options}

where {options} include one or more of the following commands:

| **Switch** | **Description** | **Action If Missing** |
| --- | --- | --- |
| /S | Start the daemon process; verify it started successfully; and report the results to the caller. The exit code returns EAGAIN if in initialization, EBUSY if in shutdown, and ZERO if the daemon is operational. Any other return code indicates an error during processing. | No Action |
| /P | Starts the daemon as the local process. This setting should only be used during debugging to analyze the operation of the program. | No Action |
| /T | Terminate the currently running daemon process. The program returns successfully if the signal is received to terminate which is not necessarily after the application has completely shutdown. | No Action |
| /R | Executes the terminate command immediately followed by a start with instructions to wait until the termination is complete prior to starting the new instance. | No Action |
| /V? | Sets the verbose level that controls how much information is written to the screen file during normal operation. “?” is a value from 0 to 5 that controls the amount of information: 0=None; 1=Normal Major Events; … ; 5=All Output Statements | Defaults to value in RTPINIT.ini. |
| /O=? | Establishes wait time in seconds for a response from the command processor on any commands submitted for processing. “?” represents a number of seconds. | Default if not specified is 30 seconds. |
| /W=? | Establishes wait time in seconds for shutdown to complete before attempting to start the daemon. “?” represents a number of seconds. This option is only valid in combination with options S, R, and P. | Default if not specified is 30 seconds. |
| /I=? | Establishes wait time in seconds for the program to wait for the newly started daemon to achieve a status of “Normal Operation”. “?” represents the number of seconds. This option is only valid in combination with options S, R, and P. A value of ZERO causes the program to return without checking the status of the daemon. | Default if not specified is 60 seconds. |
| /N | If included this switch signals the program that no information should be written to the local output device. | Not Set |

If no parameters are provided the program returns the status of the executing daemon. If there is not an instance of the daemon available, the program returns a NOT FOUND error code.

### Basic Application Flow

### Class Definitions

The following classes are used within the *Daemon Initialization* program. Details on calling and return parameters are contained within comments in the class source code.

#### CCmdLine Class

The *CCmdLine* class is used to parse and handle error reporting for the command line arguments of all RTP executables. This class should be initialized at the start of “main” where it is provided with the *argc* and *argv* variables. The next step in the main application should be to call methods such as *Define* to establish the rules by which the parameters should be evaluated. RTPCMDLINE.h and RTPCMDLINE.cpp contain the class definition and methods source code accordingly. The following methods are supported by this class:

**General Methods**

**CCmdLine** - The constructor for this class initializes all class variables and moves cleaned up copies of the command line arguments into local storage.

**~CCmdLine** – The destructor releases all allocated system resources.

**Validation Control and Inquiry Methods**

**DefineParameter** – Establishes a definition for a command line parameter and causes the class to evaluate the availability of the required information. Arguments are evaluated against the definition. Supported parameter types include a switch, a numeric value, a string value, or a general parameter such as a filename.

**ParameterCheck** – Checks to see if a command line switch or parameter exists.

**GetValue** – Retrieves the values associated with a given parameter. This includes either numeric or string information.

**SetError** – Store error string that should be reported to the caller based on a parameter or major application failure. Storage of a string will cause the system to prevent execution by signaling the application that an error was detected.

**Operational Information Retrieval Methods**

**UserHelpRequest** – Returns a flag that indicates if the user requested help on the command line. This method is used by external code that provides error information such as missing parameters to make sure that the parameter is not missing because the user simply wanted help information.

**MultipleNonOptionCount** – Retrieves a count of the number of arguments passed on the command line that are not options. This will retrieve things like filenames if appropriately defined for this command line class.

**ShowMessageText** – Causes the class to display the appropriate help and/or error information to the end user.

**OkToProceed** – Indicates whether or not an error was detected in the parameter list, reported through the *SetError* method, or if the user requested help. In addition, it provides a facility to trap any undefined parameters. If appropriate, this method can also display the error and help information to the user.

**GetLastError** – Retrieves the information about the last error that occurred in a class method. There are two forms of this function. The first returns both a code and text string while the second returns only the error codes.

#### CDirMgmt Class

The *CDirMgmt* class is used to provide access to directory information based on a path and a pattern match. This class retrieves directory information, make sure it matches the given pattern, provides various sorts, and allows for updates to working information such as file status. RTPDIRMGMT.h and RTPDIRMGMT.cpp contain the class definition and methods source code accordingly. The following methods are supported by this class:

**General Methods**

**CDirMgmt** - The constructor for this class initializes all class variables and loads directory information into memory.

**~CDirMgmt** – The destructor releases all allocated system resources.

**OpenOK** – Returns true if the directory load was successful. This does not necessarily mean that there were matching entries it simply indicates that no errors were detected.

**Data Update Methods**

**Exclude** – Method causes the class to remove from the current directory listing any files that match the provided pattern. This allows applications such as *RTPBATCH* the ability to exclude any file with the “.processed” extension.

**SortByTimeName** – Method resorts the directory list first by update time and then by name.

**Data Retrieval Methods**

**ResetGet** – Resets the control information within the class to the start of the directory list currently loaded. The method DOES NOT reload the directory, change the sort order, or restore any excluded files. The *Refresh* method should be called to restore the list to its original form. This method allows the caller to retrieve a series of parameters under the same name a second time without running out the list.

**Get** – Retrieves the next filename from the directory based on the current sort order and exclusion list.

**Refresh** – Reloads the directory information from the operating system. This call restores the file order to that provided by the operating system and clears any exclusions.

**Operational Information Retrieval Methods**

**GetLastError** – Retrieves the information about the last error that occurred in a class method. There are two forms of this function. The first returns both a code and text string while the second returns only the error codes.

#### CSupport Class

The *CSupport* class is used to provide individual processes within the RTP system access to standard support functionality including functions such as logging, process control, exception management, operational logs, etc… The basic class definition is contained in RTPSUPPORT.h with function source contained within RTPSUPPORT.cpp. The following methods are supported by this class:

**General Methods**

**CSupport** - The constructor for this class initializes the appropriate connections and control structures to provide the appropriate support services.

**~CSupport** – The destructor closes disconnects from any support processes (e.g., logging) and releases all allocated system resources.

**ForceRedirect** – This module is typically called from within the class constructor to redirect standard output (STDOUT) and standard error (STDERR) to the screen named pipe. This function is provided as a public method for use in the daemon initialization program.

**Show** – This module displays an output line including the appropriate prefix of time, module, and process ID. The function handles the appropriate line wrap for a clean display. Length of lines is controlled by SZ\_SHOWLIMIT defined in the *RTPGENERIC.h* source file.

**Log** – This module writes an error or log message to both the Logging and Screen processes.

**Process Management Methods**

**AddChildProcess** – This method adds an entry to the classes internal linked list that is maintained for reference to active children processes. In the case of RTPINIT, this capability is used to monitor and, if necessary, terminate special processes such as the Screen File Clean Up or Input Format Load processes.

**SignalChildrenDown** – This method sends a given signal to all the recorded child processes.

**RemoveChildProcess** – This method removes a child’s entry from the class storage.

**Data Retrieval Methods**

**Status** – Returns the status of the daemon. Definitions are provided in the RTPGENERIC.h file and all begin with “SCD\_”. Access to this field since it is a single byte does not lock the memory region. This approach improves performance since this is a commonly access field. This is also defined as an inline function for performance.

**Verbose** – Returns the verbosity level of the daemon. Definitions are provided in the RTPGENERIC.h file and all begin with “VBL\_”. Access to this field since it is a single byte does not lock the memory region. This approach improves performance since this is a commonly access field. This is also defined as an inline function for performance.

**VerboseCheck** – Returns a flag indicating if the verbosity level is at or below the level identified. Access to this field since it is a single byte does not lock the memory region. This approach improves performance since this is a commonly access field. This is also defined as an inline function for performance.

**GetHostStatus** – Returns the status of a selected host translator program. Definitions are provided in RTPGENERIC.h and all begin with “HSC\_”.

**GetLog** – Retrieves a log packet along with its associated data. This method should only be used in the Logging Process.

**GetCmd** – Retrieves a command packet along with its associated data. This method should only be used in the command processor process within the Daemon.

**GetResponse** – Retrieves a response packet from the command processor after its completion of a command request. It is anticipated that this will primarily be used within client processes.

**TransactCmd** – Sends a command packet and then waits for the response returning the result of the action to the caller.

**Data Update Methods**

**SetStatus** – Allow the *RTPINIT* program with the ability to update the status information within the Master Shared Memory segment.

**SetShowName** – Resets the process ID used with all output statements generated by the support infrastructure. This method should be called after any fork occurs.

**SetNotScreenOwner –** This disables the ownership properties for the current instance of the screen named pipe class, *CNmPipe*. This method is used after a fork in the process that originally created the pipe.

**SetNotMasterOwner –** This disables the ownership properties for the current instance of the master memory segment class, *CMaster*. This method is used after a fork in the process that originally created the master shared memory segment.

**SendCmdPacket –** Sends a command packet to the command processor within the Daemon. Command functions are defined in section 5.2.4.

**QueueCmdPacket –** Writes a command packet to the command processor without using any of the internal class structures. This is primarily for use in SIGNAL handlers such as SIGCHLD.

**SendLogPacket –** Sends a log packet to the logging process within the Daemon. This process is used to feed support information on for analysis and processing. Supported message types are defined in section 5.2.3.

**Operational Information Retrieval Methods**

**GetLastError** – Retrieves the information about the last error that occurred in a class method. There are two forms of this function. The first returns both a code and text string while the second returns only the error codes.

#### CNmPipe Class

The *CNmPipe* class is used to provide a standard implementation of FIFOs used to provide inter-process communication within the RTP system. The class includes support for client, server, and bi-directional modes. The basic class definition is contained in RTPNMPIPE.h with function source contained within RTPNMPIPE.cpp. The following methods are supported by this class:

**General Methods**

**CNmPipe** - The constructor for this class initializes the appropriate connections and control structures to provide the appropriate support services.

**~CNmPipe** – The destructor closes disconnects from any support processes (e.g., logging) and releases all allocated system resources.

**OpenOK** – Verifies that the named pipe(s) opened successfully and are available for byte level operaitons.

**OpenStream** – Verifies that the named pipe has been opened for stream transactions that are initiated through the *GetLine* and *PutLine* methods.

**Response** – Retrieves whether or not a response pipe was established during the creation of this class.

**Control Methods**

**EnableStream** – This method enables stream operations against the pipe.

**GetLine** – Retrieves a single line from the stream. This method requires a prior call to *EnableStream* in order to operate properly. This is primarily used in the Screen Process.

**PutLine** – Puts a single line into a stream. This method is specifically designed for use in the Screen Process. It should always be proceeded by a call to the *EnableStream* method.

**GetBuffer** – Retrieves a header and data block from the pipe as appropriate.

**PutBuffer** – Puts either a byte buffer or a header-data block combination into the pipe.

**GetResponse** – Retrieves a buffer of information from the response FIFO if configured.

**Operational Information Retrieval Methods**

**GetLastError** – Retrieves the information about the last error that occurred in a class method. There are two forms of this function. The first returns both a code and text string while the second returns only the error codes.

#### CMaster Class

The *CMaster* class is used to create and maintain the master shared memory segment used to share control information amongst the primary processes of the RTP system. This memory segment is created with a key based on information contained within the RTPSUPPORT.INI file. The format of the data within the segment is defined in section 5.2.1 of this document. The basic class definition is contained in RTPMASTER.h with function source contained within RTPMASTER.cpp. The following methods are supported by this class:

**General Methods**

**CMaster** - The constructor for this class either access or create the shared memory block used to share statistical or control information amongst the processes within the RTP system. This process also opens or access the semaphore used to control update access to this memory block. If this is a client open, the class also verifies the availability of the screen FIFO – this precaution is taken to assure that the daemon is fully operational and that we are not reviewing data in the shared segment between the time it is created and first filled with data.

**~CMaster** – The destructor closes disconnects from the segment and releases all allocated system resources.

**OpenOK** – Verifies that the memory segment was properly access or created.

**ReleaseOwnership** – This method turns off the ownership flag within the class that causes the system to delete the shared memory segment once the class is removed. This method should only be used in children processes as the daemon starts to prevent the child’s termination from releasing the segment too early during shutdown.

**Data Retrieval Methods**

**Status** – Returns the status of the daemon. Definitions are provided in the RTPGENERIC.h file and all begin with “SCD\_”. Access to this field since it is a single byte does not lock the memory region. This approach improves performance since this is a commonly access field. This is also defined as an inline function for performance.

**Verbose** – Returns the verbosity level of the daemon. Definitions are provided in the RTPGENERIC.h file and all begin with “VBL\_”. Access to this field since it is a single byte does not lock the memory region. This approach improves performance since this is a commonly access field. This is also defined as an inline function for performance.

**VerboseCheck** – Returns a flag indicating if the verbosity level is at or below the level identified. Access to this field since it is a single byte does not lock the memory region. This approach improves performance since this is a commonly access field. This is also defined as an inline function for performance.

**GetHostStatus** – Returns the status of a selected host translator program. Definitions are provided in RTPGENERIC.h and all begin with “HSC\_”.

**Data Update Methods**

**SetStatus** – Allow the *RTPINIT* program with the ability to update the status information within the Master Shared Memory segment.

**AddHost** – Method adds a new host configuration to the shared memory segment. It handles data validation including verification that all codes are unique.

**Operational Information Retrieval Methods**

**GetLastError** – Retrieves the information about the last error that occurred in a class method. There are two forms of this function. The first returns both a code and text string while the second returns only the error codes.

#### CINIDecode Class

The *CINIDecode* class is used to access and retrieve parameter information from a standard INI file based on the format defined in section 5.4.3 of this document. The basic class definition is contained in RTPINIDECODE.h with function source contained within RTPINIDECODE.cpp. The following methods are supported by this class:

**General Methods**

**CINIDecode** - The constructor for this class opens the file and extracts all the variables contained within the INI file. Verification of successful load requires a call to the *OpenOK* method within the class. On failure, information is stored in the last error parameters that can be retrieved through the *GetLastError* methods.

**~CINIDecode** – The destructor closes the input file and releases all allocated system resources.

**GetFilename** – Retrieves the qualified name of the file opened by the class.

**OpenOK** – Verifies that the file has been opened and parsed successfully.

**Data Retrieval Methods**

**ResetGet** – Resets the control information within the class to the start of the INI variable list. This method allows the caller to retrieve a series of parameters under the same name a second time without running out the list.

**Get** – Retrieves a variable from the INI file. If multiples exist repeated calls to this function will return each entry in sequence until none remain or the *ResetGet* method is called.

**GetField** – Retrieves individual comma delimited data elements from a single line in the INI file. A ZERO based index is used to move through the values on the line.

**Operational Information Retrieval Methods**

**FileLineCount** – Retrieves the number of lines contained within the INI file.

**VariableCount** – Retrieves the number of variables read from the INI file during validation.

**GetLastError** – Retrieves the information about the last error that occurred in a class method. There are two forms of this function. The first returns both a code and text string while the second returns only the error codes.

second returns only the error codes.

#### CDBLOps Class

The *CDBLOps* class provides a standard interface to the system’s operational log. Using embedded SQL against Oracle, the class validates last status information and provides all update capabilities. RTPDBLOPS.h and RTPDBLOPS.cpp contain the class definition and methods source code accordingly. The following methods are supported by this class:

**General Methods**

**CDbLOps** - The constructor for this class initializes all class variables, accesses the database, and updates the *OPL\_FILE\_RUN* table as defined in section 5.5.1.

**~CDbLOps** – The destructor releases all allocated system resources.

**OpenOK** – Method is used to validate that the database was accessed successfully and that the *OPL\_FILE\_RUN* record was inserted successfully.

**Data Update Methods**

**AdjustCount** – Adjusts the internal class count to indicate the number of transactions processed. This is an increment or decrement based on the value provided.

**AdjustOrphanCount** – Adjusts the internal class orphan count to indicate the number of orphans detected within the file. This is an increment or decrement based on the value provided.

**ProcessComplete** – This method updates the *OPL\_FILE\_RUN* table to show the final status of a file processed. This method should be called once the processing of the file is either complete or failed to finalize this processing.

**LogicalUnitStart** – This method marks the start of a logical batch by updating the *OPL\_LOGICAL* and *OPL\_LOGICAL\_RUN* as defined in section 5.5.1. This call also initializes class control variables to calculate information such as average transaction time.

**LogicalUnitComplete** – This method marks the end of processing a logical batch. It updates the *OPL\_LOGICAL* and *OPL\_LOGICAL\_RUN* tables. This call should be called once and only once after a LogicalUnitStart method call.

**Operational Information Retrieval Methods**

**GetLastError** – Retrieves the information about the last error that occurred in a class method. There are two forms of this function. The first returns both a code and text string while the second returns only the error codes.

#### CCfgHost Class

The *CCfgHost* class is used to retrieve destination host configuration information from the Oracle database using embedded SQL. In addition to *RTPINIT*, this class is used by *RTPDTRAN*. RTPCFGHOST.h and RTPCFGHOST.cpp contain the class definition and methods source code accordingly. The following methods are supported by this class:

**General Methods**

**CCfgHost** - The constructor for this class initializes all class variables. A class can be configured for a specific host, all hosts, today, or a different date based on the constructor parameters.

**~CCfgHost** – The destructor releases all allocated system resources.

**Data Retrieval Methods**

**ResetGet** – Resets the control information to the first row that matches the criteria specified when the class was created. The next call to *GetConfig* will retrieve the first row.

**GetConfig** – Retrieves configuration information pulled from the host database. Various versions of this method retrieve different amounts of information depending on the callers need.

**Operational Information Retrieval Methods**

**HostCount** – Returns the number of host records available through this class. If this is a host specific instance then this function will always return 1.

**GetLastError** – Retrieves the information about the last error that occurred in a class method. There are two forms of this function. The first returns both a code and text string while the second returns only the error codes.

#### CCfgLocation Class

The *CCfgLocation* class is used to retrieve location configuration information used for transaction formatting and routing from the Oracle database using embedded SQL. In addition to *RTPINIT*, this class is used by the transaction stage of *RTPBATCH*. RTPCFGLOCATION.h and RTPCFGLOCATION.cpp contain the class definition and methods source code accordingly. The following methods are supported by this class:

**General Methods**

**CCfgLocation** - The constructor for this class initializes all class variables. A class can be configured for either today or a different date based on the constructor parameters.

**~CCfgLocation** – The destructor releases all allocated system resources.

**Operational Information Retrieval Methods**

**LocationCount** – Returns the number of location records available through this class for the specified date.

**DeviceCount** – Returns a number of devices that are associated with all the locations that are actively configured. This is an estimate based on data within the database.

**GetLastError** – Retrieves the information about the last error that occurred in a class method. There are two forms of this function. The first returns both a code and text string while the second returns only the error codes.

## RTPBATCH – Batch Stage

### Command Line Parameters

The *RTPBATCH* program will normally be executed through either Mastreo based on a schedule or as the completion of a file transfer. The command structure allows the caller to either receive output through the local device or have it appended to the daemon’s standard out file. In either case, the following syntax should be followed:

RTPBATCH {options}

where {options} include one or more of the following commands:

| **Switch** | **Description** | **Action If Missing** |
| --- | --- | --- |
| /I=$..$ | Provides an input file pattern used to locate the batch input files. This can be either a full path or a filename of the current working directory. “?” within the name is considered an unknown character and “\*” is considered an zero or more unknown characters. This parameter must be included with an execution of this program. | Program Fails |
| /N | If included this switch signals the program that no information should be written to the local output device. | Not Set |
| /L | If included all output is redirect to the local console rather than being sent to the standard output screen file maintained by the daemon. | Output is sent to the daemon’s standard output file. |
| /RC=# | Switch establishes the number of times (“#”) that the system should review the directory beyond the initial pass for possible input files. This function can be used to grab late files as well as reprocess a file that may have been received out of sequence. By default, this feature is turned off. | RTPBATCH.INI value is used if available otherwise defaults to “0”. |
| /RW=# | Switch establishes the number of seconds to wait between directory passes. This value is not used unless the “RC” value is set to something other than zero which is its default value. By default, the system waits 1 second between passes. | RTPBATCH.INI value is used if available otherwise defaults to “1”. |
| /TC=# | Switch establishes the number of times the system should resubmit a transaction that is rejected because it was received before a lower sequence number is received. This feature is designed to handle if transactions are received out of sequence due to in-bound threading. The default is to attempt twice before failing the transaction. | RTPBATCH.INI value is used if available otherwise defaults to “2”. |
| /TW=# | Switch establishes the number of seconds to wait between resubmissions of a transaction that is reject because it is received before an earlier transaction was fully processed. The default is to wait 15 seconds. | RTPBATCH.INI value is used if available otherwise defaults to “15”. |
| /V? | Sets the verbose level that controls how much information is written to the screen file during normal operation. “?” is a value from 0 to 5 that controls the amount of information: 0=None; 1=Normal Major Events; … ; 5=All Output Statements | Defaults to the current setting for the daemon. |

### Basic Application Flow

### Class Definitions

The following classes are used within the *Batch Transaction Processing* program. Details on calling and return parameters are contained within comments in the class source code.

#### CLogicalBatch Class

The *CLogicalBatch* class is used to validate and pull transactions information based on the logical batch concept from a standard input file. This class is built on the *CTransFile* class for core functionality such as file validation and data retrieval. Configuration information is pulled internally through the *CINIDecode* class from the RTPLOGIC.INI file. Retrieved information is organized by logical batch for recording and transmission to the *Transaction Stage* of the data flow. RTPLOGICALBATCH.h and RTPLOGICALBATCH.cpp contain the class definition and methods source code accordingly. The methods supported by this class are listed in categories below due to the number of methods:

**General Methods**

**CLogicalBatch** - The constructor for this class initializes all class variables and opens the specified input file. Input files are always opened in an exclusive mode.

**~CLogicalBatch** – The destructor releases all allocated system resources and closes the file.

**OpenOK** – Returns true if the file has been opened and if all contained within the file are valid.

**ValidateFileStructure** – This method validates the overall file structure. It must always be called after class construction but before any other method calls except the *OpenOK* method. A return value of true indicates that the file is available for use.

**Data Retrieval Methods**

**ResetUnit** – Method positions cursor within the input file either at the start of the file or at the beginning of the identified logical unit. A *GetNextUnit* call must always occur immediately following this method but prior to retrieval of any transaction information.

**GetNextUnit** – Retrieves the logical Unit ID associated with the next logical unit within the input file.

**GetNextTransaction** – Retrieves the next transaction contained within the current logical unit. An end of file is signaled once the last transaction is read from the current logical batch.

**Release** – Releases transaction buffer returned through the *GetNextTransaction* method. This method should always be called for each returned buffer.

**Operational Information Retrieval Methods**

**FileOffset** – Retrieves the numeric offset within the input file at which the last retrieved transaction is stored. This value should be used for debugging purposes only.

**GetLastError** – Retrieves the information about the last error that occurred in a class method. There are two forms of this function. The first returns both a code and text string while the second returns only the error codes.

#### CTransFile Class

The *CTransFile* class is used to pull transactions information from a standard input file for processing. This class provides overall file integrity as well as logical batch structure verification prior to allow the retrieval of individual transactions. This approach is designed to prevent the application from submitting transactions that may have been received through a generally corrupt file. RTPTRANSFILE.h and RTPTRANSFILE.cpp contain the class definition and methods source code accordingly. The methods supported by this class are listed in categories below due to the number of methods:

**General Methods**

**CTransFile** - The constructor for this class initializes all class variables and opens the specified input file. Input files are always opened in an exclusive mode.

**~CTransFile** – The destructor releases all allocated system resources and closes the file.

**OpenOK** – Returns true if the file has been opened and if all transactions consist of a valid base sequence.

**Validation Control and Inquiry Methods**

**DefineUnitStartEnd** – Method allows the caller to establish a start and end transaction code for the logical unit of work within the file. This information is used to count the number of units and verify the integrity of the unit using the *DefineIntegrityCheck* and *Validate* methods. This method must be called prior to the *Validate* module for proper operation.

**DefineUnitCheck** – Method allows the caller to establish a validation count processed within a logical unit of work within the input file during validation. This information is used to further verify that all information was successfully received prior to processing any data. The *DefineUnitStartEnd* and *Validate* methods are critical to making this definition effective. This method must be called prior to the *Validate* module for proper operation.

**SetFailOnCountMismatch** – Switch to true if you want the entire file to fail if a single logical batch has a count mismatch in the trailer.

**Validate** – Validates the integrity and base sequence format of the opened input file. This call must be made prior to retrieving record counts and/or any transaction information.

**GetNextMismatch** – Retrieves the information about the next mismatch until there is no more mismatch information available.

**ResetNextMismatch** – Resets the pointer to the start of the mismatch list.

**ValidateElapsedTime** – Retrieves the number of milliseconds elapsed during the validation of the input file. This value can be divided by the value returned by the *Count* method to determine an average transaction validation time.

**ValidateElapsedAverage** – Retrieves the average number of milliseconds elapsed per transaction processed during the validation of the input file.

**Transaction Retrieval Methods**

**GetNext** – Returns the next transaction from the input file including a buffer version that can be used to submit it to the *Transaction Stage*.

**LastPosition** – Returns the byte offset into the file where the last read transaction begins.

**SetPosition** – Sets the file pointer for the transaction file to the given transaction including verifying that it is a transaction boundary.

**Release** – Releases storage allocated by the *GetNext* method to return the transaction buffer.

**GetUnit** – Retrieve the information for a specific logical unit within the current input file.

**GetNextUnit** – Retrieves the information about the next logical unit recognized by the *Validate* method until there is no more unit information available.

**ResetNextUnit** – Resets the pointer to the start of the logical unit list.

**Operational Information Retrieval Methods**

**Count** – Returns a count of the number of transactions contained within the input file. This method only works if the file contains all valid base sequences. This call must be made after a call to the *Validate* method.

**OrphanCount** – Returns a count of the number of transactions detected outside of the identified logical unit.

**UnitCount** – Returns a count of the number of logical units detected.

**UnitMismatch** – Returns true if a count contained within the logical unit does not match the record count calculated by the validation of the transaction file.

**GetLastError** – Retrieves the information about the last error that occurred in a class method. There are two forms of this function. The first returns both a code and text string while the second returns only the error codes.

#### CDbLOps Class

The *CDbLOps* class is used to update and query the *Operational Log* tables within the RTP database. This table contains summary information (e.g., when processed, number of transactions, etc…) and status information about batch files received and subsequently distributed by the RTP system. RTPDBLOPS.h and RTPDBLOPS.cpp contain the class definition and methods source code accordingly. The methods supported by this class are listed in categories below due to the number of methods:

**General Methods**

**CDbLOps** - The constructor for this class initializes all class variables, opens a connection to the database, and creates an *OPL\_FILE\_RUN* record for the batch file.

**~CDbLOps** – The destructor releases all allocated system resources.

**OpenOK** – Returns true if the class constructed, connected to the database, and made the appropriate insert into the *OPL\_FILE\_RUN* table for this run of the batch process.

**Operational Information Retrieval Methods**

**GetLastError** – Retrieves the information about the last error that occurred in a class method. There are two forms of this function. The first returns both a code and text string while the second returns only the error codes.

## RTPBATCH – Transaction Stage

### Class Definitions

The following classes are used in the *Transaction Stage* of the *Batch Transaction Processing* program. Details on calling and return parameters are contained within comments in the class source code.

#### CTransact Class

The *CTransact* class is used to validate and decode transaction information contained within the input buffer.

RTPTRANSACT.h and RTPTRANSACT.cpp contain the class definition and methods source code accordingly. The methods supported by this class are listed in categories below due to the number of methods:

**General Methods**

**CTransact** - The constructor for this class initializes all class variables and decodes the provided transaction loading it into a format that can be used by the class to serve up field level information.

**~CTransact** – The destructor releases all allocated system resources.

**Operational Information Retrieval Methods**

**GetLastError** – Retrieves the information about the last error that occurred in a class method. There are two forms of this function. The first returns both a code and text string while the second returns only the error codes.

## RTPDTRAN

## RTPEMU

## RTPDEBUG

The *Debugging Tool (RTPDEBUG)* program is designed to provide support personnel with access to control information within the running daemon. In addition, this tool provides an access point to update control information such as active Input Version formats and the Verbosity level. Use of the update functions within this tool can have a major impact on the data generated by the application so care should always be used when accessing this tool. By default, the application has an interactive user interface. Equivalent command-line arguments are provided to execute any supported functionality so that this tool can be used for automation within Unix scripts. The following is a list of functionality supported by this application:

* Initialization
* Adjusting Control Parameters such as Verbosity Level
* Display Selective Status Information
* Force Load of Input Version Information
* Force Execution of the Screen File Clean Up

### Command Line Parameters

The *RTPDEBUG* program is executed from the command line by a user or through the Maestro scheduling system. In either case, the following syntax should be followed:

RTPDEBUG {options} {command(s)}

where [command(s)] consists of

zero to multiple interactive commands that should be executed automatically by the tool. Commands are executed from left to right. A failure of a single command will prevent the execution of any other commands in the list. Commands on the command-line are optional.

where {options} include one or more of the following commands:

| **Switch** | **Description** | **Action If Missing** |
| --- | --- | --- |
| /E | Causes the program to exit once all commands included on the command line (e.g., /P option) have been executed. Failure codes are returned as exit codes. | Enter Interactive Mode |
| /N | Prevents the program from sending any information to the local console. In this mode, the program exists once all commands include on the command line (e.g., /P option) have been executed. Failure codes are returned as exit codes. | Information Written to Standard Out |
| /O=? | Establishes wait time in seconds for a response from the command processor on any commands submitted for processing. “?” represents a number of seconds. | Default if not specified is 30 seconds. |

### Interactive Commands

In general, the *Debugging Tool* is designed to be an interactive tool used for the support of the operational application. Executing the tool without any options always brings it into interactive mode. From that point, the user enters at the prompt titled “Command(“?” for help):” a comms

RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC

RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC

RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC

RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC

RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC

RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC

RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC RDC

|  |  |  |  |
| --- | --- | --- | --- |
| **Command** | **Subcommand** | **Description** | **Daemon Status** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| Run | V?? | Requests that the Daemon run a background task to load a new input format version into shared memory. | Operational |
| Run | Screen Clean | Requests that the Daemon runs the *screen clean* process that removes any older STDOUT files from the system. The number of files retained is based on the SCREEN\_FILE\_RETENTION parameter contained within the RTPSUPPORT.ini file. | Operational |
| Terminate | N/A | Requests that the Daemon normally terminate. The tool waits until the Daemon has achieved a final shutdown state before returning control to the user. | Operational |
| Exit | N/A | Exits the debugging tool. | Any |
| ? or Help | {command} | Provides help for the use of the system. Typed alone it provides a list of command available. Typed with another command causes the system to list a more detailed description of the specified command if it is available. | Any |

# Data Structures

This chapter is designed to provide detailed information about all data within the RTP system. It defines, at the field level, both content and structure. Data definitions are provided for system specific transactions, inter-process communication, physical non-database data files, and logical definitions of database tables.

## Special Transactions

There are two special transactions used to validate the completeness of each batch input file being processed by RTP. Both transactions are the only ones whose format is not database driven and are used only within the *Batch* stage of the data flow process. The following sections define the minimum requirements for the data contained within these transactions.

### Logical Batch Open (LBO)

The *Logical Batch Open (LBO)* transaction marks the beginning of a collection of transactions for a given business date. All LBO transactions must have a corresponding *Logical Batch Close* transaction. The field titled “Base Sequence” represents the transaction header as defined in section 2.1.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Byte Position | Data Type | Name | Description | Case Sensitive |
| 0 – 9 | Char(10) | Base Sequence | Fixed Value “@LBO002600” | Y |
| 10 – 11 | Char(2) | Source System | Two character code that uniquely identifies the system that created this batch. Valid codes are defined below. | Y |
| 12 – 17 | Numeric  999999 | Batch Sequence Number | This sequential number should uniquely identify a logical batch for a given source system. This number should not be reused before six months have elapsed. | N |
| 18 – 25 | Date | Business Date | This is the business date of the transactions contained within this logical batch. Business date is typically updated at each nightly close to the next valid date. The format is MMDDYYYY. | N |
| 26 – 37 | Date/Time Stamp | Create Stamp | This is the date and time the batch file was created. The format is MMDDYYYYHHMM where the hours are specified in military time. | N |

**Valid Source Systems**

|  |  |
| --- | --- |
| **Code** | **System** |
| WG | Walt Disney World GSA |
| WM | Walt Disney World Matra |
| DM | Disney Cruise Line Micros |
| DR | Dining Reservation System |
| SE | Special Events System |
| MO | Mail Order System |

### Logical Batch Close (LBC)

The *Logical Batch Close (LBC)* transaction marks the end of a collection of transactions for a given business date. This transaction must always follow a corresponding *Logical Batch Open* transaction.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Byte Position | Data Type | Name | Description | Case Sensitive |
| 0 – 9 | Char(10) | Base Sequence | Fixed Value “@LBC002A00” | Y |
| 10 – 11 | Char(2) | Source System | Two character code that uniquely identifies the system that created this batch. Valid codes defined in section 5.1.1. | Y |
| 12 – 17 | Numeric  999999 | Batch Sequence Number | This sequential number should uniquely identify a logical batch for a given source system. This number should not be reused before six months have elapsed. | N |
| 18 – 25 | Date | Business Date | This is the business date of the transactions contained within this logical batch. Date is in the form of MMDDYYYY. | N |
| 26 – 31 | Numeric  999999 | Transaction Count | This is a count of the number of transactions contained within this logical batch. | N |
| 32 – 37 | Numeric  999999 | Retail Count | This is a count of the number of retail transactions contained within the logical batch. | N |
| 38 – 41 | Numeric  9999 | Open Count | This is a count of the number of store opens contained within the logical batch. | N |

## Inter-process Communication

The RTP System consists of a series of executables each of which consists of one or more processes. The sharing of information and system coordination within RTP is based on several forms of Unix Inter-Process Communication. This section provides detailed information on the data content along with associated usage guidelines.

### Master Shared Memory Segment

The *master shared memory segment* is designed to house statistical and control information that is shared across all the processes that make up the RTP system. Shared memory is used to provide the best performance while allowing the operating system to distribute the execution of the application across multiple physical processors. Each element within the segment typically has a single process assigned to make updates with multiple processes reading each field. A semaphore controls access to the segment to prevent read/write conflicts. The segment is created as the first step of daemon initialization and is populated based on command line parameters and data within the *RTPINIT.INI* file. The following table defines the information contained within the segment as declared in the module “RTPMASTER.h”:

| **Offset[[8]](#footnote-8)** | **Field Size** | **Data Type** | **Field Name** | **Description** |
| --- | --- | --- | --- | --- |
| **System Control Structure** (SMSYSTEMCONTROL) | | | | There is a single instance of this information at the start of the shared memory block. Updated by the daemon initialization process, this structure contains information that directly controls the overall operation of the system. |
| 0 | 2 | int | iAccessCtrlSem | Contains the ID for the access control semaphore used to prevent multiple updates to the master shared memory segment. |
| 2 | 1 | Byte | bDaemonStatus | Flag indicating the current operating status of the daemon. At present valid values include:  O = Initialization In Progress  16 = Daemon Process Starting  32 = Repository Resynchronization  64 = Business Management Reload  128 = Normal Operation  192 = Terminating Child Processes  200 = Awaiting Child Terminations  240 = Shutdown In Progress  250 = Final Stage Semaphore Gone  Constants are defined in RTPMASTERSEG.h and begin with the prefix “SCD\_” for each of these codes. |
| 3 | 1 | Byte | bVerboseLevel | Flag indicating how much information should be written to the screen file during operation. Valid values include:   * 0 = Silent Operation * 1 = Normal Operation * 2 = Statistical Information * 3 = Individual Transaction Information * 4 = Detailed Operational Information * 5 = Maximum Information   Constants are defined in RTPMASTERSEG.h and begin with the prefix “VBL\_” for each of these level. |
| 4 | 2 | Unsigned Short | usInVersionCnt | Count of the number of input file versions presently loaded into memory. |
| 6 | 2 | Unsigned Short | usInVersionLmt | Count indicating the maximum number of simultaneously supported input formats. This field is loaded from the RTPINIT.INI file and defines the number of structures allocated during initialization for SMINPUTVERSION structures. |
| 8 | 2 | Unsigned Short | usHostCnt | Count of the number of hosts that may be receiving data from this system. |
|  |  |  |  |  |
| **System Statistics Structure** (SMSYSTEMSTATISTICS) | | | | There is a single instance of this information immediately following the system control structure in the shared memory block. Updated by all processes, this structure contains qualitative data describing the overall operation of the system. |
|  |  |  |  |  |
|  |  |  |  |  |
| **Array of Input Version Structure**(s)  (SMINPUTVERSION) | | | | There is an array of instances of this information immediately following the system statistics structure in the shared memory block. One entry exists for each input version being actively supported. Updated by the RTPINIT command processor, each structure contains the necessary information to access a shared memory segment that contains field level definitions for a specific input version. |
| 0 | 3 | Char | cVersion | Contains a NULL terminated string that is the unique version identifier for transaction based on this file input format. |
| 3 | 1 | BOOL-EAN | bActive | Flag that is set to true if the input version is active and available for use by RTPBATCH. |
| 4 | 2 | Int | iMemID | Contains the shared memory ID for the segment that contains the details of this input format version. |
|  |  |  |  |  |
| **Array of Destination Host Structure(s)**  (SMDESTINATIONHOST) | | | | There is an array of instances of this information immediately following the input version structure(s) in the shared memory block. One entry exists for each input version being actively supported. Updated by the RTPINIT command processor, each structure contains the necessary information to access a shared memory segment that contains field level definitions for a specific input version. |
| **Control Structure**  (Control) | | | | Maintained by the host interface process, the sub-structure contains data that controls the operation of the individual host interface. |
| 0 | 1 | Byte | bHostStatus | Flag indicating the current operating status of this instance of the host translator. At present valid values include:  O = Initialization In Progress  16 = Fork Complete Awaiting EXEC  32 = Initializing Translator  48 = Loading Configuration Information  64 = Resynchronization In Progress  128 = Normal Operation  240 = Shutdown In Progress  250 = Shutdown Complete |
| 1 | 31 | Char(31) | cHostName | Null terminated character string that contains the name of the host system that this instance feeds. This field is used for reporting purposes. |
| 32 | 2 | Char(2) | cHostCode | Two character ID code that identifies the host within the configuration database. This field is loaded by RTPINIT from the configuration. |
| 34 | 4 | PID | pHostID | Process ID for the instance of the translator once it has been forked. A value of zero means the process has not yet started or has terminated. |
|  |  |  |  |  |
|  |  |  |  |  |
| **Statistics Structure**  (Statistics) | | | | Maintained by the host interface process, the sub-structure contains qualitative data about the operation of the individual host interface. |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

### Screen FIFO Protocol

The unidirectional screen FIFO is used to receive information from all the processes making up the RTP system and store them interlaced into a single sequentially file. The resulting file is specifically designed to help with troubleshooting by providing a combined activity report from all processes. Information received on this FIFO is the result of each process redirecting STDOUT and STDERR into this pipe. All information should consist of properly terminated ASCII text lines with a normal length not to exceed 80 characters.

### Log FIFO Protocol

This unidirectional FIFO is used to submit major events to the logging process such as error or warning messages. Individual processes submit a header along with a data buffer, if appropriate. The logging process then evaluates, files, and if appropriate passes the information along to another support system such as paging or service center. The standard log header structure is defined in the RTPLOGFIFO.h file. Standard methods within the *CSupport* class allow individual processes to easily submit messages. The header structures are defined as follows:

| **Offset** | **Field Size** | **Data Type** | **Field Name** | **Description** |
| --- | --- | --- | --- | --- |
| **Log Header Structure** (SLOGFIFO) | | | | |
| 0 | 1 | Char | cMarker | Must always contain a “|”. This field is used to verify byte alignment on the pipe to make sure we are not reacting to invalid commands. Failure to detect this character will terminate the application. |
| 1 | 1 | Char | cLevel | Contains the level of message being received. Primarly used for messages valid types include:  E – Error Message  W – Warning Message  I – Information Message  LTC\_MESSAGE are the only packet types that support this field. All others should set it to NULL. |
| 2 | 2 | Int | iLogCode | Contains the LTC\_ log type code that indicates the type of message this packet represents. |
| 4 | 4 | Long | lErrNum | Contains the message number associated with this packet if appropriate. LTC\_MESSAGE are the only packet types that support this field. All others should set it to NULL. |
| 8 | 4 | Long | lSize | Contains the size of the data immediately following this log header. |
| 12 | 4 | pid\_t | Pid | Contains the process ID of the process that submitted this request. |

Utilizing this structure, the following message types are supported by the RTP daemon:

**LTC\_MESSAGE (8)**

This type of packet contains a message that is reporting a major event or issue that occurred in one of the related processes. Information is either stored for future review or reported to another support system such as paging or service center. The data block associated with this message type contains the message text.

**LTC\_CHILDDOWN (200)**

This type of packet contains a message from a child termination signal handler to report to the console that the process is going down. This is used to maintain reentrant safe operation within the signal handler.

**LTC\_SIGTERM (220)**

This type of packet is used in a SIGTERM signal handler to report receipt of the signal to the console.

**LTC\_TERMINATE (255)**

This command initiates a shutdown of the logging process. No additional messages are processed once this packet is received.

### Command FIFO Protocol

The bi-directional[[9]](#footnote-9) command FIFO is used to submit commands/requests to the core daemon that impact its operation or initiate an update to shared memory. A standard command and response structure as defined in the RTPCMDFIFO.h is used to simplify the operation of these processes. Commands consist of a standard header followed by a command specific data buffer. Responses include a header only. The command and response header structures are defined as follows:

| **Offset** | **Field Size** | **Data Type** | **Field Name** | **Description** |
| --- | --- | --- | --- | --- |
| **Command Structure** (SCMDFIFO) | | | | |
| 0 | 1 | Char | cMarker | Must always contain a “~”. This field is used to verify byte alignment on the pipe to make sure we are not reacting to invalid commands. Failure to detect this character will terminate the application. |
| 1 | 1 | Byte | bCmdCode | Contains the DCD\_ command code that directs the change and/or processing required of the daemon. |
| 2 | 4 | Long | lSize | Contains the size of the data immediately following this command header. |
| 4 | 4 | PID | pidRespond | Contains the process ID of the process that submitted this request. Responses are returned through a FIFO named by appending this value to the end of the command FIFO name. NULL in this field indicates no response expected. |
| **Response Structure** (SRESPFIFO) | | | | |
| 0 | 2 | Int | iRespCode | Contains the DCR\_ primary response code that indicates the basic result of the requested command. |
| 2 | 4 | LONG | lExtendedCode | Contains an extended error code from the daemon if available. This code may be from the database or the OS based on the command and response code. |

Utilizing these structures, the following commands are supported by the RTP daemon:

**DCD\_LOADINVERSION (8)**

This command initiates the load of a new Input Version configuration into an independent shared memory segment for active use by the batch processes. This command includes a two-character data buffer that contains the version code as defined in the *base sequence* of the transaction. Valid responses to this command include:

|  |  |  |
| --- | --- | --- |
| **Label** | **Value** | **Description** |
| DCR\_OK | 0 | New segment has been loaded successfully and is available through the master shared memory segment. |
| DCR\_DUPLICATEVERISON | 1000 | The segment was not loaded since it is already available in a shared memory segment. |
| DCR\_LIMITEXCEEDED | 1100 | The segment was not loaded because the number of segments in shared memory exceeds the limit defined within the RTPINIT.INI file. |
| DCR\_DATABASEERROR | 4000 | The segment was not loaded because of a database error. In this case, the extended code contains the database return code. |
| DCR\_OSFAILURE | 4100 | The segment was not loaded because of an operating system error. In this case, the extended code contains the operating system error code. |
| DCR\_CLASSERROR | 9000 | This indicates that an error occurred reading the response packet. The “GetLastError” method will provide details on the cause of this return code. |

**DCD\_SET\_VERBOSITY (64)**

This command resets the daemon verbosity level adjusting the amount of information written to the screen file. This command includes a single byte data buffer that contains the level value between zero and five. Valid responses to this command include:

|  |  |  |
| --- | --- | --- |
| **Label** | **Value** | **Description** |
| DCR\_OK | 0 | New segment has been loaded successfully and is available through the master shared memory segment. |
| DCR\_LIMITEXCEEDED | 1100 | This indicates that the verbosity level is not between the expected values of 0 and 5. |
| DCR\_BADPARAMETER | 1700 | This indicates that the new level was not provided with the submitted command. |
| DCR\_INVALIDSTATE | 2000 | This indicates that the verbosity change occurred after the daemon is in a final shutdown stage. |

**DCD\_NVTR\_INTEGRITY (96)**

This command is sent from the Transaction Stage of the system to the daemon to signal that it detected an integrity issue either within the active NVTR file or the shared memory. This command typically requires a re-initialization of all business control logic that normally occurs during daemon initialization. This command has no data and does not normally return a response code.

**DCD\_NVTR\_ROLLOVER (104)**

This command is sent from the Transaction Stage of the system to the daemon to signal that the size of the transaction repository has exceeded the soft-limit and that the system should roll-over to a new repository file. This command has no data and does not normally return a response code.

**DCD\_TERMINATE (128)**

This command initiates the controlled termination of all the processes associated with the daemon. There is no data buffer associated with this command. Successful completion of this command indicates that the daemon has started the shutdown process. It does not necessarily indicate that the daemon shutdown was completely successful. Valid responses to this command include:

|  |  |  |
| --- | --- | --- |
| **Label** | **Value** | **Description** |
| DCR\_OK | 0 | The shutdown process has been successfully requested. |
| DCR\_INVALIDSTATE | 2000 | The shutdown request was unsuccessful being the daemon is in a state (e.g., Awaiting Child Terminations) that prevents the command from being processed. |
| DCR\_CLASSERROR | 9000 | This indicates that an error occurred reading the response packet. The “GetLastError” method will provide details on the cause of this return code. |

**DCD\_CHILD\_DOWN (196)**

This command is sent from a “signal” to update process information when a child process terminates. This is a one-way command that neither receives nor expects a response. The “pidRespond” field, for this command only, contains the process ID of the child that was terminating. The data section contains a single LONG value that is the status code of the terminating process.

**DCD\_CLEAN\_STDOUT (200)**

This command is sent from the screen process to request that the STDOUT file clean process is started in the background. The command processor starts the background process and monitors its completion.

## Environment Variables

A limited number of environment variables are utilized and/or required by the system to better enable the application. The following is a complete list of the environment variables required along with a basic description:

* **INIPATH=$..$** where the $..$ is a series of semi-colon (or colon on HPUX) separated directory paths that may contain INI files used by the RTP system. The *CINIDecode* class first searches the local directory and then all the directories in the order listed within this environment variable.

## Physical Non-database Files

One of the key methods the RTP system uses to store information that is needed beyond the current operating processes is a set of standard Unix files. These non-database files are typically used to facilitate higher performance for data that is accessed sequentially element by element or at a specific known offset. This section details information on the data content of each of these files along with, if appropriate, the associated usage guidelines.

### Non-volatile Transaction Repository

### Suspense Repository

### Initialization Files (INI)

The INI files used within the RTP system are designed to allow easy parameter modification without the need for code alterations. A standard class, “CINIDecode” has been developed to load and decode these files based on the following standard guidelines.

* Files are completely ASCII text based.
* Parameters exist as a combination of a variable and its corresponding value similar to information contained within environment variables. For example, a “wait time” parameter set to 5 seconds might appear as *WAIT\_TIME=5*.
* Blank lines are always ignored.
* File must end with the appropriate end of line character (CR, LF, or CR/LF).
* Spaces are ignored unless contained within double quotes. For example, *TITLE=BIG PICTURE* would result in the variable TITLE being set to “BIGPICTURE” unless it was entered as follows: *TITLE=”BIG PICTURE”*.
* Any line where the first non-space character is a “#” is considered a comment and ignored by the decoder during the processing of the file.
* Variables that are unknown to the application will result in a failure of the application.
* Variables that are not contained within the INI file will be assigned their defined default values.

The next several sections provide detailed documentation on each of the parameters supported within the various INI files used by the RTP system.

#### RTPSUPPORT – Support Infrastructure Parameters

The *RTPSUPPORT.INI* file is used by all executables to house control values used for access to the master shared memory segment, configuration of the debugging tools, configuration of metrics, and other common control/statistical functionality. The following table defines each of the variables supported by this application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| VERBOSITY | Contains a single character initial and/or default verbose level for the daemon and all future batch processes. The default value of “1” causes the processes to provide minimal operational information in the screen file. | Char | “0” – “5” | “1” |
| WORKING\_  DIRECTORY | Contains the working directory for the daemon during normal operation. If not specified the daemon will run where it is started. “rtpinit.bootlog” is ALWAYS created in the directory where the daemon is started. | Char | Any valid pathname | “.” |
| TRANSLATOR\_DIRECTORY | Contains the path used to locate the translator programs. All translation programs must be in the same directory and share their working directory with the daemon. If not specified the assumed location is in the WORKING\_DIRECTORY for the daemon. | Char | Any valid pathname | See Desc. |
| PROJECT\_ID | This numeric value is used with System V IPC to generate unique keys. | Numeric | Between 0 and 255 | None |
| PATTERN | This character string must contain a series of numbers that are used as a pattern to protect control information within RTP. Do NOT change this value without clear understanding of the security impact. | String with numeric values only. | Any valid numeric string | None |
| START\_FIFO\_  WAIT | Contains the wait time in seconds that we should wait after gaining access to the shared memory segment for the screen FIFO before failing the open. | Numeric | Between 1 and 240 | 8 Seconds |
| START\_  MASTER\_WAIT | Contains the wait time in seconds that we should wait for access to the shared memory segment. | Numeric | Between 4 and 240 | 20 Seconds |
| COMMAND\_  FIFO | Contains a NULL terminated string that defines the fully qualified pathname of the FIFO used to submit commands to the daemon. | Char | Valid System Pathname | None |
| SCREEN\_FIFO | Contains a NULL terminated string that defines the fully qualified pathname of the FIFO used for the reroute of STDOUT. | Char | Valid System Pathname | None |
| LOG\_FIFO | Contains a NULL terminated string that defines the fully qualified pathname of the FIFO used to submit commands to the daemon. | Char | Valid System Pathname | None |
| FIFO\_CMD\_  WAIT | Contains the number of seconds the command processor waits for a FIFO to open that is being used to respond to a command. The open is attempted every second until either the time expires or a connection is established. | Numeric | Zero or any Positive value. | 1 |
| FIFO\_WAIT | Contains the number of seconds to wait for a FIFO to open during the initialization process. The open is attempted every second until either the time expires or a connection is established. This value is used for the Command, Log, and Screen FIFOs. | Numeric | Zero or any Positive value. | None |
| VERSION\_  LIMIT | Contains a numeric count that limits the number of active input file version formats that the system will support. | Numeric (Count) | Between 1 and 255 | 5 |
| LENGTH\_ BASE62 | Contains a true/false flag that controls whether the transaction length is in hexi-decimal or base 62. A value of true indicates its base 62. | Char | True or False | False |
| LOCATION\_  AVG\_SIZE | Contains number of bytes that should be allocated initially for each active location in the business management segment. | Numeric | Between 32 and 65,535 | 512 Bytes |
| DEVICE\_AVG\_  SIZE | Contains number of bytes that should be allocated initially for each expected physical device. | Numeric | Between 32 and 65,535 | 128 Bytes |
| MASTER\_SEG\_  LIMIT | Contains the maximum number of bytes that should be allocated in the master memory segment. | Numeric | Between 8K and 16M | 128K Bytes |
| SEG\_  EXTENSION | Contains the size of each master memory segment extension that should occur if the business management system requires additional storage. | Numeric | Between 8K and 1M | 128K Bytes |
| MEMORY\_  DEBUG\_LEVEL | Contains a numeric value that indicates how much memory information should be written to the screen file. This value is used ONLY within CMASTER for master shared memory management. | Numeric | Between 0 and 7 | 0 |
| SCREEN\_FILE\_  PATTERN | Contains a filename format used to create the standard output and error capture files. Flags within this string are based on standards used for the ANSCI C “strftime” command (e.g., %Y=4 digit year, %m=2 digit month, and %d=2 digit day). | Char | Valid path once strftime variables are replaced. | None |
| SCREEN\_FILE\_  RETENTION | Contains the number of screen files maintained on the system. Files are sorted based on creation date followed by name with the oldest and lowest value being removed first. For this reason, it i s recommended that the screen pattern use year, month, and day in that order as part of the filename so that the oldest is always removed first regardless of file date. | Numeric | Between 2 and 60 | 7 |
| TICK\_  INTERVAL | Contains the number of seconds between tick checks that control how often the system writes a tick mark to the screen file during an inactive period. Interval checks always start at mid-night. | Numeric | Between 900 and 3600 | None |
| SCREEN\_  WARNING | Contains a number of megabytes after which the system sends a page to indicate that the screen file is becoming extremely large. An out of range value will result in a setting of default value of 1G. | Numeric | 1 to 2048 | 1024 |
| BLANK\_LINE\_WARNING | Contains a number of lines that the screen process can receive blank before it generates a warning message. | Numeric | 1,000 to 1,000,000 | 5000 |
| MESSAGE\_  LOG\_  RETENTION | Contains the number of days worth of log messages to retain within the system. Messages older than this date will be permanently removed from the system. An invalid or missing value will result in a value of 7 being assumed. | Numeric | Between 1 and 60 | 7 |
| OPERATION\_  LOG\_  RETENTION | Contains the number of days worth of logical batch and file run information retained within the systems. Records older than this date will be permanently removed from the system. An invalid or missing value will result in a value of 180 being assumed. | Numeric | Between 30 and 365 | 180 |
| FILE\_HISTORY\_RETENTION | Contains the number of days worth of RTPDTRAN output file history maintained within the database for audit and relationship resolution. An invalid or missing value will result in a value of 45 being assumed | Numeric | Between 21 and 90 | 45 |
| CREDIT\_  LOCATION\_  RETENTION | Contains the number of months worth of credit card usage by location statistics to be maintained within the database. Any invalid or missing value will result in the default value of 7 months being used. | Numeric | Between 2 and 48 | 7 |
| REPORT\_  REQUEST\_  RETENTION | Contains the number of days worth of reporting engine requests to be maintained within the database. Any invalid or missing value will result in the default value of 7 days being used. | Numeric | Between 1 and 60 | 7 |
| GSA\_CONTROL\_RETENTION | Contains the number of days worth of GSA control information used by TLTS in the form of 15S5 and user information. | Numeric | Between 7 and 365 | 30 |
| PENDING\_TXN\_RETENTION | Contains the number of days worth of Pending Transaction information retained in the system. | Numeric | Between 3 and 365 | 7 |
| INFORM\_  LOAD\_WAIT | Contains the number of seconds to wait for an input format to get loaded into shared memory by the daemon. | Numeric | Between 15 and 3600 | 60 |
| RECYCLE\_  WAIT | Contains the number of seconds to wait after a daemon shutdown is detected to attempt a reconnect of the support class. | Numeric | Between 5 and 3600 | 45 |
| NVTR\_FILE\_  PATTERN | Contains the basic pattern for the filename that make up the NVTR. Daily the file is opened using this base with the extension “.YYYYXXX” where “Y” is the four-digit year and “X” is the three-digit day of the year. | Char | Valid basic filename without a path | “NVTR” |
| NVTR\_PATH | Contains the path used to create the transaction repository. | Char | Valid path name | “./” |
| NVTR\_SIZE\_  LIMIT | Contains the soft limit in megabytes for a single physical NVTR file. The system will attempt a roll-over to a new physical file once the repository exceeds this value. Roll-over will NOT necessarily occur at exactly the provided limit. Transactions NEVER span multiple files. | Numeric | Between 10 and 2,032 – default is used if value is out of range. | 1024 |
| NVTR\_  BUNDLE\_  BACKUP | Contains a number of calendar days that the system should backup from the calendar activation date to capture business date information during a recovery of a bundle file. The larger the number the longer the time required initializing the daemon. The smaller the number the greater chance some transactions will be missed. | Numeric | Between 0 and 14 | 1 |
| DAEMON\_WAIT | Contains the number of seconds to wait before failing if a support call is waiting for the daemon to either recycle or start. If invalid or not provided the system defaults to 30 seconds. | Numeric | Between 10 and 32767 | 30 |
| CHILD\_WAIT | Contains the number of seconds to wait for the translators to shutdown before shutting the daemon the rest of the way down. If invalid or not provided the system defaults to 25 seconds. | Numeric | Between 10 and 32767 | 25 |
| DTRAN\_DEBUG | Contains a debugging flag for the Data Driven Translator. If set to “Y” RTPINIT does not start the translators. Afterwards, the translators can be run individually as a separate process routing output to the screen and terminating upon receiving a interrupt without taking down the daemon. | Char | “Y” or “N” | “N” |
| BUNDLE\_  MINIMUM | Contains a byte size below which RTPDTRAN will not perform a bundle clean up. Bundle clean up is deferred until the size of the file grows beyond this limit. | Numeric | Between 4,096 and 256,000 | 32,767 |
| CACHE\_LIMIT | Contains a megabyte size limit for caching of translation data before information starts getting traded out. | Numeric | Between 1 and 1,024 | 1 |
| MEMORY\_LOG\_KEY\_POINTS | Contains a Y/N flag that indicates whether applications should log key memory allocation information into the standard output file. This parameter is designed to be used in conjunction with the NEW/DELETE memory allocation tracking function. Not all modules within RTP honor this request – at publication RTPDTRAN was the only executable that utilized this parameter. | Char | “Y” or “N” | “N” |
| KEYBASE | Contains the encryption key used by RTP to encode and decode the database control information. This value is combined with internal control information to form the key. | Char | Any String | Not Listed |
| KEYMAP | Contains a series of “1”s and “0”s that are used in the creation of the key for encryption of database access information. | Char | A string containing a series of 1s and 0s. | Not Listed |
| BL\_DUMP\_FILE | Contains the fully qualified path/filename information for the business logic shared memory dump file used to reduce restart time. | Char | Any valid path and filename | ./rtpinit.bldump |
| BL\_HOLD\_TIME | Contains the number of days that an entry is held in memory for business management logic. After that point, it is pulled from the database archive. | Numeric | Any value between 1 and 60 | 2 |
| BL\_ARCHIVE\_ RETENTION | Contains the number of days worth of archive detail to be maintained in order to allow reloading of duplicate check control information. After this point, duplicates will no longer be rejected. | Numeric | Any value between 2 and 365 | 180 |
| BL\_QIDX\_  CONTROL | Contains the fully qualified path/filename information for a file which is created and maintained by the quick index facility indicating where it last processed information within the NVTRs. | Char | Any valid path and filename | ./rtpinit.qidx |
| OPWAIT\_  TIMEOUT | Contains the minimum number of seconds to wait for the shared memory status flag to go operational before failing a transaction submission. This is listed as a minimum since the actual time will be rounded to the next “Check Interval” or timer tick as appropriate. If the value is out of range or missing then the default is assumed. | Numeric | Any Positive Numeric Value | 45 |
| OPWAIT\_  CHECK\_  INTERVAL | Contains the number of seconds to wait between checking the status flag in the shared memory in order to submit a transaction. If the value is out of range or missing then the default is assumed. | Numeric | Any Positive Numeric Value | 8 |
| CPWAIT\_ TIMEOUT | Contains the number of seconds to wait for a response from the daemon to a command request handled within the support class. A default of 16 seconds is used if this value is missing or invalid. | Numeric | Any Positive Numeric Value | 16 |
| PRODUCTION | Contains a Y/N flag that indicates if this instance is a production instance. This function disables certain remote control functions that should be limited to test. | Char | “Y” or “N” | Y |
| REGION\_NAME | Contains a string that uniquely identifies the region represented by this configuration instance. At publication, valid values included PROD, QA, QA2, QA3, and QA4. | String | Any unique string up to 4 characters | UNKN |
| TASK\_QUEUE | Contains the pathname for the file used to contain queue’d task information. This file is used by RTPINIT when task limits have been exceed to store the next instance up. | Path and/or Filename | Any valid path and filename | ./rtpinit.taskq |
| FIFO\_CLEAN | Contains the pathname only for the fifo clean up that should occur during system initialization to ensure that the application starts clean without any carry over of old pipes. A missing value will prevent the system from starting. | Path | Any valid path | NONE |
| SECAUDIT\_ INTERVAL | Contains the number of second after which the security audit process awakens and checks for new data to migrate to the NVTR regardless of any signals. The default value is used if unspecified or out of range. | Numeric | Value between 8 and 3600 | 300 |
| SECAUDIT\_  STORE | Contains the location key to be used processing the security audit log information. There is no default value assumed. | String | Any valid location key | NONE |
| SECAUDIT\_  DOWNWAIT | Number of seconds to wait after the system drops out of operational before closing the security audit semaphore down and removing from the system. If unspecified or out of range the system uses the default value. | Numeric | Value between 4 and 120 | 30 |
| CCS\_ENCRYT\_WAIT | Number of seconds to wait after the system is up and through batch recovery before starting a re-encryption of the CCS\_MASTER account number. If unspecified or out of range than the default is assumed. | Numeric | Value between 60 and 21,600 | 300 |
| CONTROL\_  BACKUP | Path and pattern used to build backups of the BIN files prior to migration to a new encryption key. The default value is used if not present and is: ./%Y%m%d%H%M%S. | String | Any valid path followed by a pattern | ./%Y%m%d%H%M%S. |
| CONTROL\_LIST | List of keys separated by a comma that can be used for the rotation of keys. No default is provided and this is required. | String | Any list of key numbers | N/A |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

In addition, the *RTPSUPPORT.INI* file has zero or more repeating configuration lines that drive the purge process for the pending transaction table. Each line starts with the label “PNDT\_PURGE” and identifies each unique purge criteria. The following table contains a description for each of the fields provided in a “PNDT\_PURGE” line as they would appear in the control file.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| SOURCE\_ID | This string contains the source ID entered by the system when creating the transactions. It is used to control which type of key data is removed from the tables. This value must always be provided. | String | Any valid string | NONE |
| STATUS\_CODE | Contains a single character status code to which this purge criteria applies. | Character | Any valid status code as defined in the design. | S |
| DAYS\_KEPT | Contains a number of days worth of transaction information that is maintained within the system. The purge occurs when the last modified date/time is older than the number of days to be kept. This function is time of day dependent. | Number | Between 2 and 365 | 7 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

#### RTPINIT – System Management Parameters

The *RTPINIT.INI* file is used by the daemon to control the operation of external processes associated with or managed by the daemon. The first defined line label is “TASK” which identifies each unique instance of a process. The following table contains a description for each of the fields provided in a “TASK” line as they would appear in the control file.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| NAME | Name that uniquely identifies this process instance. This label is used for logging and to demand execution. | String | Any unique string. | None |
| DESCRIPTION | Human readable string used in logging and for display to an end user. | String | Any | None |
| EXECUTABLE | Executable module without any path information. A“@” symbol indicates that this record applies to one of the internal clean up routines. By default, it runs the primary or ZERO routine; otherwise, it runs the routine associated with the single digit immediately following the “@” symbol. Currently, that includes:  **0** = Main System Clean Up  **1** = NVTR Archiving Process  **2** = Pending Transaction Clean Up  **3** = File System Clean Up  **4** = Credit Data Clean Up  **5** = Log Database Clean Up  **6** = Generic Table Clean Up  **9** = TLTS/GSA Clean Up | String | Any valid executable module | None |
| OVERRIDE PATH | Contains a special directory used to start the application if the binary is not contained in the current working directory with the other binaries. | String | Any valid path | Working Directory |
| COMMAND LINE ARGUMENTS | Contains the command line arguments required for this instance of the application. A ^ can be placed within the string which is replaced by the process ID of the command processor. A % sign can be placed within this string that is replaced at run-time for an “on demand” process with provided data. Either of these can be used once and only once within the overall command line argument information. | String | Any valid switches | None |
| START UP MODE | Contains the start up mode for this application. The following modes are valid:  L – At system load  O – On demand  Q – On demand with queue  D – Daily one time  I – Interval  T – Time Range once  P - Start if processing inbounds  R – Start if ready for new batch  M – Monthly  W – Weekly  Z – Pre-Daemon Start | Char | As defined in the description | L |
| START UP DATA | Contains a string of data that is specific to the start up mode. The following is a list of format guidelines for each related mode:  D – Contains the time of day normally triggered in the form HHMM.  I – Contains the number of minutes between executions.  T – Contains a time range during which the task is run once daily. The form for this data is HHMM-HHMM.  M – Contains information on run in the form “$HHMM#1-#2” where the $ is a Y if you want it to run immediately if a new task; HHMM is the 24 hour clock time to run the process (1200 assumed if missing); #1 day of the month to run (1st is assumed if not provided; and #2 is the number of months to advance before the next run (1 is assumed if not provided).  W – Contains information on the run in the form $HHMM where $=day of week (0 or Sunday assumed); and HHMM is the time to run.  **WARNING**: Scheduling time based runs after 11:45pm and before mid-night can result in missed or duplicate triggers. This time period should, therefore, be avoided. | String | As defined in the description | None |
| TERMINATION ACTION | Contains the action to be taken if the application terminates. The following is a list of valid values:  N – No Termination Action  S – Shutdown System  E – Error Report Only  R – Restart Always  F - Restart on Failed Exit Code  I – Input On-line Process | Char | As defined in the description | I |
| RELATIONSHIP | Contains the relationship between the daemon and the resulting process. A value of “C” indicates it is a child directly managed while “I” creates an independent process. | Char | Either C or I | C |
| ACTIVE LIMIT | Contains a count of the number of process instances that can be outstanding for this task entry. This ONLY applies to children tasks. A value of ZERO allows for unlimited instances. | Numeric | Positive Value | 0 |
| RETART COUNT LIMIT | Contains the number of times the process can be restarted before it is considered to have failed and shuts down the daemon. | Numeric | Between 1 and 32 | 2 |
| RETART COUNT RESET TIME | Contains the number of seconds to wait after each restart before clearing the restart count in seconds. | Numeric | Between 4 and 300 | 30 |
| STOP\_TIMING | Contains a flag indicating the types of termination signals sent to a child process during termination of the daemon. Valid values include:  “E”arly – Sends a SIGTERM at the start of the shutdown process.  “L”ate – Sends a SIGTERM at the end of the shutdown process as core daemon tasks (e.g., logging start terminating, etc…).  “B”oth – Sends a SIGTERM at the start of the shutdown process and a SIGKILL at the end of the shutdown process.  “X”clude – Does not send termination signals through the normal process. The system only signals this task down like “B” if a hard failure occurs. | Char | E, L, X or B | E |
| INPUT\_FLAG | Contains an optional flag that indicates if this task is an input process. These tasks are shutdown if inputs are disabled. The options for this command are as follows:  “K” causes the task to be killed if running  “F” causes the task to be finished with no additional starts – requests are queued until inputs are restarted  “T” causes the task to be termed if running  “N” causes the task to not be in the input group  If nothing is specified or any other value then those listed above the task is assumed to be a value of “N” making it NOT part of the input processes. “K” should only be used for system start up tasks with “F” only being used for on demand or timed tasks. “T” can be used with either category as appropriate. | Char | K, F, T, N | N |

The second defined line label is “CLEAN” which identifies any daily file clean up required. Each line represents a specific file search pattern and indicates the amount of information to be retained. The following table contains a description for each of the fields provided in a “CLEAN” line as they would appear in the control file.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| PATH | Contains the path in which the files reside that should be removed. This parameter contains path information ONLY ending with a ‘/’. | String | Any valid path | None |
| PATTERN | Contains a pattern with “\*” wild cards that identify the files that should be removed from the system. | String | Any valid file pattern | None |
| EXCLUSION | Contains a pattern for files that should excluded from processing. This string can be used to reduce the number of files matching the pattern. All files are included if this parameter is blank. | String | Any valid file pattern | None |
| RETENTION TYPE | Contains a flag indicating how we determine the retention level. A value of “F” indicates that we should keep the number of files specified starting with the newest files. A value of “D” means we should remove files that are more then the specified number of days old regardless of the number of files. | Char | F or D | None |
| RETENTION QUANTITY | Contains the number of files or days to be retained as indicated by the RETENTION\_ TYPE parameter above described. | Numeric | 0 to 32,767 | None |

The third defined line label is “COMPRESS” which identifies any daily file that should be compressed to conserve space.. Each line represents a specific file search pattern and indicates the amount of information to be retained uncompressed. The following table contains a description for each of the fields provided in a “COMPRESS” line as they would appear in the control file.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| PATH | Contains the path in which the files reside that should be removed. This parameter contains path information ONLY ending with a ‘/’. | String | Any valid path | None |
| PATTERN | Contains a pattern with “\*” wild cards that identify the files that should be removed from the system. | String | Any valid file pattern | None |
| EXCLUSION | Contains a pattern for files that should excluded from processing. This string can be used to reduce the number of files matching the pattern. All files are included if this parameter is blank. | String | Any valid file pattern | None |
| RETENTION TYPE | Contains a flag indicating how we determine the retention level. A value of “F” indicates that we should keep the number of files specified starting with the newest files. A value of “D” means we should remove files that are more then the specified number of days old regardless of the number of files. | Char | F or D | None |
| RETENTION QUANTITY | Contains the number of files or days to be retained as indicated by the RETENTION\_ TYPE parameter above described. | Numeric | 0 to 32,767 | None |

The fourth defined line label is “TABLE” which identifies any daily table clean up required. Each line represents a specific table search pattern and indicates the amount of information to be retained. The following table contains a description for each of the fields provided in a “TABLE” line as they would appear in the control file.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| NAME | Contains the name of the database table to be cleaned by this entry. | String | Any valid table name | None |
| LOCK\_FIELD | Contains a field name within the table that is set to “Y” if the row should always be retained. | String | Any valid field within the above table | None |
| DATE\_FIELD | Contains the name of a field within the table that is to be used as the basis for the purge process. This field should be a date field. | String | Any valid field within the above table | None |
| DAYS | Contains the number of days worth of information to be retained in the table. Valid values must be at least one. 14 is assumed if a value outside of this range is specified. | Numeric | Short Integer | 14 |

#### RTPLOGIC – Logical Batch Configuration Parameters

The *RTPLOGIC.INI* file is used by the *CLogicalBatch* class to house basic configuration information about the format of the logical batch open/close transactions. This initialization file allows for flexibility in the structure, verification rules, and overall format of these non-database transactions. All information is read as the class is initialized. The following table defines each of the variables supported by this application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| LB\_START | Defines the key information contained within the logical batch open. Multiple parameters are housed on this single parameter line separated by commas as follows: 3 character transaction code, offset for the start of the ID, the size of the ID, the size of the system identifier that is contained at the start of the ID string, the offset of the chronological identifier, length of the chronological identifier, the offset of the creation stamp, and the length of the creation stamp. | Char | Must contain eight fields separated by commas | LBO, 10, 8, 2, 18, 8, 26, 12 |
| LB\_END | Defines the key information contained within the logical batch open. Multiple parameters are housed on this single parameter line separated by commas as follows: 3 character transaction code and offset for the start of the ID. Length is assumed to match open since they must be equal. | Char | Must contain two fields separated by a comma | LBC, 10 |
| LB\_OVERALL\_VERIFY | Defines the offset and length of the overall transaction count contained within the logical batch end transaction. This count is verified against the actual count prior to any processing. This verification does not include LBO or LBC in the count. | Char | Must contain two fields separated by a comma | 26, 6 |
| FILE\_BAD\_ WITH\_ ORPHANS | A value of “Y” indicates that the logical batch class should consider the entire file bad if there is one or more orphans within the input file. “N” causes the system to process all other transactions moving the orphans into the exception management system. | Char | Y or N | N |
| TRANSACTION\_COUNT | Defines the key information used to determine which transaction codes are counted within the logical batch and verified against the end transaction data. Multiple instances of this parameter may exist within the INI file. In addition, multiple parameters are housed on each parameter line separated by commas as follows: 3 character transaction code, offset to the start of the count in the end transaction, and the size of the count. | Char | Must contain three fields separated by commas | RTT, 32, 6 |
| FILE\_BAD\_ON\_FAILED\_COUNT | A value of “Y” indicates that the logical batch class should consider the entire file bad if one or more invalid count exists within a single logical unit. This failure will prevent the processing of any data within the file. “N” causes the system to fail only the effected logical unit. (Optional) | Char | Y or N | N |
| ARCHIVE\_ DIRECTORY | Contains the qualified path where compliant files should be moved once they have been successfully processed. This is a required field that will cause the system to shutdown on failure. | String | Must be valid path | None |

#### RTPBATCH– Batch Configuration Parameters

The *RTPBATCH.INI* file is used by the RTPBATCH executable to define default values for processing transaction files. This initialization file allows the ability for the application to change standard default processing values without requiring code modifications. These values are **only** used if override parameters are not entered at the command line. The following table defines each of the variables supported by this application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| RECURSE\_  COUNT | This value indicates the number of times the system should repeat its attempt to process the files that match the given path and pattern. A value of ZERO causes it to be attempted only once and a value of 99 causes it to repeat until the directory is empty. | Numeric | Between 0 and 99 inclusive | 0 |
| RECURSE\_  WAIT | Contains the number of seconds to wait after each pass through the directory. This value only applies if *RECURSE\_COUNT* is set to a value other than ZERO. | Numeric | Between 1 and 900 | 1 |
| RETRY\_COUNT | Contains the number of times that a transaction should be resubmitted to the transaction phase IF the response indicates that this transaction is being submitted out of sequence. | Numeric | Between 0 and 16 | 2 |
| RETRY\_WAIT | Contains the number of seconds to wait between retries. | Numeric | Between 1 and 900 | 15 |

#### RTPDTRAN – Data Driven Translator Configuration Parameters

The *RTPDTRAN.INI* file is used by both RTPDTRAN and RTPSHUTL executables to define control variables such as working path and time-outs that are common to all instances of the translator and file distribution programs. This initialization file allows the ability for the application to change common processing values without requiring code modifications. The following table defines each of the variables supported by this application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| BUNDLE\_PATH | Contains a fully qualified base path and file name for the bundling files used during the translator process. Extensions are added to this base to create a unique control file along with a series of files one per host. | Char | Any valid pathname | “./BUNDLE” |
| BUILD\_PATH | Contains a path used for building the final files for transmission to the host system. This drive will have heavy activity during bundling of summarized data so its performance is critical. | Char | Any valid pathname | “./” |
| CLEAN\_PATH | Contains a path used for creating temporary files used during bundle clean up process and the final resting place for BUNDLE archives. | Char | Any Valid pathname | “./” |
| ARCHIVE\_PATH | Contains a path where files that are transferred successfully (other than a MOVE transfer type) are placed once the transfer is complete. This value must NEVER match the BUILD\_PATH. | Char | Any valid pathname EXCEPT that in BUILD\_  PATH | “./” |
| CLEAN\_DAY\_  BUNDLE | Contains the number of days to be combined into a single archive file. Each archive file contains in the name the 4-digit year along with a bundle reference based on this number of days. For example, if the value is 4 then the first four days will be in 2002000 and the next four days will be in 2002001. The default is used if the value is either missing or invalid. | Numeric | Between 1 and 30. | 7 |
| KEEP\_BUNDLE\_FILE | Contains the number of bundle file archives retained for each host after a daily clean up process is completed. The oldest files are permanently removed until the number is less than or equal to this value. An invalid or missing value causes the system to assume a setting of 5 files. | Numeric | Between 1 and 20 | 5 |
| FTP\_PATH | Contains a path used for final deposit of files to be picked up for transport to a batch host system. Access to this drive is critical. | Char | Any valid pathname | “./” |
| BUNDLE\_ID\_  LIMIT | Contains the maximum ID that should be used to identify a bundle during the build process. | Numeric | Number up to 2G containing all 9s. | 999999 |
| MAXIMUM\_ ACTIVE\_XFER | Contains the maximum number of simultaneous host transfers that can be active at any given time. A value of ZERO indicates that there is no limit on the number active. Values out of range will be set to the closest limit. | Numeric | Number between 0 and 48. | 4 |
| MAXIMUM\_ XFER\_WAIT | Contains the number of seconds to wait for a transfer to complete before paging the on-call for support. Values out of range will be set to the closest limit. | Numeric | Seconds between 8 and 3,600 | 120 |
| MAX\_FILE\_  LIMIT | Contains the maximum number of files that can be opened at one time during the bundling process. This parameter is used to prevent DTRAN from reaching its file system limit. | Numeric | Number 32 or greater | 384 |
| CPROC\_WAIT\_ TIME | Contains the maximum number of seconds shuttle should wait for CPROC to acknowledge a request for a demand task start. The higher the number the more chance of delaying other transfers the lower the number increases chance of demand task failure. | Numeric | Between 2 and 600 seconds | 8 |
| PGP\_PROGRAM | Contains the fully qualified path and file name for the Gnu PGP command line tool. The default value is used if this parameter is not provided. | Char | Any valid file and path name | /usr/ local/ bin/gpg |
| PGP\_PARAM | One or more entries that are used to build the parameter list for PGP. A “%” is replaced with the trusted key and a “~” is replaced with the recipient ID. There are no default values and no parameters are required. | Char | Any valid parameter | N/A |
| REQUESTOR\_  USERID | Contains a string with a valid user ID that should be listed as the originator of the report. If not provided then the default is “RTP01”. | String | Any Valid RTP User ID | RTP01 |
| DISTRIBUTION | Contains a pattern used for the group name to which the report is being distributed. The group name pattern must contain a “%s” where the host code should be inserted. If not provided the default is used of “Report\_HostScan\_%s”. | String | Any Valid RTP Group Name | Report\_Host Scan\_ %s |
|  |  |  |  |  |

#### RTPOPENV – OpenView Interface Parameters

The *RTPOPENV..INI* file is used by the RTPOPENV executable and the RTPINIT logging process to house control values used for ticket and paging interface to OpenView. These parameters are loaded during initialization and require a reset of the system to modify the operation of the application. The following table defines each of the variables supported by this application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| FIFO\_NAME | Contains a NULL terminated string that defines the fully qualified pathname of the FIFO used to submit the OpenView requests. | Char | Valid System Pathname | None |
| FIFO\_WAIT | Contains the number of seconds to wait for the OpenView process to become active. | Numeric | Between 2 and 60 | 4 |
| APPLICATION | Contains a string used to identify the application impacted within the OpenView system. | String | None | RTP |
| OBJECT | Contains the object string passed to the OpenView system. | String | None | “ “ |
| GROUP | Contains the group string passed to the OpenView system. | String | None | RTP |
| TARGET | Contains the target production support group which should be the receiver of the OpenView action (e.g., page, e-mail, ticket) | String | None | T7prod |
| DEBUG\_MODE | Contains a control flag that if set to true disables the actual paging process. While disabled, the system writes all interfaces to a file titled: "./send\_page%s.log" where %s is the current date. | Char | TRUE or FALSE | FALSE |
| IGNORE\_  ERROR | Contains a control flag that causes all messages with a severity level of “E” to be ignored by the process. | Char | TRUE or FALSE | FALSE |
| IGNORE\_  WARNING | Contains a control flag that causes all messages with a severity level of “W” to be ignored by the process. | Char | TRUE or FALSE | FALSE |
| IGNORE\_INFO | Contains a control flag that causes all messages with a severity level of “I” to be ignored by the process. | Char | TRUE or FALSE | FALSE |
| IGNORE\_STATS | Contains a control flag that causes all messages with a severity level of “S” to be ignored by the process. | Char | TRUE or FALSE | FALSE |
| {Error Prefix + Message Number} | A single line within the file is entered for each message that needs to have action controlled (e.g., “RTP1000”). The data on each line has one or more of the following flags:  P = Page  T = Ticket  E = E-mail  N = No Action  If the message doesn’t exist in the file, the system assumes that it should take the action of paging and creating a ticket. | Char | Any combo of P, T, E, or N | PT |
|  |  |  |  |  |
|  |  |  |  |  |

#### RTPMON – RTP Monitoring Parameters

The *RTPMON.INI* file provides parameters that control the operation of the RTP monitoring application. These parameters can be used to tailor the rules surrounding monitoring and the associated notifications. The following table defines each of the variables supported by this application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| SCREEN\_FILE\_ PATTERN | Contains a filename format used to create the standard output and error capture files. Flags within this string are based on standards used for the ANSCI C “strftime” command (e.g., %Y=4 digit year, %m=2 digit month, and %d=2 digit day). | Char | Valid path once strftime variables are replaced. | ./monitor |
| SCREEN\_FILE\_ PATTERN2 | Contains a filename format used to create the standard output and error capture files for the ServiceGuard monitoring routine. Flags within this string are based on standards used for the ANSCI C “strftime” command (e.g., %Y=4 digit year, %m=2 digit month, and %d=2 digit day). | Char | Valid path once strftime variables are replaced. | ./sgmon |
| BLACKOUT | Indicates a daily time period during which system checks are not active. There can be any number of entries beginning with this tag. Each single parameter line contains multiple parameters separated by commas as follows: start time in the form HHMM, the end time in the form HHMM. In both cases, HH is the hour in 24-hour format and MM is the associated minutes. If not provided then no blackout is assumed. | Char | Must contain two time values separated by commas | N/A |
| FTP\_GROWTH\_LIMIT | Number of consecutive checks that must show a sustained pattern of growth in the FTP directory before a page is initiated to the on-call. | Numeric | Must be a positive integer | 3 |
| DU\_PATTERN | Contains the information used to control monitoring of disk utilization. There can be any number of entries that are individually checked. Each single parameter line contains multiple parameters separated by commas as follows: first parameter is a path name which can be used to identify the device being monitored, number of percentage point change during a single monitoring interval that warrants a page, number of percentage points change in a given day that warrants a page, an overall percentage limit that warrants a page, and finally a text string pattern that can be used to identify the information line. A sample line might appear as follows:  DU\_PATTERN=/opt/apps/rtp,2,4,80,":%used block" | Char | Must contain five fields as defined in the description separated by commas. | N/A |
| MONITOR\_ INTERVAL | Number of seconds between system reviews. This value indicates how many seconds to wait after each check before initiating the next analysis run. | Numeric | Any positive Number | 600 |
| SG\_MONITOR\_ INTERVAL | Number of seconds between ServiceGaurd system reviews. This value indicates how many seconds to wait after each check before initiating the next analysis run. | Numeric | Any positive Number | 600 |
| CONTROL\_FILE | Contains a fully qualified path and filename for the control file used to manage monitoring on the fly. Default path and filename are used if not specified. | Char | Any valid path and file name | ./rtpmon.ctrl |
| SG\_CONTROL\_  FILE | Contains a fully qualified path and filename for the control file used to manage whether fail-over will occur if the system is shutdown. Default path and filename are used if not specified. | Char | Any valid path and file name | ./rtpmonsg.ctrl |
| SG\_CPROC\_  WAIT | Contains the number of seconds to wait for a response from the RTPINIT command processor to ensure that it is operational and responsive. The default is 20 seconds which is assumed if not specified or out of range. | Numeric | Numeric between 8 and 120 seconds | 20 |
|  |  |  |  |  |

#### RTPMQSERIES – MQ Series Support Interface Parameters

The *RTPMQSERIES.INI* file provides parameters that control the RTPMQSRV interface application’s use of MQ Series. It defines not only the attributes of the queues but also default values used by the application. The following table defines each of the variables supported by this application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| QMGR | Contains the name of the queue manager to which this application should connect. | Char | Any valid queue manager name | None |
| QMGR\_  VERSION | Contains a string that describes the version of queue manager that is currently in operation. Valid values at time of publishing include 5.1, 5.2, 5.3, or CURRENT. The default is to use the “current” version. | Char | 5.1, 5.2, 5.3, or Current | Current |
| SI\_Q\_INBOUND | Contains the fully qualified name of the support interface inbound queue. | Char | Any valid queue name | None |
| SI\_Q\_  OUTBOUND | Contains the fully qualified name of the support interface outbound queue. | Char | Any valid queue name | None |
| TX\_Q\_  INBOUND | Contains the fully qualified name of the real-time RTP compliant transaction inbound queue. | Char | Any valid queue name | None |
| TX\_Q\_  OUTBOUND | Contains the fully qualified name of the real-time RTP compliant transaction outbound queue. | Char | Any valid queue name | None |
| BT\_Q\_  INBOUND | Contains the fully qualified name of the RTP batch file inbound queue. | Char | Any valid queue name | None |
| LG\_Q\_  INBOUND | Contains the fully qualified name of the RTP logging queue used by RTP components not located on the primary server as a tool to submit information to the master application log as well as OpenView as appropriate. | Char | Any valid queue name | None |
| RESPONSE\_TTL | Contains the number of second that the responses should be held on the outbound queue before they are dropped as no longer valid. A value outside of the valid range will cause the system to use the default. | Numeric | Between 4 and 90, inclusive | 30 |
| CONTENTION\_WAIT | Contains the number of seconds to wait before retrying a queue manager connection if the reason for a prior failure is contention. This feature is designed to prevent restart failure due to timing within the queue manager. A default of 4 is used in case of a missing or invalid value. | Numeric | Between 2 and 30, inclusive | 4 |
| STOP\_WAIT | Contains the number of seconds to wait before retrying a queue manager connection if the reason for a prior failure is the queue stopping or unavailable. This feature is designed to allow the restart of the queue manager without shutting down RTPINIT. A default of 60 us used in case of missing or invalid value. | Numeric | Between 30 and 300, inclusive | 60 |
| RESTART\_  LIMIT | Contains the number of times to reattempt connection if the initial queue manager connect command fails due to contention or quiesing. This feature is designed to prevent restart failure due to timing within the queue manager. A default of 8 is used in case of a missing or invalid value. | Numeric | Between 5 and 60, inclusive | 8 |
| MQ32\_APPL | Contains the path of the 32-Bit MQ series interface application that provides RTPSUPP access to MQ series. This entry could either by a local filename or fully qualified program path. | String | Any valid executable name | ./rtpmqsrv |
| TERM2KILL\_  WAIT | Contains the number of seconds after issuing a SIGTERM to the MQ interface before issuing a SIGKILL. The time is specified in seconds. | Numeric | Any numeric | 4 |
| BATCH\_XFER\_BUFFER | Contains the number of bytes to allocate for the maximum buffer size coming from a batch interface. A missing or invalid value will result in the default being used. | Numeric | Between 4096 and 4M | 32768 |
| DEBUG\_  PATTERN | Contains a fully qualified path and file name with a substitution value of “%s” embedded within the string. The substitution value is replaced with the destination filename. This pattern is used to save off a file partially transferred before a failure occurred. | String | Any valid qualified path | None |
| WORKING\_  PATH | Contains the fully qualified path for storage of the temporary files used during the transfer. These files are NEVER considered complete. | String | Any valid qualified path | ./ |
| FORMAT | File contains one or more lines with this prefix that defines control parameters for each format received over this queue. Detailed field definitions are contained in the table following these variable definitions. | Array of Strings | See following table | N/A |
| RTPBATCH\_  START\_WAIT | Contains the number of seconds to wait after sending the RTPBATCH start command to RTPINIT for a response. This parameter only impacts RTPEDS and RTPGSA. If missing or invalid the default of 8 seconds is assumed. | Numeric | Between 4 and 60, inclusive | 8 |
| EDS\_LOWEST\_TERMINAL | Contains the lowest terminal that can be used by the EDS program for assigning sequence numbers. If the value is missing or invalid then the default is used. | Numeric | Between 1 and 9999, inclusive | 9100 |
| EDS\_HIGHEST\_TERMINAL | Contains the highest terminal that can be used by the EDS program for assigning sequence numbers. If the value is missing or invalid then the default is used. | Numeric | Between 1 and 9999, inclusive. Must be greater than the lowest value. | LOW-EST value plus 99 |
|  |  |  |  |  |

The *RTPMQSERIES.INI* file also contains a series of one or more lines that begin with the label “FORMAT”. These lines define the formats accepted and the action that should be taken once the entire file is received on the local host. The following table defines each of the variables supported by this application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| FORMAT CODE | Contains a numeric value in string form that represents the file type sent to the service by the client. This code is used to indicate the action the system should take for this file. | Numeric | Any positive value | N/A |
| DESTINATION PATH | Contains the destination path in which this file should be stored once it is successfully received at the host. | String | Any valid path | None |
| TASK NAME | Contains the name of the task as defined in the *RTPINIT.INI* file that should be run to process this file once received successfully. | String | Any valid task name | None |
| COMMAND LINE ARGUMENTS | Contains the command line arguments included with the file name as instance specific data for the task being started. | String | Any valid switches | None |
|  |  |  |  |  |

#### RTPCMMQ – Communication Manager MQ Series Parameters

The *RTPCMMQ.INI* file provides parameters that control the RTPMQ32IF interface application’s use of MQ Series. It defines standard attributes such as timing for all queues. No parameters within this file are queue object specific. The following table defines each of the variables supported by this application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| CONTENTION\_WAIT | Contains the number of seconds to wait before retrying a queue manager connection if the reason for a prior failure is contention. This feature is designed to prevent restart failure due to timing within the queue manager. A default of 4 is used in case of a missing or invalid value. | Numeric | Between 2 and 30, inclusive | 4 |
| STOP\_WAIT | Contains the number of seconds to wait before retrying a queue manager connection if the reason for a prior failure is the queue stopping or unavailable. This feature is designed to allow the restart of the queue manager without shutting down RTPINIT. A default of 60 us used in case of missing or invalid value. | Numeric | Between 30 and 300, inclusive | 60 |
| RESTART\_  LIMIT | Contains the number of times to reattempt connection if the initial queue manager connect command fails due to contention or quiesing. This feature is designed to prevent restart failure due to timing within the queue manager. A default of 8 is used in case of a missing or invalid value. | Numeric | Between 5 and 60, inclusive | 8 |
| DEFAULT\_ BUFFER | Contains the number of bytes to allocate for the maximum buffer size coming from a standard MQ Series interface. A missing or invalid value will result in the default being used. | Numeric | Between 4096 and 4M | 32768 |

#### RTPWRAP – Wrap Conversion Utility Parameters

The *RTPWRAP.INI* file provides configuration information for wrapping non-compliant data files for import into RTP. There are two types of entries in this file one describing how to identify record types and the second determine how to implement transaction key variables. The following table defines the fields within the *RECD* lines that define how the system will identify transaction codes associated with input records. The parameters are comma delimited from left to right as defined in the table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| Source System Code | Source system code used to associate this configuration with specific process within the RTP system. |  |  |  |
| Transaction Code | Three character transaction code associated with this specific line of configuration |  |  |  |
| Pattern String | Pattern which uniquely identifies this type of record in the source file. |  |  |  |
| Pattern Offset | Offset of pattern within the source system file for this specific record type. |  |  |  |
| Business Date Offset | Offset of the business date within this record within the source system file. |  |  |  |
| Remove Duplicates | Flag indicates if duplicate records should be removed |  |  |  |
| Automatic Open Flag | Uses terminal number 9999 – never include in leasing for WRAPPED files |  |  |  |
| Location Key Offset | Only used if auto-open enabled |  |  |  |
| Location Key Size | Only used if auto-open enabled |  |  |  |

The second type of lines begins with *FLD* and is used to populate the transaction key fields required within each record. Each source system/transaction code combination has between zero and four entries representing the chronology key, selling location, physical device, and/or sequence number. The following table describes the comma-delimited values from left to right.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| Source System Code | Source system code used to associate this configuration with specific process within the RTP system. |  |  |  |
| Transaction Code | Three character transaction code associated with this specific line of configuration |  |  |  |
| Value Type | This single character identifies the type of field defined by this line of the configuration. Valid values include:  B – Business Date  S – Shop/Store Number  T – Terminal Number  N – Sequence Number |  |  |  |
| Field Size | Size of the field being identified with this line of information. |  |  |  |
| Equation | Used to define the value that should be placed within this field. “{$INLINE}” system parameter is used to access the source information. The “@” symbol is used to perform a special internal calculation for terminal numbers and sequence numbers. A string such as “@10;8;18;5;1000-9000” in the terminal field will cause the system to automatically assign terminal numbers. The “10;8” defines the offset and size of the business date within the source record. The “18;5” defines the offset and size of the shop number within the source record. The final range “1000-9000” in our example defines the range of terminal numbers that can be used. |  |  |  |
| Default | (Shop Number ONLY) Equation used if base equation fails |  |  |  |

#### RTPWRAP – Comma Delimited File Support

The *RTPWRAP.INI* file also provides support to convert a comma delimited file into a fixed length file before providing the standard wrap service. This capability allows support for non-compliant data files in a comma delimited format being imported into RTP through standard interfaces. There are two additional types of entries in this file one describing characteristics of the source file and the second determine the length of each of the delimited fields. The following table defines the fields within the SYS*D* lines that define whether or not this type of processing applies. The parameters are comma delimited from left to right as defined in the table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| Source System Code | Source system code used to associate this configuration with specific process within the RTP system. |  |  |  |
| Delimited | Flag set to “Y” or “T” if the original source file is delimited rather than fixed length. Any other value assumes the file is already fixed field length. |  |  |  |
|  |  |  |  |  |

The second set of lines in this type of interface begin with *COM* and control how fields are converted from their original comma delimited format into a fixed field length equivalent. Each line within this file represents a single field within the original source file. The following table defines the fields within the *COM* lines that defines how data is converted. The parameters are comma delimited from left to right as defined in the table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| Source System Code | Source system code used to associate this configuration with specific process within the RTP system. |  |  |  |
| Sequence Number | This numeric value is a zero based index for the source field that should be used as the basis of this fixed field entry. For example, the first field in the comma delimited source file would be represented by a ZERO “0” in this field. This field is automatically placed in the same position in the output file. In this example, it would be at the start of the line. |  |  |  |
| Field Length | This numeric value indicates the length that should be used in the output file for this field. |  |  |  |
|  |  |  |  |  |

#### RTPRENG – Reporting Engine Control Parameters

The reporting engine to establish standard-operating parameters uses this configuration file. Control information is general such as maximum number of threads. The following table defines each of the variables supported by this application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| MAXIMUM\_ THREADS | This parameter sets the maximum number of threads that the engine can start to generate individual reports at any given moment. If no value is specified then the system assumes a single thread. | Numeric | Between 1 and 64 | 1 |
| TEMP\_PATH | Contains the default path used to store the reporting files while they are being built. | String | Any valid path | ./ |
| NVTRMGMT\_  PROGAM | Contains a qualified (from the daemon working directory) path/file combination that is used to start the NVTR Management system. If missing or invalid the default is assumed. | String | Any valid path and filename | ./rtpnvtrmgt |
| NVTRMGMT\_ PARAMETERS | Contains one or more options passed to the NVTR Management program during start up. If invalid or missing then the default options are assumed. You are limited to a maximum of 9 separate parameters within this string. | String | Any character string | -N |
| DEBUG\_MODE | Contains a Y/N flag that indicates if the reporting engine is operating in debugging mode. If set to “Y” then the system will open all temporary files without the corresponding delete. This operation will cause the system to leave these files for later review. | Char | Y/N | N |
| EA\_LINEITEM\_TOTAL | Contains a mathematical expression used to determine the line item total for this transaction. | String | Any valid RTP expression | None |
| TR\_ORDER\_OK | Contains an equation that when evaluated as true causes the system to consider the transaction valid for use in the ticket retrieval reports. This expression can be overridden through use of the “A” parameter in the report request. | String | Any valid RTP expression | None |
| TR\_VAR\_OK | Contains an equation that when evaluated as true causes the system to include the ticket in the variance report. The “A” parameter does NOT override this equation. | String | Any valid RTP expression | None |
| TR\_VAR\_  HEADER | Contains an equation that when run against an RTT produces a value for the transaction based on header information only. | String | Any valid RTP expression | None |
| TR\_VAR\_  ITEM | Contains an equation that when run against an RTT produces a value for the transaction item, tax, discount detail – this basically represents the revenue generating components of the transaction. | String | Any valid RTP expression | None |
| TR\_VAR\_  TENDER | Contains an equation that when run against an RTT produces a value for the tender information – this represents the receipts associated with the transaction. | String | Any valid RTP expression | None |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

#### RTPEXFIX – Batch Exception Fix Parameters

The batch fix utility is designed to correct know exception errors in bulk. Upon completion, the system requests a report through the reporting engine. The parameters in this file control the distribution of that report. The following table defines each of the variables supported by this application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| REQUESTOR\_  USERID | Contains a string with a valid user ID that should be listed as the originator of the report. | String | Any Valid RTP User ID | None |
| DISTRIBUTION | Contains either a group name or user ID to which the report is being distributed. A string alone indicates a group name; placing a ‘+’ at the start of the name indicates that a specific user ID is provided. | String | Any Valid RTP Group Name or User ID | None |
|  |  |  |  |  |
|  |  |  |  |  |

#### RTPTLTS – TLTS Processor Control Parameters

The TLTS control processor is designed to process information feed from GSA via the TLTS application. This file contains control parameters used by this program to allow customization for a specific implementation. The following table defines each of the variables supported by this application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| ARCHIVE\_PATH | Contains a fully qualified pathname into which processed TLTS input files are placed once processing is complete. | String | Must be a valid pathname | None |
| DBATCH\_TASK | Contains the name of the task that is started when a store is closed by the tlts process. A missing or blank value will cause the system not to start any task. | String | Must be a defined task | None |
| CASHOUT\_SKU | Contains a string that identifies the SKU information required to generate a corporate gift card cash out. These values are global. The string contains two values separated by a “|”. The first is the SKU number the second is the four digit department class. This field is required for system operation. There is no default value. | String | Must be valid SKU, “|”, and Department/Class | None |

#### RTPCM – Communication Manager Control Parameters

Communication’s Manager is one of the most sophisticated components within the RTP system. This subsystem is designed to effectively manage all external communications required in the operation of the RTP system. This file contains control parameters used by this program to allow customization for a specific implementation. The following table defines each of the variables supported by this application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| DOWN\_WAIT | This optional parameter defines the number of seconds after issue a normal termination to child process that the subsystem should force a shutdown through KILL signals. This time should be long enough to allow the normal termination of the application. | Numeric | Seconds between 8 and 32767 | 24 |
| CM\_ROUTER\_ ADDR | Contains one or more address and port combinations on which the system should bind the router to for command and control. The first string is the address (\*=Local Host) followed by a “|” followed by the port number. The default value is used if it is not specified. | String | Any valid host or address followed by a valid port number. | \*|8444 |
| CM\_  ALTERNATE\_ ADDR | Contains one or more address and port combinations for alternate communication manager instances which this system can use. The first parameter is the name of the instance followed by a “|” followed by a string with the address (\*=Local Host) followed by a “|” followed by the port number. There is no default values. | String | Any valid host or address followed by a valid port number. | None |
| RETRY\_WAIT | Contains the number of seconds to wait between attempted reconnections to CM Router. Default is used if out of range or unspecified. | Numeric | Number of seconds between 4 and 3600 | 30 |
| RETRY\_LIMIT | Contains the number of times to try and attempt connection to CM router before failing the system. The default is used if out of range or unspecified. | Numeric | Number between 3 and 100; or “0” to try forever. | 3 |
| LOOP\_WAIT | Contains the number seconds to wait between reviews of listening port bind requirements. This controls the granularity for the system checking for the availability of ports for binding. The default of 8 seconds is used if nothing is specified. | Numeric | Number between 1 and 512 seconds. | 8 |
| NOPORT\_PAGE | Contains the number of seconds from system start up after which the system will page the on-call if we have not successfully binded to at least one of the defined ports. By default the value is set to 80 seconds just past the FIN\_WAIT2 value. The default is assumed if the value is out of range. | Numeric | Number between 15 and 65535 | 80 |
| NOPORT\_FAIL | Contains the number of seconds from system start up after which the system will shutdown if it has not successfully binded to at least one of the defined ports. This value MUST be greater than the NOPORT\_PAGE. The default value is used for out of range entries and is set to 300 seconds. | Numeric | Number between 16 and 65536 | 300 |
| PORTLEFT\_ PAGE | Contains the number of seconds from system start up after which a page is issued if and only if there are still ports left to be attached. The default value is 240 seconds and is used if the value is missing or out of range. | Numeric | Number between 15 and 65536 | 240 |
| RESTART\_ LIMIT | Contains the number of times an individual task should restart before the system should fail. The default count is 3. This value is assumed if an out of range value is presented. | Numeric | Number between 2 and 30 | 3 |
| RESTART\_ TIMEOUT | Contains the amount of time the system must go without a restart before the count is reset to zero preventing a system failure. The default value is 60 seconds. This value is used any time an out of range value is presented. | Numeric | Number of seconds between 30 and 600 | 60 |
| BASE\_IF\_WAIT | Number of seconds to wait from router start to initiation of the non-input bound interface tasks. The default is used if out of range or not specified. | Numeric | Number of seconds between 1 and 600 | 2 |
| INCH\_WAIT | Number of seconds to wait between checks of the inbound flag for starting input tasks. The default is used if out of range or not specified | Numeric | Number of seconds between 2 and 600. | 4 |
| INCH\_PAGE | Number of checks after which a single page is sent to the on-call to warn of a potential hang up or failure in the start process. The default is used if out of range or not specified. | Numeric | Number between 10 and 32767 | 150 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

#### RTPKEYSRV – Key Service Control Parameters

Key Service is used as a control interface to the key service provided by Stratus. This subsystem is designed to effectively manage controlled keys that are received from outside of the system. This file contains control parameters used by this program to allow customization for a specific implementation. The following table defines each of the variables supported by this application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| SCREEN\_FILE\_ PATTERN | Contains a filename format used to create the standard output and error capture files. Flags within this string are based on standards used for the ANSCI C “strftime” command (e.g., %Y=4 digit year, %m=2 digit month, and %d=2 digit day). | String | Any valid path and filename | ./keysrv |
| ADDR | Contains one or more address and port combinations on which the system should bind the router to for command and control. The first string is the address (\*=Local Host) followed by a “|” followed by the port number. The default value is used if it is not specified. | String | Any valid host or address followed by a valid port number. | \*|8444 |
| CLIENT\_WAIT | Number of seconds to wait for a client connection to the local key service. If invalid or missing the default value of 8 seconds is used. | Numeric | Any value between 2 and 60 seconds | 8 |
| KEYSERVER | Set of values used to establish key servers for looking up information. Parameters in order include low index, high index, priority, address, port, verification key, the encryption key code, and the numeric test flag value. Parameters are separated by a single comma. These entries are required for system operation no default is supported. | String |  | None |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

### Boot Log

During initialization, the Daemon process creates a temporary log file titled “*RTPINIT.BOOTLOG”* which records information that occurs during the start of the daemon but prior to any of the logging facilities coming on-line. It is expected that this file only be used if there is a failure during initialization.

### Database Control Information

For the sake of security, database access control information is stored in an encrypted file located within the INIPATH directory. The name of the file is RTPDBCTRL.BIN. Details on the encryption of this information are provided in a separate document for security reasons.

### Next Event Recovery File

The *Next Event* recovery file provides storage for task trigger information for RTPINIT based tasks[[10]](#footnote-10) that are either triggered daily or during a certain period during the day. The file contains one line for each task with the task name on the left and the next trigger time in the form YYYYMMDDHHMM on the right side.

## Logical Database Design

The following sections define the logical design of the database tables used by the RTP system.

### Operational Log

The operational log tables are designed to capture and provide information about the operation of the system. Data of this type is captured in one of two distinct types. The first is to store information about activities such as batch files received; logical batches processed; or log messages generated by one of the RTP processes. The second type is operational information that is updated and used by the applications within the system. Examples include sequence number/terminal ID leases, output file history for cross-reference, and aged integrity check information. Figure 11 shows the basic layout of the tables that make up the operational logs. Batch file information is maintained for 120 days past the last date the file or logical batch was processed. Messages are stored for 90 days from generation. The *CdbLOps,* *CDBLMsg,* *CDBLease,* and *CDBOut* classes are used to access and maintain information within these tables.



Figure 11.Operational Log Table Structure

**OPL\_FILE\_RUN Data Definition**

This table contains a single row for each time a file is processed by the RTPBATCH executable. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | INSTANCE\_ID | Numeric | Contains a key that uniquely identifies each row within the table. This value is also used to link the file to the logical batch run information. |
|  | FULL\_PATHNAME | VarChar | Contains the fully qualified path and filename that was processed in conjunction with this record. |
|  | PROCESS\_START\_  DATETIME | Date/Time | Contains the date and time that the processing began. This field coupled with the *FULL\_PATHNAME* should always be unique. |
|  | PROCESS\_END\_  DATETIME | Date/Time | Contains the date and time that processing completed. |
|  | STATUS | Char(1) | Contains the status of the file. Valid values included the following:   * I = In Progress * A = Awaiting Another Pass * S = Completed Successfully * F = Failed Entire File * B = Failed One or More Logical Batches * U = Unknown Software Issue * X = Indicates a System Failure |
|  | HOST\_NAME | Char(24) | The name of the host system that processed this input file. |
|  | PROCESS\_ID | Numeric | Process ID of the process that processed this input file. |
|  | AVG\_PROCESS\_  TIME | Numeric | The average amount of time used to process each transaction within the input file. |
|  | ORPHAN\_COUNT | Numeric | The number of orphans detected within this file during processing. |

**OPL\_LOGICAL Data Definition**

This table contains a single row for each unique logical batch processed by the RTPBATCH executable. A logical batch must be unique based on *Source System* and *Batch ID.* The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | SOURCE\_SYSTEM | Char(2) | Two-character field that uniquely identifies the type of selling device that submitted this batch. |
|  | BATCH\_ID | Numeric | A six-digit number that uniquely identifies this logical batch. It is expected that these numbers are not reused for at least six months. |
|  | BUSINESS\_DATE | Date | Contains the business date that is represented by this logical batch. |
|  | CREATE\_DATE | Date/Time | Contains the date and time that the batch was created at the source system. |
|  | OVERALL\_  STATUS | Char(1) | Contains the status of the file. Valid values included the following:   * I = In Progress * E = Awaiting An Additional Pass for Entire Batch * A = Awaiting An Additional Pass for One or More Transactions * S = Completed Successfully * F = Failed Entire Batch * B = Failed One or More Transactions * X = Indicates a System Failure |

**OPL\_LOGICAL\_RUN Data Definition**

This table contains a single row for each time a unique logical batch is processed through the RTPBATCH executable. All logical run records must have a corresponding *OPL\_LOGICAL* and *OPL\_FILE\_RUN* records. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | SOURCE\_SYSTEM | Char(2) | Two-character field that uniquely identifies the type of selling device that submitted this batch. |
|  | BATCH\_ID | Numeric | A six-digit number that uniquely identifies this logical batch. It is expected that these numbers are not reused for at least six months. |
|  | INSTANCE\_ID | Numeric | Contains the key that corresponds to the record in the *OPL\_FILE\_RUN* table that uniquely ties this process to the processing of a file. |
|  | INSTANCE\_  STATUS | Char(1) | Contains the status of the file. Valid values included the following:   * I = In Progress * E = Awaiting An Additional Pass for Entire Batch * A = Awaiting An Additional Pass for One or More Transactions * S = Completed Successfully * F = Failed Entire Batch * B = Failed One or More Transactions * X = Indicates a System Failure |
|  | TRANS\_  PROCESSED | Numeric | Count of the number of transaction processed within this logical batch. |
|  | TRANS\_FAILED | Numeric | Count of the number of transactions that did not process due to an error other than “already processed”. |
|  | TRANS\_PRIOR | Numeric | Count of the number of transactions not processed because they were previously submitted for processing. |
|  | AVG\_PROCESS\_  TIME | Numeric | Average number of seconds/milliseconds used to process each transaction within the logical batch. |

**OPL\_MESSAGE Data Definition**

This table contains a single row for each time a file is processed by the RTPBATCH executable. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | DATETIME\_  STAMP | Date/Time | Contains the date and time at which this message was written by the logging process to the database. |
|  | SEQUENCE\_  NUMBER | Numeric | Contains a unique number generated automatically by the system to make a unique key. |
|  | LOG\_CODE | Numeric | Contains the logging code received by the logging process. Valid values are defined in the RTPLOGFIFO.h file and prefixed with “LTC\_”. |
|  | PREFIX | Char(3) | Contains the error message prefix. This string allows you to identify the source of the failure code: RTP-Core System, RAP-API Interface, RUI-Web User Interface. |
|  | ERROR\_NUMBER | Numeric | Contains the message number as reported to the logging process. Valid values are between 0001 and 9999 inclusive and are defined in chapter 7 of this document. This field is only populated for LTC\_MESSAGE. |
|  | MSG\_LEVEL | Char(1) | Contains the level of message being received as follows: I=Information; W=Warning; E=Error. This field is only populated for LTC\_MESSAGE. |
|  | TEXT | VarChar  (192) | Contains the text of the message as submitted to the logging process. This message should be continuous without any forced line breaks. |
|  | HOST\_NAME | Char(32) | Contains the name of the host that submitted this message. |
|  | MODULE | Char(16) | Contains the name of the module that submitted this message. |
|  | HOST\_CODE | Char(2) | Contains a two-character host code, if applicable, that represents the host for which the RTPDTRAN instance was doing translations. A value of NULL is appropriate for all other modules. |

**OPL\_ARCHIVE\_LOCATION Data Definition**

This table contains a single row for location/chronology combination that has been monitored within the business logic. Records are written to this table once they have been inactive and removed from the shared memory segment. This information is retained until all the corresponding *OPL\_ARCHIVE\_DEVICE* records have expired and been removed from the system. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | LOCATION\_KEY | VarChar  (5-16) | Contains the location component of the transaction key. This value uniquely identifies the physical selling location. |
|  | CHRONO\_KEY | Date | Contains the chronological component of the transaction key which declares the related business date. |
|  | LAST\_STATUS | Char(1) | Contains the final mode of the location prior to the archive being written. |

**OPL\_ARCHIVE\_DEVICE Data Definition**

This table contains a single row for location/chronology/device combination that has been monitored within the business logic. Records are written to this table once they have been inactive and removed from the shared memory segment. This information is retained for a period defined in the ???.INI file. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | LOCATION\_KEY | VarChar  (5-16) | Contains the location component of the transaction key. This value uniquely identifies the physical selling location. |
|  | CHRONO\_KEY | Date | Contains the chronological component of the transaction key which declares the related business date. |
|  | DEVICE\_KEY | VarChar  (5-16) | Contains the device identification component of the transaction key. This value uniquely identifies the physical selling device. |
|  | NEXT\_SEQUENCE | Numeric | Contains the next sequence number that should be submitted for processing. |
|  | EXPIRE\_DATE | Date | Contains the date after which this record should be removed from the system. RTPINIT uses this date to clean up the archive file. |

**OPL\_OUTFILE\_HISTORY Data Definition**

This table contains a single row for each unique output file created by E. It is used for both debugging and by RTPSHUTL to rename output files based on the location group used during file creation. Information is maintained within this table for at least 45 days after creation. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | FILENAME | VarChar2 (96) | Contains the filename component of the output file created by RTPDTRAN. |
|  | GROUPING\_  LABEL | Char(8) | Contains an eight-character string that is associated with location when it is submitted to the host. Locations with a similar grouping label are transmitted to the associated host as a set. |
|  | CREATE\_STAMP | Date | Contains the date and time this entry was added to the table. This entry is used to purge aged information off the table. |

**OPL\_LEASE\_PROCESSID Data Definition**

This table is used to manage terminal sequence numbers for near real-time translations such as required for RTPEDS. The table contains a single row for each unique business date/location/terminal combination. Leases are either released when the program completes or when after 4 hours when the lease expires. Records are removed from this table once they have seen no activity for 14 days. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | CHRONO\_KEY | Date | Contains the chronological component of the transaction key which is the business date. |
|  | LOCATION\_KEY | VarChar  (5-16) | Contains the location component of the transaction key. This value uniquely identifies the physical selling location. |
|  | TERM\_ID | Numeric | Contains the terminal number represented by this record and used as the device key for RTP compliant transactions. |
|  | OWNING\_  PROCESS\_ID | Numeric | Contains the owning process ID for the process that currently owns this terminal number. A value of –1 indicates that the ID is no longer in use. |
|  | NEXT\_SEQUENCE | Numeric | Contains the next sequence number to be used by this terminal. |
|  | LEASE\_  EXPIRATION | Date | Contains the date/time stamp after which the lease is considered expired and the record should be released. |

### Exception Management

The exception management tables contain information about transactions that failed either an edit or audit while being processed by the RTP Transaction or Batch[[11]](#footnote-11) Stage. Transactions written to this repository are reviewed through the on-line interface, flagged corrected or deleted, and finally resubmitted through an automated process for reprocessing. Figure 11 shows the basic layout of the four tables that make up exception management. The EXC\_AUDIT table is only used by the user interface to track activity. All information is maintained within these tables for 120 days after the transaction has successfully been resubmitted or marked to purge. The *CDbExcept* class is used to access and maintain information within these tables.



Figure 12.Exception Management Table Structure

**EXC\_TRANSACTION Data Definition**

This table contains a single row for each unique transaction key that has at some point in time generated an exception. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | TRANSACTION\_  KEY | VarChar  (22-64) | Contains the transaction key that uniquely identifies the transaction submitted to RTP. This key is variable length since it is build based on configuration information contained within the input configuration tables described in section 5.5.4.1. |
|  | OVERALL\_  STATUS\_CODE | Char(1) | Contains a single character code that identifies the overall state of this transaction. The following is a list of valid status codes:   * ‘U’ser Review Pending * ‘A’waiting Data Correction * ‘C’orrections made but not Approved * ‘R’eady for Resubmission * ‘S’ubmitted for Processing * ‘P’rocessed * ‘D’eleted Never to be Processed * ‘X’fer to new Transaction Key |
|  | RGI\_ASSIGNED\_  TO | Numeric | Contains a numeric ID that uniquely identifies the group responsible for correcting this transaction. A value of NULL is entered after the transaction is placed in the Ready, Submitted, or Processed states. |
|  | CHRONO\_KEY | Date | Contains the chronological component of the transaction key as an independently searchable field from the transaction key. This field is used only to provide summary reporting within the on-line tool. |
|  | LOCATION\_\_KEY | VarChar  (5-16) | Contains the physical location component of the transaction key as an independently searchable field from the transaction key. This field is used only to provide summary reporting within the on-line tool. |
|  | DEVICE\_KEY | VarChar  (5-16) | Contains the physical device component of the transaction key as an independently searchable field from the transaction key. This field is used only to provide summary reporting within the on-line tool. |
|  | PROCESSED\_  DATETIME | Date/Time | Contains a date/time stamp for when the transaction was finally successfully processed by the RTP system. |

**EXC\_PROCESS\_INSTANCE Data Definition**

This table contains a single row for each unique version of a single transaction (identified by a row in EXC\_TRANSACTION) was processed. For example, a single row would be inserted in this table if an edit failed during the initial submission of the transaction. If after modifying and resubmitting the transaction, the transaction fails an edit again the system would create a new row representing this new failure. Multiple rows should NOT be inserted if the same version is accidentally processed multiple times. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | TRANSACTION\_  KEY | VarChar  (22-64) | Contains the transaction key that uniquely identifies the transaction submitted to RTP. This value must always correspond to a unique row in the EXC\_TRANSACTION table. |
|  | SEQUENCE\_  NUMBER | Numeric | Contains an automatically assigned number starting with zero that uniquely defines this version of the transaction. Number ZERO is always the original transaction as submitted to RTP. |
|  | STATUS | Char(1) | Contains a single character code that identifies the overall state of this processing instance. The following is a list of valid status codes:   * ‘U’ser Review Pending * ‘A’waiting Data Correction * ‘C’orrections made but not Approved * ‘R’eady for Resubmission * ‘S’ubmitted for Processing * ‘P’rocessed Successfully * ‘F’ailed During Resubmission * ‘D’eleted Never to be Processed * ‘X’fer to new Transaction Key |
|  | ORIGINAL\_  BUFFER | Blob | Contains the transaction buffer in a raw form as submitted to RTP for this processing instance. |
|  | CORRECTED\_  BUFFER | Blob | Contains the transaction buffer with the user edits that have been applied to the transaction to correct the errors. This field is blank until established with data corrections. |
|  | CREATED\_  DATETIME | Date/Time | Contains the date and time this processing instance was added to the database. |
|  | SUBMITTED\_  DATETIME | Date/Time | Contains the date and time the corrected transaction buffer was submitted back to RTP for reprocessing. |

Insertions into this table must follow a strict protocol to prevent the insertion of duplicate data and the perception of a lack of data integrity. Unless the update represents a new unique transaction key, new processing instances should only be added to this table if the current processing instance is in a status of “S”ubmitted for processing. An update request received with the prior instance in any other state should return to the caller that this is a duplicate transaction indicated by a last error code of EEXIST as defined in ERRNO.h. Successful processing of a transaction of this type should transition the status from “S”ubmitted to “P”rocessed to prevent duplicate processing.

**EXC\_ISSUE Data Definition**

This table contains a single row for each issue detected within the processing instance of a unique transaction key. Issues failures such as field edits failures, orphan records, and transaction audit failures. One entry should exist for each error detected. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | TRANSACTION\_  KEY | VarChar  (22-64) | Contains the transaction key that uniquely identifies the transaction submitted to RTP. This value must always correspond to a unique row in the EXC\_TRANSACTION table. |
|  | SEQUENCE\_  NUMBER | Numeric | Contains an automatically assigned number starting with zero that uniquely defines this version of the transaction. This value in combination with TRANSACTION\_KEY must always correspond to a unique value in the EXC\_PROCESS\_INSTANCE table. |
|  | ISSUE\_ID | Numeric | Contains an automatically assigned number starting with zero that uniquely identifies the issue for a given transaction key and sequence number combination. This number is used to display the issues in the order they were detected and to provide a unique key for updates. |
|  | STATUS | Char(1) | Contains a single character code that identifies the overall state of this processing instance. The following is a list of valid status codes:   * ‘U’ser Review Pending * ‘A’waiting Data Correction * ‘C’orrected but not Approved * ‘R’eady for Resubmission * ‘D’eleted Never to be Changed * ‘X’fer to new Transaction Key |
|  | RECORD\_  INSTANCE\_CODE | Char(2) | Contains the two-character record instance code used to determine which record had the failure in a transaction that contains multiple records such as DEPosit and Retail TransacTtion. |
|  | FIELD\_NAME | Char(12) | Contains the name of the field in which the issue was detected. |
|  | MESSAGE\_TEXT | VarChar  (192-255) | Contains the message text associated with the detected error as provided by the configuration database. |
|  | RGI\_OVERRIDE | Numeric | Contains a numeric ID that uniquely identifies the group responsible for correcting this specific issue if different from the group responsible for correcting the transaction overall. This value should be maintained even after the correction has been submitted for audit purposes. |

**EXC\_AUDIT Data Definition**

This table is used by the user interface to maintain audit information for the modifications made to transactions. A single row is inserted for each and every change made through the user interface to the transaction. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | TRANSACTION\_  KEY | VarChar  (22-64) | Contains the transaction key that uniquely identifies the transaction submitted to RTP. This key is variable length since it is build based on configuration information contained within the input configuration tables described in section 5.5.4.1. |
|  | AUDIT\_ SEQUENCE\_  NUMBER | Numeric | Contains an automatically assigned number starting with zero for each transaction key. This value is used to maintain the sequence in which audit events occurred. |
|  | SEQUENCE\_  NUMBER | Numeric | Contains an automatically assigned number starting with zero that uniquely defines this version of the transaction. This value in combination with TRANSACTION\_KEY must always correspond to a unique value in the EXC\_PROCESS\_INSTANCE table. |
|  | ISSUE\_ID | Numeric | Contains an automatically assigned number starting with zero that uniquely identifies the issue. This number corresponds to an entry the EXC\_ISSUE. |
|  | ACTION\_CODE | Char(2) | To be determined by the user interface. |
|  | OLD\_DATA | VarChar | Contains the original data before modification |
|  | NEW\_DATA | VarChar | Contains the result of the modification |
|  | RGI\_USER | Numeric | Contains a numeric ID that uniquely identifies the user that executed this event. |
|  | DATATIME\_  STAMP | Date/Time | Contains the date and time that the modification was submitted. |

### Reporting Engine

The reporting engine tables contain control and interface information for the *System Of Record* reporting components within the RTP System. The web interface handles initial updates to the primary table after which the reporting engine application maintains the on-going status and control information. Records are maintained within this system once a report is complete for no more than 24 hours. The *CDbReportEngine* class is used to access and maintain information within these tables.



Figure 13.Reporting Engine Table Structure

**REQ\_ACTIVELIST Data Definition**

This table contains a single row for each request that has been submitted for processing. Data is contained within this table for the life plus 24 hours of the request. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | INSTANCE\_ID | Numeric | Contains a unique numeric value that identifies this specific request. This value is assigned during insertion into the table. |
|  | REPORT\_CODE | Char(2) | Contains a two-character code that identifies which output has been requested. Valid codes include:   * EA – Exception Aging Report * TR – Ticket Retrieval Report * OS – Operational Statistics * FE – Batch Exception Fix * IC – Invalid Category Report * ED – Daily Exception Report * HS – Host Scan Report |
|  | REQUESTED\_ DATETIME | Date | Contains the date and time that the request was added to the table. Once set, this value should not change throughout the life of the request |
|  | REQUESTEDBY\_ID | Numeric | Contains the user ID number that uniquely identifies the RTP user record that submitted this request. |
|  | PRIORITY\_CODE | Char(1) | Contains a priority code indicating the importance of this request. Valid codes include:   * I – Immediate Action * H – High Priority * M – Medium Priority * L – Low Priority * D – Idle Time Processing |
|  | STATUS\_CODE | Char(1) | Contains a single character code that indicates the overall state of the system through the general process. This code is primarily designed for system use. Valid codes include:   * Q – Queued Waiting Processing * S – Process Start In Progress * I – Report Module Initialization * L – Loading Required Information * G – Generating Report * M – Mailing Report * C – Report Complete * H – Report on Hold * D – Report Request Deleted * F – Report Failed During Processing |
|  | STATUS\_MESSAGE | VarChar2 (64) | Contains a text message that should be used for any displays of status. This will provide user style detail related to the status. |
|  | HOLD\_FLAG | Char(1) | Contains a Y/N flag that should start out as “N”. A value of “Y” tells the reporting engine to stop any processing related to this request without removing it from the system. To restart processing, simply change the “Y” back to “N”. |
|  | CANCEL\_FLAG | Char(1) | Contains a Y/N flag that starts out “N”. A value of “Y” indicates that the reporting engine should stop any further processing and delete the associated request. Once set, the system will no longer honor this request. |
|  | PROCESS\_ID | Numeric | Contains the process ID that identifies the Unix process that is analyzing this reporting requirement. A value of ZERO indicates that no report specific processing was yet underway. |
|  | LASTUPDATE\_  DATETIME | Date | Contains the last date and time this record was updated. Once in a complete or delete state this date is used to age off information. |

**REQ\_CRITERIA Data Definition**

This table contains a single row for each criterion that should be used with a request that has been submitted for processing. Data is contained within this table as long as the request is maintained within the system. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | INSTANCE\_ID | Numeric | Contains the unique numeric value that identifies this specific request. This value must match the corresponding value in the REQ\_ACTIVELIST. |
|  | SEQUENCE\_  NUMBER | Numeric | Contains a unique number that sequences individual criteria within a single request that contains multiple criteria. This value should start with ZERO although this is not mandatory. |
|  | CRITERIA\_TYPE | Char(1) | This code indicates the type of criteria that is being used. Specific values that are supported depend on the report type and are published below this definition. |
|  | LOW\_VALUE | VarChar2  (200) | Contains a free form lower end value for the evaluation criteria. The format of this field depends on the report type and the criteria type. |
|  | HIGH\_VALUE | VarChar2  (200) | Contains a free form higher end value, if appropriate, for the evaluation criteria. If this is a singular match then this field is blank. The format of this field depends on the report type and the criteria type. |

**General Criteria Type**

* “S” indicates either detail or summary information is requested. If the parameter is missing then the report should assume that detail information is requested; otherwise, summary information only is provided. Low and high values have no meaning.
* “A”ll overrides the order type verification expression and includes all tickets in this report.

**Exception Aging**

* “L” indicates that this report is for a specific line of business set. The list of lines of business codes should be set in the LOW\_VALUE field where multiples are separated by “;”. HIGH\_VALUE is not used with this option.

**Ticket Retrieval**

* “T”ransaction Key is used to specify that we retrieve a single ticket back from the POS system. Entry of this parameter causes all other report specific search criteria to be ignored. The LOW\_VALUE contains the ticket number to be retrieved. Partial ticket numbers will retrieve all values that match the provided criteria – all request must, however, include at least the first 8 characters which constitute the Business Date requested.
* “D”ate range is used to specify the range of business dates from which the information should be retrieved. A single low value will indicate one day only; otherwise, the low/high combination will define a range of dates.
* “A”ll transactions are included in the report regardless of the guidelines set forth in the “TR\_ORDER\_OK” parameter in the RTPRENG.ini configuration file.
* “C”heck ID restriction allows you to enter a range for the check ID to retrieve from the system. Sales will be limited to those who match this value for the “CHECK\_ID” field.
* “F”ormat is used to specify the output format for the reports. A summary report provides information at the ticket level – valid format values if the summary flag is set are as follows:
  + “I”mpact Value listing
  + Detail “V”ariance Report

**Batch Exception Fix**

* “I”d contains the event ID used for the report being pulled. This is the key used to access the detail information in the EXC\_BATCH\_FIX table.

**Exception Daily Report**

* “D” contains the working date to use as the current date of the file. If not specified then the system generates the report for today.
* “G” allows you to specify a target audit group. Failure to specify an audit group will cause the system to generate an overall report.
* “H” allows you to specify whether the historical information is outstanding exceptions or the new exceptions for the last thirty days.

**Host Scan Report**

* “O” contains the operational code used for the report. This is a required parameter.
* “I” contains the event ID used for report generation. This is a required parameter.
* “E” contains the exit code from the process that triggered this report. This optional parameter if provided adds results to the end of the report. ZERO indicates a successful completion.

**REQ\_RESPONSE\_LIST Data Definition**

This table contains a single row for each request that has been submitted for processing. Data is contained within this table for the life plus 24 hours of the request. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | INSTANCE\_ID | Numeric | Contains the unique numeric value that identifies this specific request. This value must match the corresponding value in the REQ\_ACTIVELIST. |
|  | SEQUENCE\_  NUMBER | Numeric | Contains a unique number that sequences individual criteria within a single request that contains multiple criteria. This value should start with ZERO although this is not mandatory. |
|  | ID\_NUMBER | Numeric | Contains the user identification number within the RTP system to who this report should be mailed. This value must exist within the USR\_PROFILE table. |

### Configuration

As described throughout this document, RTP is a data driven system that is designed to limit the amount of code changes required in the future and improve the overall re-usability of the system. The following sections defined the various configuration tables loaded into memory during initialization of RTP and used to directly control the operation of the system.

#### Input Format Definitions

The input transaction format tables control the system’s analysis of the input records during transaction processing. By transaction records, fields, edits, and message content are described completely within these tables. Figure 14 shows the basic record structure along with the high-level organization. In addition, fields whose names contain either “MESSAGEID” or “EDITCODE” are also linked to the ITF\_MESSAGE\_DEF and ITF\_EDITS tables respectively. In the case of “MESSAGEID”, it refers to a message used for reporting if the related verification fails. In the case of “EDITCODE”, it defines a series of edits that should be associated with the fields. The *CCfgInFormVer* class is used to access information in the ITF\_VERSION table while the *CCfgInForm* class accesses all other “ITF\_” tables.



Figure 14.Input Transaction Format Table Structure

**ITF\_VERSION Data Definition**

This table contains a single row for each input file version that is supported by the RTPBATCH executable. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | VERSION\_ID | Char(2) | This two-character field is the version identification number that uniquely identifies this input format. Version IDs begin with “00” and proceeds through “ZZ”. |
|  | LOWER\_LIMIT | Char(2) | This two-character field identifies a lower version identification number that can be processed using this format. This capability is used to update information such as message contents without always requiring the selling devices upgrade their corresponding software. |
|  | EFFECTIVE\_DATE | Date | This contains the date on which the format should be made available to any selling device. |
|  | DISABLE\_DATE | Date | This contains the date after which the format is no longer valid. Any selling device submitting transactions in this format after the specified date will be rejected without being processed. |
|  | PRELOAD | Char(1) | The Y/N flag indicates if the input format should be loaded during daemon initialization. A “Y” in this field causes it to be preloaded. A “N” means the format is only loaded if a transaction is received using it. |
|  | CHRONO\_KEY | Char(12) | This twelve-character field contains the “field name” for the header field that identifies the chronological key used for each transaction. |
|  | LOCATION\_KEY | Char(12) | This twelve-character field contains the “field name” for the header field that identifies the location key used for each transaction. |
|  | DEVICE\_KEY | Char(12) | This twelve-character field contains the “field name” for the header field that identifies the device key used for each transaction. |
|  | SEQUENCE\_KEY | Char(12) | This twelve-character field contains the “field name” for the header field that identifies the sequence number used in the key for each transaction. |

**ITF\_TRANSACTION Data Definition**

This table contains a single row for each transaction code supported in the corresponding input file version. A foreign key link is established between this and the ITF\_VERSION table. In addition, the VALID\_MODE\_MESSAGEID field links this table to a unique entry in the ITF\_MESSAGE\_DEF table. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | VERSION\_ID | Char(2) | This two-character field is the version identification number that uniquely identifies this input format. Version IDs begin with “00” and proceeds through “ZZ”. |
|  | TRANSACTION\_  CODE | Char(3) | This three-character field uniquely identifies the transaction. This code is contained within the *Base Sequence* of corresponding transaction and is used as the basis for interpreting all the data within the transaction. |
|  | VALID\_MODE | Char(1) | This single character field identifies the mode under which this transaction is valid. Mode operation is defined in section 2.2.2.2 of this document. |
|  | VALID\_MODE\_  MESSAGEID | Numeric | This numeric value combined with VERSION\_ID uniquely identifies the message text within the ITF\_MESSAGE\_DEF table that should be used if the appropriate *Transaction Mode* does not meet or exceed the VALID\_MODE defined for this transaction. A value of ZERO is only valid for Mode “A” transactions since they are always valid. |
|  | RESULTING\_  MODE | Char(1) | This single character field contains the mode that should be assigned to this location once this transaction has been successfully processed. A space (0x32) within this field indicates that no change is appropriate. |
|  | SCREEN\_TITLE | VarChar | This contains a human readable title that should be used when displaying this transaction. |
|  | VALUE\_  EXPRESSION | VarChar | This contains a standard RTP mathematical expression that can be used to determine the overall value of the transaction associated with this code. The default value assumed if not present is “0”. |
|  | USEOK\_  EXPRESSION | VarChar | This contains a standard RTP mathematical expression that when evaluated to true or false indicates if this transaction is considered of value. This function is normally used to eliminate fully voided transactions and training transactions from reporting totals. The default value assumed if not present is “1” which is considered true. |

**ITF\_RECORD Data Definition**

This table contains a single row for each record type code supported in the corresponding transaction code and input file version combination. A foreign key link is established between this and the ITF\_VERSION and ITF\_TRANSACTION tables. In addition, the REQUIRED\_MESSAGEID and SINGLE\_INSTANCE\_MESSAGEID fields link this table to a unique entry in the ITF\_MESSAGE\_DEF table. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | VERSION\_ID | Char(2) | This two-character field is the version identification number that uniquely identifies this input format. Version IDs begin with “00” and proceeds through “ZZ”. |
|  | TRANSACTION\_  CODE | Char(3) | This three-character field uniquely identifies the transaction. This code is contained within the *Base Sequence* of corresponding transaction and is used as the basis for interpreting all the data within the transaction. |
|  | RECORD\_TYPE | Char(2) | This two-character field uniquely identifies the record within the transaction buffer. A type of “—“ always identifies the first or header record within the transaction. All other records must contain a unique set of values between 0-9 and A-Z. |
|  | REQUIRED\_  MESSAGEID | Numeric | This numeric value combined with VERSION\_ID uniquely identifies the message text within the ITF\_MESSAGE\_DEF table that should be used if the corresponding record doesn’t exist. A value of ZERO indicates that the record type is not required within this transaction. |
|  | SINGLE\_  INSTANCE\_  MESSAGEID | Numeric | This numeric value combined with VERSION\_ID uniquely identifies the message text within the ITF\_MESSAGE\_DEF table that should be used if multiple instances of this record type exist within the transaction. A value of ZERO indicates that multiple instances of this record type is acceptable for this type of transaction. |
|  | SCREEN\_TITLE | VarChar | This contains a human readable title that should be used when displaying this record type. |
|  | DISPLAY\_  SECURITY | Char | This flag controls when this record is available in exception management interface. Valid values include:   * A – Always show the record * O – On error in the record * N – Never show the record |

**ITF\_FIELD Data Definition**

This table contains a single row for each field contained within a record type code, transaction code, and input file version combination. A foreign key link is established between this and the ITF\_VERSION, ITF\_TRANSACTION, and ITF\_RECORD tables. In addition, the TYPE\_MESSAGEID field links this table to a unique entry in the ITF\_MESSAGE\_DEF table. The COMMON\_EDITCODE and SPECIAL\_EDITCODE link this entry to one or more rows in the ITF\_EDIT table defining the edits that should be processed against the represented input field. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | VERSION\_ID | Char(2) | This two-character field is the version identification number that uniquely identifies this input format. Version IDs begin with “00” and proceeds through “ZZ”. |
|  | TRANSACTION\_  CODE | Char(3) | This three-character field uniquely identifies the transaction. This code is contained within the *Base Sequence* of corresponding transaction and is used as the basis for interpreting all the data within the transaction. |
|  | RECORD\_TYPE | Char(2) | This two-character field uniquely identifies the record within the transaction buffer. A type of “—“ always identifies the first or header record within the transaction. All other records must contain a unique set of values between 0-9 and A-Z. |
|  | FIELD\_NAME | Char(12) | This twelve-character field identifies the field name that will be used during the translation. This label must be unique within a transaction-record combination. It is not case sensitive and must only contain characters and numbers. |
|  | OFFSET | Numeric | This is the number of bytes from the start of the transaction to the beginning of this field. |
|  | LENGTH | Numeric | This is the number of bytes represented by this field. |
|  | TYPE | Char(1) | This flag indicates the type of information contained within this field. A value of “0” indicates the data is an integer[[12]](#footnote-12) numeric[[13]](#footnote-13) with or without a proceeding sign, “X” indicates the data is zoned decimal, “A” is alphanumeric (spaces OK), a value of “Z” is an alphanumeric field that RTP should right justify and zero fill, a value of “C” is alpha characters only, and a value of “E” allows every ASCII character. All of the above formats allow the use of 0x20 to fill out the right side of the field. A character value from “1” to “9” indicates a fixed decimal value where the numeric equivalent of the character indicates the number of decimal places past the decimal point a sign with this type of field is acceptable (e.g., “2” should be used for all dollar fields to indicate the last two numbers are the cents). A value of “D” indicates that it is a date in the form “MMDDYYYY” that should be translated to “YYYYMMDD”. Other date formats or dates that should not be translated should use a setting of “0”. |
|  | TYPE\_  MESSAGEID | Numeric | This numeric value combined with VERSION\_ID uniquely identifies the message text within the ITF\_MESSAGE\_DEF table that should be used if data contained within the field does not match the type declared for this field. A value of ZERO is never valid. |
|  | REQUIRED\_  MESSAGEID | Numeric | This numeric value combined with VERSION\_ID uniquely identifies the message text within the ITF\_MESSAGE\_DEF table that should be used if the corresponding field doesn’t exist. A value of ZERO indicates that the field is not required within this record. Records filled with spaces 0x20 are considered to exist if the field type is set to “A”. |
|  | COMMON\_  EDITCODE | Numeric | This numeric value identifies one or more records within the ITF\_EDIT table that should be used in verifying this data. “Common” edits are edits used by many fields. These edits should verify things link ranges or patterns such as date. A value of ZERO indicates that there are no common edits. |
|  | SPECIAL\_  EDITCODE | Numeric | This numeric value identifies one or more records within the ITF\_EDIT table that should be used in verifying this data. “Special” edits are edits that are unique to this fields or a small group of fields. These edits should verify specific information such as valid values. A value of ZERO indicates that there are no special edits. |
|  | SCREEN\_TITLE | VarChar | This contains a human readable title that should be used when displaying this record type. |
|  | DATA\_STORAGE\_MODE | Char | This field controls how information is stored within the NVTR. It is designed to minimize potential exposure of the information. Valid values include:   * S – Standard Machine Form * I – Ignore and Replace with Spaces * C – Encode ASCII Strings * E – Encrypted Strings * B – Both Encode and Encrypt Strings |

**ITF\_EDIT Data Definition**

This table contains a single row for each edit that should be applied to one or more fields within the input transactions. Edits can be used multiple times within an input format version; however, they cannot be referenced across versions. A foreign key link is established between this and the ITF\_VERSION table. In addition, the MESSAGEID field links this table to a unique entry in the ITF\_MESSAGE\_DEF table. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | VERSION\_ID | Char(2) | This two-character field is the version identification number that uniquely identifies this input format. Version IDs begin with “00” and proceeds through “ZZ”. |
|  | EDITCODE | Numeric | This number uniquely identifies the set of edits represented. It is used to link the edits to one or more field definitions contained within the database. Assignment of this number is purely random. |
|  | SEQUENCE\_  NUMBER | Numeric | This contains a sequence number beginning with ZERO. This field is only used to order the edits. |
|  | EDIT\_TYPE | Char(1) | This field contains a flag that identifies the type of edit that should be executed against the field. Valid types include:   * P – Pattern Match where EDIT\_DATA contains the pattern to be matched[[14]](#footnote-14). * V – Dollar Value Check where EDIT\_DATA contains a series of switches as follows: “+” indicates a sign is OK, “0” indicates zero filled (no spaces), and a “.” means it must contain the decimal values including the first digit to the left of the decimal as ZERO filled characters. Switches can appear in any order and do not need to be separated by any special characters. * N – Numeric Range where EDIT\_DATA contains a low and high value separated by a semi-colon. A “#” at the start of the field indicates all spaces without a numeric value is also considered valid. * C – Character Range where each character in the string is represented by a set of low and high characters contained sequential in the field. * L – Left justified field where the EDIT\_DATA contains the pad character. * R – Right justified field where the EDIT\_DATA contains the pad character. * S – Verify field against a “set of valid values” where the EDIT\_DATA field contains a series of semi-colon delimited potential responses. * I – In Set where the EDIT\_DATA field contains a set of characters that are considered the valid characters for the input field. * E – Exclude where the EDIT\_DATA field contains a set of characters that cannot be contained within the input field. * M – Must exist can only be used for fields that reference another record type. If this edit is set the EDIT\_DATA field contains information[[15]](#footnote-15) about the record type that the sequence number in this field should represent. * D – Date check where the EDIT\_DATA contains the date format. “Y” indicates year, “M” indicates month, “D” indicates day, “H” indicates hours, “I” indicates minutes, and “S” indicates seconds. A leading “~” indicates that the field is optional and considered passed if all blank. * F – Must be filled indicates that the field must contain non-blank data. EDIT\_DATA is not used for the edit type. |
|  | EDIT\_DATA | Char(127) | This field contains values appropriate to the edit type declared in the prior field. |
|  | MESSAGEID | Numeric | This numeric value combined with VERSION\_ID uniquely identifies the message text within the ITF\_MESSAGE\_DEF table that should be used if data contained within the field does not pass the edit. A value of ZERO is never valid. |

**ITF\_MESSAGE\_DEF Data Definition**

This table contains a single row for each output message that can be used to represent the failure of an input edit. Messages can be used multiple times and support a minimal level of variable substitution capabilities. A foreign key link is established between this and the ITF\_VERSION table. The MESSAGEID field links each row to its corresponding edit. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | VERSION\_ID | Char(2) | This two-character field is the version identification number that uniquely identifies this input format. Version IDs begin with “00” and proceeds through “ZZ”. |
|  | MESSAGEID | Numeric | This numeric value combined with VERSION\_ID uniquely identifies the message within the ITF\_MESSAGE\_DEF table. This combination is used to link the edits to the message. A value of ZERO is never valid. |
|  | TEXT | VarChar | This field contains the actual readable text that should be used when reporting the associated edit failure. The system will make the following substitutions as required:   * %V = Input Version Code * %T = Transaction Code * %R = Record Type Code * %S = Record Sequence Code * %F = Field Name |

**ITF\_AUDIT Data Definition**

This table contains a single row for each evaluation that should be checked against a given transaction in order to verify its contents. Audits are only run against a transaction if all the field level edits are completed successfully. The system then moves to each audit first evaluating the CONDITION string to see if this verification applies then using the EQUATION to validate the audit. If the CONDITION is true but the EQUATION fails then the system reports the message ID as an exception preventing further processing of the associated transaction. Audits are different then edits in that they can compare information between fields and/or records where as edits are for a specific field only. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | VERSION\_ID | Char(2) | This two-character field is the version identification number that uniquely identifies this input format. Version IDs begin with “00” and proceeds through “ZZ”. |
|  | TRANSACTION\_  CODE | Char(3) | This three-character field uniquely identifies the transaction. This code is contained within the *Base Sequence* of corresponding transaction and is used as the basis for interpreting all the data within the transaction. |
|  | RECORD\_TYPE | Char(2) | This two-character field identifies the record type within the transaction buffer to which this audit should be applied. The audit is applied to each record that has the corresponding record type until all appropriate records have been reviewed. A type of “—“ always identifies the first or header record which is then singularly applied across the entire transaction. |
|  | CONDITION | VarChar  (192) | This contains a numeric mathematical expression as described in section 5.5.6 that must evaluate to true for this audit to be applied to the associated transaction. |
|  | EQUATION | VarChar  (192) | This contains a numeric mathematical expression as described in section 5.5.6 that causes an exception based on the MESSAGEID text if it is evaluated true. This is considered the audit. |
|  | MESSAGEID | Numeric | This numeric value combined with VERSION\_ID uniquely identifies the message text within the ITF\_MESSAGE\_DEF table that should be used if the CONDITION is true but the EQUATION fails during evaluation. A value of ZERO is not valid. |
|  | Exception State Code | Char(1) | This optional value is used to set the overall state code applied to the exception generated by this message. |

#### Host/Location Definitions

The host/location definition tables provide the system information on the operation of the hosts, the physical locations, and any corresponding inter-relationship between these two entities. Information is retrieved from these tables during the initialization of the daemon and is maintained throughout system operation. Figure 15 shows the basic record structure along with the high-level organization. The *CCfgHost* and *CCfgLocation* classes are used to access the corresponding host and location data, respectively.

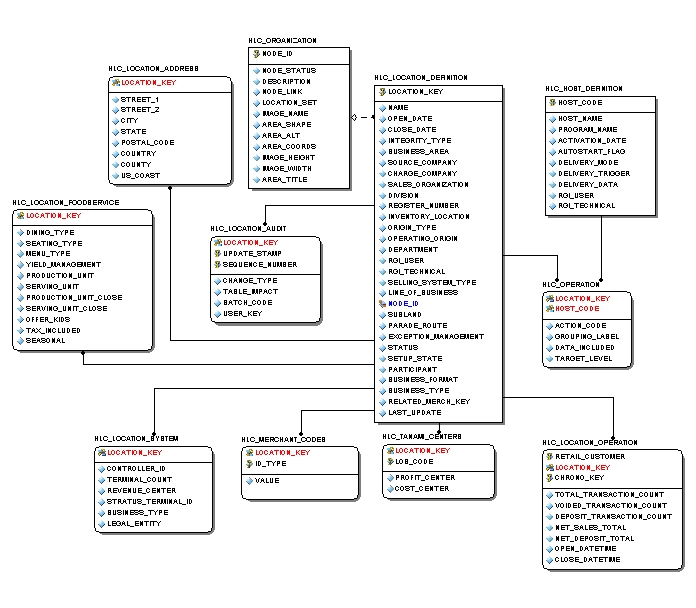


Figure 15.Host/Location Definition Table Structure

**HLC\_HOST Data Definition**

This table contains a single row for each host that can receive information from RTP through a RTPDTRAN instance. Retrieved during daemon and RTPDTRAN initialization, this information is used to initializes these background processes and, subsequently, control how information is passed to the host system. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | HOST\_CODE | Char(2) | Contains the two-character host identification code that uniquely identifies the host information. This code is used by RTPDTRAN to identify the appropriate configuration information. |
|  | HOST\_NAME | Char(24) | Contains a text string that provides a human readable name for reporting status and operation. |
|  | PROGRAM\_NAME | Char(64) | Contains the program name that is executed to provide host translation. The path for this program is contained within the RTPSUPPORT.ini file under the parameter titled TRANSLATOR\_DIRECTORY. |
|  | ACTIVATION\_  DATE | Date | Contains a date in the form YYYYMMDD that indicates the date on which this entry becomes active. No transactions with a chronological key earlier than this date are processed. |
|  | AUTOSTART\_  FLAG | Char(1) | Contains a Y/N flag indicating if this translator should be started by the daemon during initialization. A value of “Y” indicates that it should automatically be started. |
|  | DELIVERY\_MODE | Char(2) | Contains a flag in the first character of this field that indicates the type of delivery used for this host. Support values include:   * “F” – Fixed Field/Record Lengths * “H” – Fixed Field/Record Lengths with support for hidden fields. * “N” – Fixed Field/Record Lengths without end of record/line markers. WARNING: This mode only works for ASCII files. * “D” – Character[[16]](#footnote-16) Delimited Data Fields * “X” – XML (Phase III) |
|  | DELIVERY\_  TRIGGER | Char(1) | Contains a flag that controls the trigger that initiates a transfer of information to this host. The valid values are as follows:   * “F”ixed Time of Day – Data contains a 24 hours time in the format HHMM * “I”nterval through the Day – Data contains two values in the form “XX;YY” where XX is the number of minutes past midnight the interval starts and YY is the minutes between transmissions. * “W”eekly at a fixed day of week and time. Data contains DHHMM where D=day of week 0=Sun..6=Sat and HHMM is the 24 hours clock time. * “T”ransaction Trigger[[17]](#footnote-17) – Data contains the three-character transaction code that should trigger transmission of data. * “L”oop Back – Setting causes information to be loaded into a local database table. The delivery data field contains the name of the table into which the data is inserted. * “R”eal-time – Data is not used in this case. |
|  | DELIVERY\_DATA | Char(32) | Contains trigger specific information as defined in the description of DELIVERY\_TRIGGER. |
|  | RGI\_USER | Numeric | Contains a group ID number that uniquely identifies the business users responsible for failures of this host. |
|  | RGI\_TECHNICAL | Numeric | Contains a group ID number that uniquely identifies the technical team responsible for failures of this host. |

**HLC\_LOCATION\_DEFINITION Data Definition**

This table contains a single row for each location that submits sales information to RTP. Retrieved during daemon and RTPDTRAN initialization, this information is used to verify and control information flow within the system. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | LOCATION\_KEY | VarChar  (5-16) | Contains the location component of the transaction key. This value uniquely identifies the physical selling location. |
|  | NAME | Char(32) | Contains a human readable string that is used to describe this location in user displays and reports. |
|  | OPEN\_DATE | Date | Contains the date on which this location was opened for business. This date is only used to monitor on-going business – sales received dated prior to this date WILL be accepted. |
|  | CLOSE\_DATE | Date | Contains the date on which this location was closed for further business. This date is only used to control monitoring of on-going business. Transactions received after this date WILL be processed. |
|  | INTEGRITY\_TYPE | Char(1) | Contains a flag that indicates what type of integrity checks are enforced for this shop/location number. Valid values include:   * N – No checks made all records passed * M – Transaction Node only enforced * S – Sequence Number only enforced * B – Both Transaction and Sequence |
|  | BUSINESS\_AREA | Char(4) | Contains the Tanami Business Area code. |
|  | SOURCE\_  COMPANY | Char(6) | Contains the Tanami source company code. |
|  | CHARGE\_  COMPANY | Char(6) | Contains the Tanami charge company code. |
|  | SALES\_  ORGANIZATION | Char(4) | Contains the Tanami Sales Organization |
|  | DIVISION | Char(2) | Contains the Tanami Division information |
|  | REGISTER\_  NUMBER | Char(5) | Contains the Retail Sales register number for posting revenue and deposits. |
|  | INVENTORY\_  LOCATION | Char(5) | Contains the Merchandise Inventory location used to report inventory changes to Merlin. |
|  | ORIGIN\_TYPE | Char(1) | Contains a single character value indicating the type of origin this represents. Valid values include:   * P – Park * R – Resort * M – Miscellaneous |
|  | OPERATING\_  ORIGIN | Char(3) | Contains the operating origin |
|  | DEPARTMENT | Char(3) | Contains the department |
|  | RGI\_USER | Numeric | Contains a group ID number that uniquely identifies the business users responsible for failures of this location. |
|  | RGI\_TECHNICAL | Numeric | Contains a group ID number that uniquely identifies the technical team responsible for failures of this location selling device. |
|  | SELLING\_ SYSTEM\_TYPE | Char(2) | Contains a two-character code that uniquely identifies the predominate type of selling system being used at the identified location. Valid codes are defined in the *Batch File Specification* document for the field “Source System” in the LBO and LBC transaction. |
|  | LINE\_OF\_  BUSINESS | Char(2) | Contains a two-character code indicating the predominate line of business represented by this location. Valid values include:   * “M ”erchandise * “F ” for Food and Beverage * “L ” for floral. * “FM” Food location with Merchandise * “MF” Merchandise location with food * “R ” Recreation * “A ” Attraction * “RX” Recreation with F&B and Merch. |
|  | NODE\_ID | Numeric | Contains the Node Identification number under which this location should be reported as defined in the HLC\_ORGANIZATION table. |
|  | SUBLAND | Numeric |  |
|  | PARADE\_ROUTE | Char(1) | Contains a Y/N flag that indicates if the location is physically on a parade route. |
|  | EXCEPTION\_ MANAGEMENT | Char(1) | Contains a Y/N flag that indicates if exception management is enabled for this location. A value of “Y” causes RTP to enforce data validation rules. A value of “N” causes it to ignore data issues outside of the transaction key. |
|  | STATUS | Char(1) | Contains the latest update until it is transferred to the downstream system (e.g., Merlin). Valid values include:   * A – Added to the Database * C – Closed to be deleted * U – Updated recently pending transfer * O – Open to further modification. All changes have been sent to Merlin. |
|  | SETUP\_STATE | Char(1) |  |
|  | PARTICIPANT | Char(1) | Contains a Y/N flag that indicates if this location represents a participant. A value of ‘Y’ indicates that a participant runs this location. |
|  | BUSINESS\_ FORMAT | Char(1) |  |
|  | BUSINESS\_TYPE | Char(1) |  |
|  | RELATED\_ MERCH\_KEY | Varchar2 (16) | Contains a LOCATION\_KEY for the location that the merchandise inventory is legally attached to. This field is left blank in non-consolidated locations or where the location is the primary merchandise inventory location. |
|  | LAST\_UPDATE | Date | Contains the date and time which this location information was last updated. |

**HLC\_TANAMI\_CENTERS Data Definition**

This table contains a single row for each location-line of business combination. It is used to set Tanami revenue values when reporting sales to SAP. It is loaded into the master shared memory segment during initialization of RTPINIT. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | LOCATION\_KEY | VarChar  (5-16) | Contains the location component of the transaction key. This value uniquely identifies the physical selling location. |
|  | LOB\_CODE | Char(2) | Contains a code that uniquely identifies the line of business associated with this sale. The following is a list of valid values   * “M”erchandise * “F” for Food and Beverage * “R”ecreation * “A”ttractions * “S”upplemental * “P”articipants * “O”ther |
|  | PROFIT\_CENTER | Char(10) | Contains the 10-character Tanami profit center that should receive revenue for sales of product for this line of business that occur at the identified physical selling location. |
|  | COST\_CENTER | Char(10) | Contains the 10-character Tanami cost center that should be burdened with the cost of sales for product from this line of business that occur at the identified physical selling location. |

**HLC\_OPERATION Data Definition**

This table contains a single row for each host-location combination. It is used to control whether or not the related host receives information from this selling location. It is loaded into memory during initialization of RTPDTRAN. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | LOCATION\_KEY | VarChar  (5-16) | Contains the location component of the transaction key. This value uniquely identifies the physical selling location. |
|  | HOST\_CODE | Char(2) | Contains the two-character host identification code that uniquely identifies the host information. This code is used by RTPDTRAN to identify the appropriate configuration information. |
|  | ACTION\_CODE | Char(1) | Contains a code that controls whether any records received from the associated selling location are considered for transmission to the related host. A value of “P” indicates that they should be processed while a value of “I” indicates they should be ignored. |
|  | GROUPING\_  LABEL | Char(8) | Contains an eight-character string that is associated with location when it is submitted to the host. Locations with a similar grouping label are transmitted to the associated host as a set. |
|  | DATA\_INCLUDED | Char(1) | Contains a control field used in determination of whether host bundling can only proceed if this location is included in the bundling data. This DOES not ensure that the location has met minimum data requirements nor that a specific business date is included. Valid values include:   * A = Always Required * 1 = Percent of Total Group 1 * 2 = Percent of Total Group 2 * 3 = Percent of Total Group 3 * 4 = Percent of Total Group 4 * N or Space = Ignored for this Check |
|  | TARGET\_LEVEL | Numeric | Contains the number of transactions expected for the location to be considered “included” for the purpose of the “DATA\_INCLUDED” functionality. A value of ZERO means a single transaction will cause the system to consider the location included. |

**HLC\_LOCATION\_OPERATION Data Definition**

This table contains a single row for each location-business date combination. This table is maintained through the automated load process and is used for monitoring purposes only. Information in this table is expired after 6 months. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | LOCATION\_KEY | VarChar  (5-16) | Contains the location component of the transaction key. This value uniquely identifies the physical selling location. |
|  | CHRONO\_KEY | Date | Contains the chronological component of the transaction key which declares the related business date. |
|  | FINAL\_  TRANSACTION\_  COUNT | Numeric | Contains the number of transaction processed for the selling devices at this location during the given business date. This is a control value only. |
|  | NET\_SALES\_  TOTALS | Numeric | Contains a dollar amount without the decimal indicating the net total of sales processed for the given business date. |
|  | NET\_DEPOSIT\_  TOTALS | Numeric | Contains a dollar amount without the decimal totaling all deposits. |
|  | START\_  DATETIME | Date/Time | Contains the date and time that the first transaction is received for this location-business date combination. |
|  | CLOSE\_  DATETIME | Date/Time | Contains the date and time that the store close is received from this location for the given business date. |

**HLC\_MERCHANT\_CODES Data Definition**

This table is designed to support GSA and Compris TLOG files which do not provide merchant ID information in their store open records. All other POS systems provide these values in the Store Open records. For store using this table, one row exists for each combination of ID\_TYPE and location key. This information is loaded from the CcfgLocation class and is loaded directly from the database as needed. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | LOCATION\_KEY | VarChar  (5-16) | Contains the location component of the transaction key. This value uniquely identifies the physical selling location. |
|  | ID\_TYPE | Char(1) | Contains a single character code that indicates the type of merchant code contained within this record. Valid values include:  A – American Express  B – Bank Card (VISA/Mastercard)  D – Discover Card  I – Disney Credit Card  N – Diners Card  J – JCB Card  C – Co-branded Redemption Cards |
|  | VALUE | Char(15) | Contains the merchant ID associated with the ID\_TYPE. |

**HLC\_ORGANIZATION Data Definition**

This table contains one or more rows that identify the reporting hierarchy for a set of locations. One row exists for each unique level of hierarchy and the corresponding choices for leaving the current hierarchy level. A value of “Y” in the location set indicates that individual locations are directly associated with the specific hierarchy which typically complete a leg of the tree. This information is used by the User Interface only and is loaded on demand. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | NODE\_ID | Numeric | Contains a numeric value that uniquely identifies this row. In addition, this value is used to link this level with other levels in the tree. |
|  | NODE\_STATUS | Char(1) | Contains a flag that indicates the status of the node. Valid values include:   * “A”ctive * “I”nactive * “D”eleted Awaiting Removal |
|  | DESCRIPTION | Char(32) | Contains a human-readable text string describing this tree-level. |
|  | NODE\_LINK | Numeric | Contains a numeric value that identifies the parent of this item. A value of –1 indicates this is the top level entry in the tree. |
|  | LOCATION\_SET | Char(1) | Contains a Y/N flag indicating if there are locations associated with this level of the organization. A value of “Y” indicates that one or more rows in the HLC\_LOCATION\_DEFINITION table have this row’s Node\_ID in the corresponding Node\_ID field. |
|  | IMAGE\_NAME | VarChar2 (96) |  |
|  | AREA\_SHAPE | VarChar2 (10) |  |
|  | AREA\_ALT | VarChar2 (100) |  |
|  | AREA\_COORDS | VarChar2 (255) |  |
|  | IMAGE\_HEIGHT | Numeric |  |
|  | IMAGE\_WIDTH | Numeric |  |
|  | AREA\_TITLE | VarChar2 (35) |  |

#### System Internal Use Definitions

The System Internal Use tables provide global internal control information designed to simplify customization of the user interface. Figure 17 shows the basic record structure along with the high-level organization.



Figure 16.System Internal Use Table Structure

**SIU\_CODE\_DEFINITION Data Definition**

This table is used to define the type of codes defined in the SIU\_CODES table. It is used during the maintenance of the codes tables to clarify data content for the user. There is a single row in this table for each *Code\_Group.* Each *Code Group* identifies the purpose of the list (e.g., filling a module list box). Each *Code* identifies all of the possible values. This table is not used within the Unix application. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | CODE\_GROUP | Char(5) |  |
|  | DESCRIPTION | VarChar2 (48) |  |

**SIU\_CODES Data Definition**

This table is used as a code table for validation of input and loading of combo and/or list boxes in the user interface. Each *Code\_Group* identifies the purpose of the list (e.g., filling a module list box). Each *Code* identifies all of the possible values. This table is not used within the Unix application. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | CODE\_GROUP | Char(5) |  |
|  | CODE | VarChar2 (24) |  |
|  | SHORT\_DESC | Char(15) |  |
|  | LONG\_DESC | Char(40) |  |
|  | STATUS | Char(1) |  |

**SIU\_SOURCE\_SYSTEM Data Definition**

This table is used to provide sequential batch ID numbers for programs such as RTPWRAP, RTPGSA, and RTPEDS that produce RTP Compliant batch files. There is a single entry for each source system as defined in section 5.5.4.2 of this document. There is one and only one row for each source system. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | SOURCE\_  SYSTEM\_KEY | Char(2) | Contains the two-character code that identifies the corresponding source systems. Valid values are defined in the *Batch File Specification*. |
|  | SEQUENCE\_  NUMBER | Numeric | Contains the next number to be used when creating a new RTP Compliant batch file within the LBO and LBC transactions. |

#### Output Definitions

The output definition tables provide detailed descriptions on how the data driven translators should generate host system input records. These descriptions start with defining under what conditions records are generated, how they are grouped, and what the field level data is in the output records. Information is retrieved from these tables during the initialization of each of the DTRAN instances based on the provided host codes and is maintained throughout system operation. Figure 17 shows the basic record structure along with the high-level organization. The *CCfgOut* class is used to access the corresponding data.



Figure 17.Output Format Definition Table Structure

**OTF\_VERSION Data Definition**

This table contains a single row for each unique version of the output format that is defined within the system. During RTPDTRAN initialization, the system looks for the version ID with the latest effective date and uses that to initialize the translator. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | OUT\_VERSION\_ID | Numeric | Contains a numeric value that uniquely identifies all the record and field definitions associated with output formats defined at a specific implementation. |
|  | EFFECTIVE\_DATE | Date | Contains the date on which this output version becomes active. This date is used to allow the early entry of new format that should be held to a specific activation date. |

**OTF\_TRANSACTION Data Definition**

This table contains a single row for each host-transaction combination by version number that may generate one or more output records for the related host. This table provides a control code that indicates whether this host receives any records for the given transaction. During RTPDTRAN initialization, the system loads this information into memory for use by the translator. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | OUT\_VERSION\_ID | Numeric | Contains a numeric value that uniquely identifies all the record and field definitions associated with output formats defined at a specific implementation. |
|  | HOST\_CODE | Char(2) | Contains the two-character host identification code that uniquely identifies the host information. This code is used by RTPDTRAN to identify the appropriate configuration information. |
|  | TRANSACTION\_  CODE | Char(3) | This three-character field uniquely identifies the transaction. This code corresponds to values contained within the *Base Sequence* of affected transactions. |
|  | ACTION\_CODE | Char(1) | Contains a code that controls whether any records are generated for the associated transaction. A value of “P” indicates that they should be created while a value of “I” indicates that no records are generated. |
|  | FORCED\_SEND | VarChar2  (192) | This contains a numeric mathematical expression as described in section 5.5.6 that if it evaluates to true will cause the system to ALWAYS send this information in the next bundle regardless of the conditions / restrictions put on bundling. |
|  | TOTAL\_  CALCULATION | Char(1) | Contains a “Y/N” field that enables whether a pre-build transaction size total is calculated for this type of transaction. Enabling this calculate will impact overall system performance so it should only be used when needed. If this parameter is missing then the system assumes that it is disabled. |

**OTF\_RECORD Data Definition**

This table contains a single row for each host-transaction-record type combination by version number that generates one or more output records. This table identifies an output record code that should be used to generate output records if the specified record type exists and meets the criteria specified in the conditional definitions. The grouping indicator is used to connect related instances of this record type. During RTPDTRAN initialization, the system loads this information into memory for use by the translator. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | OUT\_VERSION\_ID | Numeric | Contains a numeric value that uniquely identifies all the record and field definitions associated with output formats defined at a specific implementation. |
|  | HOST\_CODE | Char(2) | Contains the two-character host identification code that uniquely identifies the host information. This code is used by RTPDTRAN to identify the appropriate configuration information. |
|  | TRANSACTION\_  CODE | Char(3) | This three-character field uniquely identifies the transaction. This code corresponds to values contained within the *Base Sequence* of affected transactions. |
|  | RECORD\_TYPE | Char(2) | This two-character field uniquely identifies the record within the transaction buffer. A type of “—“ always identifies the first or header record within the transaction. All other records must contain a unique set of values between 0-9 and A-Z. |
|  | SEQUENCE\_  NUMBER | Numeric | Contains a sequence number starting with ZERO that makes for a unique primary key. |
|  | GROUP\_ID | Numeric | Contains a code that identifies one or more grouping records in the OTF\_GROUPING table that controls which record instances are reviewed as a set rather than individuals. Multiple records grouped together produce only a single output record. In the case of string data, the first matching instance is used. Numeric fields, however, are summed. |
|  | CONDITIONS\_ID | Numeric | Contains a code that uniquely identifies the single conditional equation defined in the OTF\_ CONDITIONS table and used to control whether or not this output record definition applies. A value of NULL indicates that this entry should always apply. |
|  | OUTPUT\_  RECORD\_ID | Numeric | Contains a unique code that identifies the field definitions used to create this output record. This number is generated and maintained by the system. |
|  | OUTPUT\_  RECORD\_TYPE | Char(1) | Contains a single character flag that indicates the type of character set used in the output record generated for this record ID. Valid values are as follows:   * “A” – ASCII * “E” – EBCDIC   This setting MUST be consistent across all entries which generate records within a given output file (e.g., a header can not be EBCDIC with ASCII detail records). |
|  | OUTPUT\_  RECORD\_LENGTH | Numeric | Contains the number of bytes in the fixed record format or the maximum number of data bytes in variable length records as created by this output definition. |
|  | FAIL\_ACTION | Char(1) | Contains a control code that indicates what action should be taken by the system if this record has either a configuration error or if a non-system error occurs evaluating the CONDITIONS\_ID expression. Valid values include:   * “F”ail translator that will result in the daemon terminating. * “S”uspend translator that will result in this translator only being suspended. * “I”gnore the transaction which will cause the system to skip to the next NVTR transaction without doing any processing for this definition. |

**OTF\_XML Data Definition**

This table contains a single row for each output record field level definition by version number. Each entry is ONLY used if the specific field data is being written to an XML interface; otherwise, it is ignored. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | OUTPUT\_  RECORD\_ID | Numeric | Contains a unique code that identifies the field definitions used to create this output record. This number is generated and maintained by the system. |
|  | XML\_ACTION | Char(1) | Contains a control code that is ONLY used if the delivery mode is set to XML and it defines how this record plays into the XML structure. Valid values include:   * “N”ormal processing * “S”tart of a new Label Grouping * “E”nd of an existing Label Grouping * “W”rap the fields in this record * “D”ata only from this record * “X”tended data between <> no indent   The translator does not do anything to match start and ends nor does it ensure order. This is the responsibility of the person creating the configuration. |
|  | XML\_LABEL | Char(32) | Provides a fixed text string that can be placed on each individual RECORD entry. This field is only used if the XML\_ACTION of “W”rap is selected. In all other cases, it is ignored if provided. |

**OTF\_GROUPING Data Definition**

This table contains a series of rows associated with a set of ID number(s) that are used to group like records together within a multi-record transaction. For example, in order to produce a single output record for each unique category and owning location combination within a set of IE records in a single transaction simply create two rows containing CATEGORY and OWN\_LOCATION as the field names with a single ID that is placed in the GROUP\_ID for the appropriate OTF\_RECORD entry. The system will then combine[[18]](#footnote-18) the information in all the corresponding IE record instances to produce the desired output. During RTPDTRAN initialization, the system loads this information into memory for use by the translator. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | GROUP\_ID | Numeric | This is a number that uniquely identifies the grouping set. This value is used in the OTF\_RECORD table to invoke this function. |
|  | FIELD\_NAME | Char(12) | This twelve-character field identifies the field name that will be used during the translation. This label must be unique within an output record ID definition. It is not case sensitive and must only contain characters and numbers. |
|  | CASE\_SENSITIVE | Char(1) | Contains a Y/N flag that indicates if comparisons done for the purpose of grouping should be case sensitive. If the flag is “Y” then the system will not consider upper and lower case representations as equivalent. |

**OTF\_CONDITIONS Data Definition**

This table contains a single row for each host-transaction-record type combination by version number that generates one or more output records. This table identifies an output record code that should be used to generate output records if the specified record type exists and meets the criteria specified in the conditional definitions. The grouping indicator is used to connect related instances of this record type. During RTPDTRAN initialization, the system loads this information into memory for use by the translator. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | CONDITIONS\_ID | Numeric | This contains a unique numeric ID that identifies this condition for use within the system. This ID is placed in one or more OTF\_RECORD to control if the specific entry applies. |
|  | EQUATION | VarChar  (192) | This contains a numeric mathematical expression as described in section 5.5.6 that must evaluate to true for the associated OTF\_RECORD entries to apply. |

**OTF\_FIELD Data Definition**

This table contains a series of rows with a single ID that define the individual fields within the output record format. One or more[[19]](#footnote-19) rows should be created for every field that should be populated in the output record. During RTPDTRAN initialization, the system loads this information into memory for use by the translator. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | OUTPUT\_  RECORD\_ID | Numeric | This contains a code that uniquely identifies the output record format of which this field should be a part. |
|  | OFFSET | Numeric | This contains the ZERO-based offset in which this field begins. |
|  | FIELD\_NAME | Char(12) | This twelve-character field identifies the field name that will be used during host export. This label must be unique within the output record ID definition. It is not case sensitive and must only contain characters and numbers. |
|  | DATA\_TYPE | Char(1) | This flag indicates the type of output data to be produced for this field. Valid values include:   * “C”haracter String where FORMAT contains the number of bytes * “H”idden[[20]](#footnote-20) fields are identical to a “C”haracter string except that they are dropped from *DELIMITED* or *HIDDEN* output file. * “N”umeric Value where the FORMAT contains the number of characters in this field. A leading ‘+’ indicates a sign should always proceed this value extending the field size by one character. An extra ZERO at the start of the size value indicates the field should be zero filled. Finally, a “.” followed by a count at the end of the format string indicates that a decimal should appear followed by the specified number of digits. Both the decimal and the digit count are added to the overall field size. * “Z”oned Decimal where the FORMAT string contains the standard output format definition (e.g., “9(7)v99”) |
|  | FORMAT | Char(16) | This field contains information designed to control the creation of the data based on the DATA\_TYPE field. Specific content is described in the definition of that field. |
|  | CONDITION[[21]](#footnote-21) | VarChar  (192) | This contains a numeric mathematical expression as described in section 5.5.6 that must evaluate to true for the associated OTF\_FIELD entry to apply. |
|  | EQUATION | VarChar  (192) | This contains a numeric or string mathematical expression as described in section 5.5.6 that is used to provide the value necessary to populate this field. The selection of string versus numeric expression is driven based on the need of the DATA\_TYPE field. |
|  | FAIL\_ACTION | Char(1) | Contains a control code that indicates what action should be taken by the system if this record has either a configuration error or if a non-system error occurs evaluating either the CONDITION or EQUATION expression. Valid values include:   * “F”ail translator that will result in the daemon terminating. * “S”uspend translator that will result in this translator only being suspended. * “D”efault value is inserted into this field and processing continues. At publication, this setting can only be used for fields with a data type of either “C” or “N”. * “I”gnore the transaction which will cause the system to skip to the next NVTR transaction without doing any processing for this definition. |
|  | DEFAULT\_VALUE | VarChar (128) | Contains the default value inserted into this field if an evaluation or configuration error occurs and the FAIL\_ACTION is set to “D”. |
|  | ENDSTAGE\_ SECURE | VarChar (128) | Contains an express used to replace this field during the final build of a interfaces whose delivery mode is either *DELIMITED* or *HIDDEN*  to better secure the data. During evaluation, the original value can be accessed using the variable “{$ORIG\_VALUE}”. Evaluation of this expression is always based on the STRING guidelines. This expression is ignored in all other delivery modes. |

**OTF\_CODE Data Definition**

The OTF\_CODE and OTF\_TRANSLATE tables are used to provide the database based validation (DBIN) and translation (DBXLAT) services described in section 5.5.6 of this document. Translations contain a two-level hierarchy. The top level defines the type of translation or set of codes that are being translated. This table provides a set of valid top-level codes used by the Web Interface to assist with updating and maintaining the information within the translation tables. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | SET\_CODE | Char(12) | Contains a twelve-character label that uniquely identifies the set of values that this translation is designed to support. |
|  | DESCRIPTION | Char(32) | Contains a human readable description used during configuration for this translation. |
|  | LAST\_UPDATE | Date/Time | Contains the date and time that this entry and/or set of associated values were last updated. |

**OTF\_TRANSLATE Data Definition**

This table contains the specific translation information used by each instance of RTPDTRAN to provide the validation (DBIN) and translation (DBXLAT) services. There is a single row contained within this table for each unique SET and KEY value that requires translation. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | SET\_CODE | Char(12) | Contains the twelve-character label that uniquely identifies the set of values to which this entry belongs. This value must exist in the OTF\_CODE table. |
|  | KEY | VarChar  (96) | Contains the key that this value represents. This is the input value in a DBIN or DBXLAT call. |
|  | RESULT | VarChar  (96) | Contains the value substituted for the key when used with the DBXLAT command. DBIN does not use this field. |
|  | CACHE\_MODE | Char(1) | Contains a flag that indicates how this record should be cached. Valid values include “P”reload, “O”n Demand, and “N”ever. |
|  | DELETE\_FLAG | Char(1) | Contains a Y/N flag used when the table is being fully refreshed to remove records that are not contained within the load file. |
|  | LAST\_UPDATE | Date/Time | Contains the date and time that the entry was last modified. |
|  | USER\_ID | Numeric | Contains the User Number for the user that last updated this record. A value of ZERO indicates that it was update by a batch process. |

#### Bundle Definitions

The bundle definition tables provide detailed descriptions on how to build the individual files that are transmitted to the associated host. These descriptions define how records should be grouped, the sort order for the output file, and if records should be summarized or detailed. Information is retrieved from these tables during the initialization of each of the DTRAN instances based on the provided host codes and is maintained throughout system operation. Figure 18 shows the basic record structure along with the high-level organization. The *CCfgOut* class is used to access the corresponding data.



Figure 18.Bundle Format Definition Table Structure

**HBF\_FILE Data Definition**

This table contains a single row for each unique host that receives data from this system. The entry in this table defines the overall file structure if there is no specific HBF\_OGRANIZATION row of type “I” that defines information in an individual file. The type “I” record always takes precedence over this information. During RTPDTRAN initialization, the system looks for the appropriate host code and uses that data to initialize the translator. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | HOST\_CODE | Char(2) | Contains the two-character host identification code that uniquely identifies the host information. This code is used by RTPDTRAN to identify the appropriate configuration information. |
|  | HDR\_OUTPUT\_  RECORD\_ID | Numeric | Contains a unique code that identifies the field definitions used to create the group header. A value of ZERO indicates that no header is required at the group level. This number is generated and maintained by the system. |
|  | HDR\_OUTPUT\_  RECORD\_LENGTH | Numeric | Contains the number of bytes in the fixed record format or the maximum number of data bytes in variable length records as created by the header output definition. |
|  | TRL\_OUTPUT\_  RECORD\_ID | Numeric | Contains a unique code that identifies the field definitions used to create the group trailer. A value of ZERO indicates that no trailer is required at the group level. This number is generated and maintained by the system. |
|  | TRL\_OUTPUT\_  RECORD\_LENGTH | Numeric | Contains the number of bytes in the fixed record format or the maximum number of data bytes in variable length records as created by the trailer output definition. |
|  | OUTPUT\_  RECORD\_TYPE | Char(1) | Contains a single character flag that indicates the type of character set used in the output record generated for this record ID. Valid values are as follows:   * “A” – ASCII * “E” – EBCDIC   This setting MUST be consistent across all entries which generate records within a given output file (e.g., a header can not be EBCDIC with ASCII detail records). |
|  | SEQ\_NUM\_  FIELDNAME | Char(12) | Contains a field name within the output record that should receive the sequence number during its final write to the host file. A value of all spaces or null indicate that there is no sequence number in this output type. |
|  | SECURE\_FLAG | Char(1) | Contains a “Y”es or “N”o value indicating if this interface is secure between DTRAN and SHUTL. The default value is “N” which is assumed if the parameter is not specified. **WARNING:** This parameter is currently ONLY supported with PGP transports from shuttle. Use with other transport types will create unusable files. This is also ONLY supported with Delimited files. The parameter is ignored with all other output types. |
|  | XML\_VERSION | Char(16) | Contains a string of up to 16 characters that represent the XML version if this interface generates XML. For all other delivery modes, this parameter is ignored. |

**HBF\_ORGANIZATION Data Definition**

This table contains zero or more rows that define how information contained within the file is grouped. The absence of rows for a given host code causes the system to write all detail records in the order collected. At publication, the system will support up to two layers of grouping. During RTPDTRAN initialization, the system looks for the appropriate host code and uses that data to initialize the translator. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | HOST\_CODE | Char(2) | Contains the two-character host identification code that uniquely identifies the host information. This code is used by RTPDTRAN to identify the appropriate configuration information. |
|  | GROUP\_LEVEL | Numeric | Contains the level of grouping defined by this record. Groupings are nested starting with level ZERO. |
|  | GROUP\_ID | Numeric | Contains a code that identifies one or more grouping records in the HBF\_GROUPING table that controls which record instances are reviewed as a set rather than individuals. Multiple records grouped together produce only a single output record. In the case of string data, the first matching instance is used. Numeric fields, however, are summed. |
|  | TYPE | Char(1) | Contains a flag that indicates how this organizational record should be implemented. The following is a list of valid values:   * “I”ndividual file – Creates a new file for each unique value. * “D”etail – Groups multiple records matching this criteria into a single section of the output file. * “S”ummarized - Summarizes multiple records matching this criteria into a single record.   \*\* I and D entries are mutually exclusive and should never be combined. It is also important to only use one HGRP field with I and D entries in order to prevent issues related to data load order. \*\* |
|  | SORT | Char(1) | Contains a single character flag that controls the order in which the collection records are stored within the file. Valid values include:   * “R”eceived – Collections are written to the output file in the order that they first appeared in the input file. Summarized records are written in reverse order (e.g., last in first out). * “A”scending – Collections are written in ascending order based on the key. * “D”escending – Collections are written in descending order based on the key.   This setting only apply to organizational entries of the type *Detail*. |
|  | HDR\_OUTPUT\_  RECORD\_ID | Numeric | Contains a unique code that identifies the field definitions used to create the group header. A value of ZERO indicates that no header is required at the group level. This number is generated and maintained by the system. |
|  | HDR\_OUTPUT\_  RECORD\_LENGTH | Numeric | Contains the number of bytes in the fixed record format or the maximum number of data bytes in variable length records as created by the header output definition. |
|  | TRL\_OUTPUT\_  RECORD\_ID | Numeric | Contains a unique code that identifies the field definitions used to create the group trailer. A value of ZERO indicates that no trailer is required at the group level. This number is generated and maintained by the system. |
|  | TRL\_OUTPUT\_  RECORD\_LENGTH | Numeric | Contains the number of bytes in the fixed record format or the maximum number of data bytes in variable length records as created by the trailer output definition. |
|  | OUTPUT\_  RECORD\_TYPE | Char(1) | Contains a single character flag that indicates the type of character set used in the output record generated for this record ID. Valid values are as follows:   * “A” – ASCII * “E” – EBCDIC   This setting MUST be consistent across all entries which generate records within a given output file (e.g., a header can not be EBCDIC with ASCII detail records). |
|  | SEQ\_NUM\_  FIELDNAME | Char(12) | Contains a field name within the output record that should receive the sequence number associated with a collection grouping if it is defined within a record during its final write to the host file. A value of all spaces or null indicate that there is no sequence number in this output type. |
|  | INDEX\_  GROUPSIZE | Number | Optional field contains the number of characters at the start of the field that can be indexed by linked list. This field is optional even if an index is in use. Indexing is only supported on collections of type “D” with a sort order of “A”. This value is ignored for all other collections. Indexing is ONLY enabled if this and the INDEX\_CHARCNT are provided. A value of ZERO is assumed if not specified. |
|  | INDEX\_  CHARCNT | Number | Optional field that is used for index searches – index searches are ONLY available for collections with type “D” and with the sort set to “A”. These values are ignored on all other collections. This indicates the number of characters collapsed into the high level indexes. Indexing is not used if not provided. This value can not exceed 11 characters – 11 is assumed if this value is exceeded. |
|  | INDEX\_  ARRAYSIZE | Number | Optional field that is used for index searches. This indicates the size of the quick access hash table. This field is required if INDEX1\_CHARCNT is provided. A default value of 64 is assumed. |

**HBF\_GROUPING Data Definition**

This table contains a series of rows associated with a set of ID number(s) that are used to group like records together within a multi-record transaction in a fashion similar to the OTF\_GROUPING table previously defined. This table, however, references field names in output records rather than input records. During RTPDTRAN initialization, the system loads this information into memory for use by the translator. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | GROUP\_ID | Numeric | This is a number that uniquely identifies the grouping set. This value is used in the HBF\_ ORGANIZATION table to invoke this function. |
|  | FIELD\_NAME | Char(12) | This twelve-character field identifies the field name that will be used during the translation. This label must be unique within a output record ID definition. It is not case sensitive and must only contain characters and numbers. |
|  | CASE\_SENSITIVE | Char(1) | Contains a Y/N flag that indicates if comparisons done for the purpose of grouping should be case sensitive. If the flag is “Y” then the system will not consider upper and lower case representations as equivalent. |

**HBF\_REQUIRED Data Definition**

This table contains zero or more rows for each unique host that receives data from this system. Each entry indicates that a given input transaction code is required within a bundle. If a code is listed and not contained within the bundle file at the time of bundling, the system will defer delivery until the next trigger point. This may mean, for example, if a store close is not received before the cut-off time then all sales are held until the next day. During RTPDTRAN initialization, the system looks for the appropriate host code and uses that data to initialize the translator. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | HOST\_CODE | Char(2) | Contains the two-character host identification code that uniquely identifies the host information. This code is used by RTPDTRAN to identify the appropriate configuration information. |
|  | TRANSACTION\_  CODE | Char(3) | This three-character field uniquely identifies the transaction. This code is contained within the *Base Sequence* of corresponding transaction and is used as the basis for interpreting all the data within the transaction. |

**HBF\_ORDER Data Definition**

This table contains zero or more rows for each unique host that receives data from this system. Each entry in sequence indicates the order in which records should be migrated to the final file. A value of “\*\*\*” in a record transaction code indicates the point at which all transactions not spelled out in this list should be inserted in sequential order. If a “\*\*\*” is not specified then it is assumed to be the last entry in the list. During RTPDTRAN initialization, the system looks for the appropriate host code and uses that data to initialize the translator. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | HOST\_CODE | Char(2) | Contains the two-character host identification code that uniquely identifies the host information. This code is used by RTPDTRAN to identify the appropriate configuration information. |
|  | SEQUENCE\_  NUMBER | Numeric | Contains a number starting with ZERO that is used to indicate the order in which the transactions should appear in the output file. |
|  | TRANSACTION\_  CODE | Char(3) | This three-character field uniquely identifies the transaction. This code is contained within the *Base Sequence* of corresponding input transaction. A value of “\*\*\*” indicates the point in which all non-specified transaction codes should appear in the file. |
|  | TYPE | Char(1) | Contains a flag that indicates how this organizational record should be implemented. The following is a list of valid values:   * “F”irst uses the first instance of the record only. * “L”ast uses the last instance of the record only. * “D”etail includes all the records of this type. |

**HBF\_SEQ Data Definition**

This table contains no more than one row for each unique host if and only if special sequence number assignments are required. At the time of publication, this record type is ONLY supported for HIDDEN field delivery modes. An entry in this table defines information which defines where the data goes within a record and what is considered a new count increment the counter. During RTPDTRAN initialization, the system looks for the appropriate host code and uses that data to initialize the translator. A record in this table is NOT required and if missing is assumed to have no impact on the systems operation. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | HOST\_CODE | Char(2) | Contains the two-character host identification code that uniquely identifies the host information. This code is used by RTPDTRAN to identify the appropriate configuration information. |
|  | ORGANIZATION\_FIELD\_ NAME\_0 | Char(12) | Contains the name of the field as defined in the output record for which that the sequence number should be maintained. A blank value or if the field does not exist for the given record then the count will be rolled against the master sequence count. |
|  | ORGANIZATION\_FIELD\_ NAME\_1 | Char(12) | Contains the name of the field as defined in the output record for which that the sequence number should be maintained. A blank value or if the field does not exist for the given record then the count will be rolled against the master sequence count. |
|  | ORGANIZATION\_FIELD\_ NAME\_2 | Char(12) | Contains the name of the field as defined in the output record for which that the sequence number should be maintained. A blank value or if the field does not exist for the given record then the count will be rolled against the master sequence count. |
|  | ORGANIZATION\_FIELD\_ NAME\_3 | Char(12) | Contains the name of the field as defined in the output record for which that the sequence number should be maintained. A blank value or if the field does not exist for the given record then the count will be rolled against the master sequence count. |
|  | START\_POINT | Number | Contains the value that the counter should be set to for each new organization field. |
|  | SUB\_FIELD\_ NAME | Char(12) | Contains the name of the field as defined in the output record that should get the sequence number. If the field exists it will be replaced with the value currently contained within the sequence counter. |
|  | FORMAT | Char(16) | Contains a standard numeric field format for this value indicating the number of digits to use. A leading zero will cause the system to fill the value to the left with leading ZEROs for the desired length; otherwise, the field is left justified filled with blanks. |
|  | OFFSET | Number | The zero based index in the output record at which point to start the comparison when checking to see if this record bumps the sequence counter. |
|  | PATTERN | Char(48) | The pattern to search for at the offset specified which indicates the counter should be bumped. |

**HBF\_SEQ\_EXCLUDE Data Definition**

This table contains zero to many rows that control record types that should be excluded from the count process outlined in the HBF\_SEQ table. No rows should exist in this table if there is not a corresponding row in the HBF\_SEQ table. Each row defines a comparison that if true will exclude a record from incrementing the sequence count. This information is ONLY reviewed if the record matches the criteria outlined in the HBF\_SEQ table. During RTPDTRAN initialization, the system looks for the appropriate host code if a corresponding HBF\_SEQ entry exists and uses that data to initialize the translator. A record in this table is NOT required and if missing is assumed to have no impact on the systems operation. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | HOST\_CODE | Char(2) | Contains the two-character host identification code that uniquely identifies the host information. This code is used by RTPDTRAN to identify the appropriate configuration information. |
|  | SEQUENCE | Number | Contains a numeric value to provide a unique primary key for this table. |
|  | OFFSET | Number | The zero based index in the output record at which point to start the comparison when checking to see if this exclusion applies. |
|  | PATTERN | Char(48) | The pattern to search for at the offset specified which indicates this exclusion has been met. |

**HWT\_MASTER Data Definition**

This table contains a single row for each unique host that requires additional wait criteria. The entry in this table defines the overall high-level wait parameters for time triggered bundling ONLY. During RTPDTRAN initialization, the system looks for the appropriate host code and uses that data to initialize the translator. A record in this table is NOT required and if missing is assumed to have no impact on the systems operation. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | HOST\_CODE | Char(2) | Contains the two-character host identification code that uniquely identifies the host information. This code is used by RTPDTRAN to identify the appropriate configuration information. |
|  | MINIMUM\_ PERCENT\_ COMPLETE | Number (1-Assumed Decimal) | Numeric value between 0 and 100 percent indicating how caught up the translator must be to accommodate proper bundling. A value of ZERO is assumed which means that it bundles immediately. A value of 100% means that all inbound records in the NVTR must be processed before the system will bundle. |
|  | SECOND\_CHECK\_SECONDS | Number | Numeric value between 0 and 59 which indicates a number of seconds to wait before rechecking the MINIMUM\_PERCENT\_COMPLETE. This parameter is designed to prevent a restart or delay in batch from causing data issues. Do NOT raise this value too high since it suspends the DTRAN process for this period of time waiting for the second check. This is NOT an asynchronous check like the others in the DTRAN process. A value out side the provided range assumes 8 seconds. A value of ZERO eliminates the second check. |
|  | MINIMUM\_ RECORD\_COUNT | Number | Indicates the minimum number of records expected in each bundling session. A value of ZERO will cause this parameter to be ignored. |
|  | WAIT\_MINUTES | Number | Indicates how many minutes to wait between testing of these conditions. The recommended value of 3 minutes will prevent multiple pages without overloading the system. Times over 5 minutes may result in multiple pages. |
|  | PAGE\_TIME | Char (HHMM) | This time based on a 24 hour clock indicates when the system should page if the bundling has not begun. A value of “9999” indicates that the system should never page. WARNING: This functionality will not handle a paging time and drop dead time crossing over mid-night. For Interval and Multiple times, this should ONLY be used for the latest trigger in a given day 00:00 through 23:59. |
|  | DROP\_DEAD\_  TIME | Char (HHMM) | This time based on a 24 hour clock indicates when the system should proceed with bundling even if the criteria is not met. A value of “9999” indicates that this should never occur. WARNING: This functionality will not handle a trigger time and drop dead time crossing over mid-night. For Interval and Multiple times, this should ONLY be used for the latest trigger in a given day 00:00 through 23:59. |
|  | LOCATION\_  PERCENT\_1 | Number (One  Assumed Decimal) | Contains the percentage of locations in group one that must be considered included prior to proceeding with bundling. A value of ZERO indicates we should proceed without this group. |
|  | LOCATION\_  PERCENT\_2 | Number (One  Assumed Decimal) | Contains the percentage of locations in group two that must be considered included prior to proceeding with bundling. A value of ZERO indicates we should proceed without this group. |
|  | LOCATION\_  PERCENT\_3 | Number (One  Assumed Decimal) | Contains the percentage of locations in group three that must be considered included prior to proceeding with bundling. A value of ZERO indicates we should proceed without this group. |
|  | LOCATION\_  PERCENT\_4 | Number (One  Assumed Decimal) | Contains the percentage of locations in group four that must be considered included prior to proceeding with bundling. A value of ZERO indicates we should proceed without this group. |

**HBF\_SCAN Data Definition**

This table contains a series of zero or more rows single for each unique host that requires a “scan” process to remove information that is being held in the bundle file for an extended period of time. During RTPDTRAN initialization, the system looks for the appropriate host code and uses that data to initialize the translator. A record in this table is NOT required and if missing is assumed to have no impact on the systems operation. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | HOST\_CODE | Char(2) | Contains the two-character host identification code that uniquely identifies the host information. This code is used by RTPDTRAN to identify the appropriate configuration information. |
|  | TIME\_STAMP | Char(4) | Contains the time of day that this entry applies too in the format HHMM. |
|  | MODE | Char(8) | Contains the mode and number of days under which a transaction should have this method applied. Valid values include:   * B# – Transaction with a business date where # is the number of days older than the current business date for the location. * C# - Transaction with a business date where # is the number of days old it needs to be from the current date. |
|  | DISPOSE | Char(1) | Flag indicates the results of this operation. Valid values include:   * F-orce Transaction to Output * D-ump Transaction without Sending It |
|  |  |  |  |
|  |  |  |  |

#### Unload Transaction Utility

The “Unload Transaction Utility” tables provide detailed descriptions on how to build compliant transactions for the system to process based on unload(s) of internal table(s) and/or view(s). These descriptions define what information is pulled and how it is formatted into a transaction that is posted into the system. Information is retrieved from these tables at the start of each RTPUBATCH run.

**UTQ\_HEADER Data Definition**

This table contains a single row for each database query that can occur. Fundamentally there are two types of queries that can be defined in this table – a SINGLE primary query defines the transaction while zero or more secondary queries can define appropriate records within the transaction. During RTPUBATCH initialization, the system looks for the appropriate extract code and uses that data to initialize the program instance. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | EXTRACT\_CODE | Char(4) | Contains the four-character identification code that uniquely identifies the unload definition that is to be executed. |
|  | TYPE | Char(1) | M=Master; S=Secondary |
|  | SEQUENCE | Number | A unique continuous sequence of numbers starting with ZERO for all combinations of EXTRACT\_CODE and TYPE. |
|  | DB\_OBJECT | Char(32) |  |
|  | DB\_WHERE | Char(250) | In secondary queries - $..$ will allow substitution of values from the primary query where the … is the name of the field in the primary query. |
|  | DB\_GROUP | Char(128) |  |
|  | DB\_ORDER | Char(128) |  |
|  | EVAL\_ CONDITION | Char(192) |  |
|  | MODE | Char(1) | A=Always; C=on Change |
|  | KEY\_FIELD | Char(48) |  |
|  | IN\_VERSION | Char(2) |  |
|  | OUT\_CODE | Char(3) |  |
|  | IN\_RECID | Number |  |
|  | OUT\_RECID | Number |  |

**UTQ\_GETFIELD Data Definition**

This table contains a single row for each field that should be retrieved as part of the query identified in the UTQ\_HEADER configuration as well as if this impacts the success of the change analysis. Selection of rows from this table is based on the IN\_RECID specified in the corresponding header record. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | REC\_ID | Number |  |
|  | DB\_NAME | Char(48) |  |
|  | SIZE | Number |  |
|  | CHANGE\_CHECK | Char(1) |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**UTQ\_FIELD Data Definition**

This table contains a single row for each field that should be written for a specific header or record. Selection of rows from this table is based on the OUT\_RECID specified in the corresponding UTQ\_HEADER record. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | OUT\_RECID | Numeric | This contains a code that uniquely identifies the output record format of which this field should be a part. |
|  | OFFSET | Numeric | This contains the ZERO-based offset in which this field begins. |
|  | FIELD\_NAME | Char(12) | This twelve-character field identifies the field name that will be used during host export. This label must be unique within the output record ID definition. It is not case sensitive and must only contain characters and numbers. |
|  | DATA\_TYPE | Char(1) | This flag indicates the type of output data to be produced for this field. Valid values include:   * “C”haracter String where FORMAT contains the number of bytes * “H”idden[[22]](#footnote-22) fields are identical to a “C”haracter string except that they are dropped from *DELIMITED* or *HIDDEN* output file. * “N”umeric Value where the FORMAT contains the number of characters in this field. A leading ‘+’ indicates a sign should always proceed this value extending the field size by one character. An extra ZERO at the start of the size value indicates the field should be zero filled. Finally, a “.” followed by a count at the end of the format string indicates that a decimal should appear followed by the specified number of digits. Both the decimal and the digit count are added to the overall field size. * “Z”oned Decimal where the FORMAT string contains the standard output format definition (e.g., “9(7)v99”) |
|  | FORMAT | Char(16) | This field contains information designed to control the creation of the data based on the DATA\_TYPE field. Specific content is described in the definition of that field. |
|  | CONDITION[[23]](#footnote-23) | VarChar  (192) | This contains a numeric mathematical expression as described in section 5.5.6 that must evaluate to true for the associated OTF\_FIELD entry to apply. |
|  | EQUATION | VarChar  (192) | This contains a numeric or string mathematical expression as described in section 5.5.6 that is used to provide the value necessary to populate this field. The selection of string versus numeric expression is driven based on the need of the DATA\_TYPE field. |
|  | FAIL\_ACTION | Char(1) | Contains a control code that indicates what action should be taken by the system if this record has either a configuration error or if a non-system error occurs evaluating either the CONDITION or EQUATION expression. Valid values include:   * “F”ail translator that will result in the daemon terminating. * “S”uspend translator that will result in this translator only being suspended. * “D”efault value is inserted into this field and processing continues. At publication, this setting can only be used for fields with a data type of either “C” or “N”. * “I”gnore the transaction which will cause the system to skip to the next NVTR transaction without doing any processing for this definition. |
|  | DEFAULT\_VALUE | VarChar (128) | Contains the default value inserted into this field if an evaluation or configuration error occurs and the FAIL\_ACTION is set to “D”. |

#### File Delivery System Definitions

The file delivery system definition tables provide detailed information about how the files created by the *RTPDTRAN* instance are to be delivered to the destination system. Information is retrieved from these tables during the initialization of *RTPSHUT*. Figure 18 shows the basic record structure along with the high-level organization. The *CCfgFDS* class is used to access the corresponding data.



Figure 19.File Delivery System Definition Table Structure

**FDS\_HOST Data Definition**

This table contains a single row for each unique host that receives data from this system. The entry provides overall information on how these files should be delivered to the receiving system such as user id, password, transfer type, etc…. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | HOST\_CODE | Char(2) | Contains the two-character host identification code that uniquely identifies the host information. This code is used to identify which DTRAN output files should use this transfer definitions. |
|  | XFER\_TYPE | Char(1) | Contains a single character identifier that indicates what type of delivery is required. The following codes are valid:   * “M”ove file to local directory * “F”TP to remote system * “Q” to use MQ Series to deliver data * “T”ask executed to process file * “S”ocket delivery via TCP/IP * “L”ock based Two Step FTP * “P”GP with FTP Transfer * “X”fer using SFTP * “Y” SFTP Transfer with Two Step Lock * “R” PGP with SFTP Transfer |
|  | DEST\_ADDR | VarChar2 (128) | Contains information that identifies where the files should be delivered. The information contained within this string is specific to the transfer type as follows:   * “M” – Destination path on the local machine followed by a “|” followed by a CHMOD permission number followed by a “|” followed by a new owner. All parameters except the path are optional. * “F” – Contains the host name or IP address of the destination system optionally followed by a “|” followed by the completion code for FTP on the destination host. By default, the system assumes 226 is the completion code. * “Q” – Contains the name of the queue manager followed by a “|” followed by the queue object. * “T” – Contains the task name. Optionally the name of the task can be followed by a “|” which proceeds the directory the file should be placed in before the task is initiated. If not provided, then the file is not moved. * “S” – Contains the following control data in order separated by “|”: host name or IP address, port number, record header template[[24]](#footnote-24), marker in the last record, marker at the start of the next record, seconds to wait before connecting, and seconds to wait before sending[[25]](#footnote-25) the next packet. All proceeding values must exist even if they are blank. A value of ZERO in either wait time causes the system to immediately proceed. A sample test string might be as follows: 127.0.0.1|50|REC^^^^|DST|DS|2|1 |
|  | USERID | Char(16) | Contains the user ID used to access the remote system if required by the transfer type. |
|  | PASSWORD | Char(16) | Contains the password used to access the remote system if required by the transfer type. |
|  | ACCOUNT | Char(16) | Contains the account used to access the remote system if required by the transfer type and the remote system. |
|  | STATUS | Char(1) | Contains a status code for this transfer definition. Valid values include:   * “A”ctive * “I”nactive |

**FDS\_FILE Data Definition**

This table contains a single row for each unique host file type that should be delivered to this host from this single DTRAN instance. The entry provides the necessary information to identify this type of file and an equation used to calculate the destination filename. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | HOST\_CODE | Char(2) | Contains the two-character host identification code that uniquely identifies the host information. This code is used to identify which DTRAN output files should use this transfer definitions. |
|  | SEQUENCE | Numeric | Contains a sequence number starting with zero that clearly identifies the order in which these conditions should be checked. One and only one entry is used for each file even if multiples match – this controls which entry will be checked first. |
|  | CONDITION | VarChar2 (192) | Contains a file evaluation string that must evaluate to true before the entry is used. A blank entry indicates this entry applies to all files. |
|  | EQUATION | VarChar2 (192) | Contains a file evaluation string that is used to generate the destination filename. |

**FDS\_COMMAND Data Definition**

This table contains zero or more rows that are used to generate the command sequence for FTP transfers. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | HOST\_CODE | Char(2) | Contains the two-character host identification code that uniquely identifies the host information. This code is used to identify which DTRAN output files should use this transfer definitions. |
|  | SEQUENCE | Numeric | Contains the sequence number that corresponds to the appropriate FDS\_FILE entry. |
|  | EXECUTION\_ ORDER | Numeric | Contains a unique sequence number starting with zero that clearly identifies the order in which the commands should be executed. |
|  | CMD\_STRING | VarChar2 (128) | Contains the string that clearly represents the command to be executed. A value of “%” is replaced with the fully qualified source filename. A value of “^” is replaced with the destination filename. |

#### Special Translation Table Entries

*OTF\_CODE* and *OTF\_TRANSLATE* tables not only provide custom translation tables but also control tables for some internal functions. The following section(s) describe those special use entries.

**Discount Calculation Configuration**

The *DISCOUNTCONFIG* translation information is used internally by the DISC command within an expression to determine how much an item is discounted. This information controls how item and transaction discounts are applied to the original unit price. One or more rows contained within the translation table define actual items that are involved in transaction level discounts. The key for these rows is composed of a *user specified label* followed by the literal “ITEM” followed by an index starting with “00”. The following table defines the fields contained on the rest of the line.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Validation** | **Default** |
| MORE | Contains a Y/N flag that indicates if additional item types are listed for this discount definition. The system will then increment the index to the next value. | Char(1) | Y or N | N |
| RECORD\_TYPE | Contains a two-character record type that identifies the records that should be evaluated using this configuration. | Char(2) | Any valid record type | None |
| SPREAD\_FIELD | Contains the name of the field that is used to calculate the spread of discounts in case of a transaction discount. | Char(12) | Any valid field name | None |
| VOID\_FIELD | Contains the name of the field that is used to indicate that a record of this type is voided. If the value is “Y” then it is ignored for the sake of these calculations. | Char(12) | Any valid field name | None |
| TRANS\_  DISCOUNT | Contains the name of the field that is used to flag whether this record is actually included in any transaction level discounts. A value of “Y” in this field includes the record type in transaction discount calculations. | Char(12) | Any valid field name | None |
| DISCOUNT\_  FIELD | Contains the name of the field that has the sequence code of any item level discounts that apply. A blank value indicates that there is never an item discount. | Char(12) | Any valid field name | None |
| DISCOUNT\_  AMOUNT | Contains the name of the field in the item discount record that has the amount of the discount. | Char(12) | Any valid field name | None |
| NEXT\_  DISCOUNT | Contains the name of the field in the item discount record that contains the sequence code of any follow on or secondary discount records. | Char(12) | Any valid field name | None |

### Item Maintenance

The item maintenance table contains information about non-inventory items that is used for price maintenance to the sales devices and for translation of tanami information including Issue Number, Material Number, and Distribution Channel as appropriate. The table is maintained through the web interface and information is transferred to the appropriate tables through the batch processes.

.



Figure 20.Item Maintenance Table Structure

**ITM\_MASTER Data Definition**

This table contains a single row for each unique non-inventory item number. Data within this table is used to maintain price lookup information in the POS systems and for translation from an item number to the appropriate tanami values. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | ITEM\_NUMBER | Char(22) | Contains the SKU or Item number that uniquely identifies this product. |
|  | NAME | Char(18) | Contains the name associated with this item. |
|  | SALE\_QUANTITY | Numeric | Contains the number of items within a pricing unit. |
|  | SALE\_PRICE | Numeric | Contains the price associated with this item in cents. |
|  | ITEM\_TYPE | Char(1) | Contains a single character code that indicates the type of product represented by this record. Valid values are based on the item type definitions listed in the *IE* record description in the *Batch Specification* document. |
|  | TAX\_TYPE | Char(1) | Contains a single character code that indicates the method for applying tax to this product. Valid values are defined in the definition of *Tax Type Indicator* contained within the *Batch Specification* document. |
|  | DEPARTMENT\_  CLASS | Char(4) | Contains the four-character department class that is used when feeding sales to the Retail Sales system. |
|  | ISSUE\_NUMBER | Char(20) | Contains the Tanami issue number that identifies the product along with the associated character or royalty. |
|  | MATERIAL\_  NUMBER | Char(20) | Contains the Tanami number that identifies the type of product. |
|  | DISTRIBUTION\_ CHANNEL | Char(2) | Contains the distribution channel associated with the product. This value is used in calculation of royalties. |
|  | STATUS | Char(1) | Contains a record status code. Valid codes include:   * “A”ctive Record * “I”nactive Record * “D”eleted Record |
|  | LAST\_MODIFIED | Date | Contains the date and time that this item record or any of the related records[[26]](#footnote-26) where updated. |

**ITM\_LOCATION\_REFRERENCE Data Definition**

This table provides a relationship between an item and the locations in which it is sold. There is one row for each unique item and location combination. This table is used to control download of item information to the selling devices. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | ITEM\_NUMBER | Char(22) | Contains the item number that uniquely identifies the related production within the ITM\_MASTER table. |
|  | LOCATION\_KEY | VarChar  (5-16) | Contains the location component of the transaction key. This value uniquely identifies the physical selling location. |
|  | LAST\_  TRANSFERRED | Date | Contains the date and time that this item was last transferred to the selling device. |

**ITM\_ALIAS Data Definition**

This table is used to provide the selling device with a secondary ID number such as EAN or SKU that should be tied to a primary item record for the purpose of pricing and Tanami value conversions. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | ITEM\_NUMBER | Char(22) | Contains the item number that uniquely identifies the related production within the ITM\_MASTER table. |
|  | ALIAS | Char(22) | Contains the SKU/item number associated with the related product. This value is sent to POS to tie any product with this value to the original pricing and tanami entry. |

**ITM\_VALIDATION Data Definition**

This table contains a single row for each valid Tanami issue number. The table is used to make sure that any issue number entered into RTP and Merlin are valid. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | ISSUE\_NUMBER | Char(20) | Contains the unique Tanami value associated with this record. |
|  | DESCRIPTION | Char(40) | Contains a human readable description used for display and for searching for a desired issue number. |
|  | ROYALTY\_FLAG | Char(1) | Contains a Y/N flag that is set to Y if the issue number represents a royalty item. |
|  | PRODUCT\_TYPE | Char(20) | Contains the Tanami product type code associated with products attached to this issue number. |

**ITM\_CHARACTER Data Definition**

This table contains a single row for each character that is represented by the associated Issue Number identified in the ITM\_VALIDATION. Data within this table is used for searches. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | CHARACTER\_  NAME | Char(24) | Contains the name of the related character. |
|  | ISSUE\_NUMBER | Char(20) | Contains the associated Tanami value in the ITM\_VALIDATION table. |

### Mathematical Expressions

The *Data Driven Translator* for the RTP system is controlled by a series of mathematical expressions contained within the outbound configuration tables. This approach was selected to allow for easy modification, easier review, and maximum flexibility. All equations are evaluated during operation and can be changed without the need for new software releases and/or code modifications. The following sections define the specific functionality supported within these processes during the initial implementation of the system. Supported functionality is based on the type of result anticipated for the field this result will populate.

#### String Results

String

Numeric

Parenthesis

LEFT

RIGHT

MID

ITOA

CHR (e.g., “CHR(39)” will generate a string of a single quote)

XLAT (e.g., “<INPUT>XLAT(1=Y;0=N)” a compare of “\*DFLT\*” sets a value if nothing matches)

DBXLAT (e.g., “<INPUT>DBXLAT($..$;#)” where $..$ is the code set and #=Field Index is used)

DBXLAT($?..?) where ?..? is a location table variable: B=Business Area, S=Source Company,

H=Charge Company, P=LOB Profit Center, C=LOB Cost Center, R=Register Number,

I=Inventory Location, O=Operating Origin, D=Department, N=Store Number,

A=Name, V=Selling Device, L=Line of Business, G=Sales Organization, F=Division,

X=Participant

DBXLAT($?..?#) where ?..? is a location table variable (P=Profit Center, C=Cost Center, B=Business

Area, S=Source Company, H=Charge Company) and # is the single character line of

business item type as defined in the batch specification or a string expression that

evaluates to a line of business item type. The system attempts to locate the value from

the following sources in order: center defined for the specified LOB, for the location’s

LOB, and finally for the first alphabetic LOB entry.

DBXLAT($M#) retrieves the merchant ID corresponding to the card type identified by the value in #

where valid values are those specified in the definition of HLC\_MERCHANT\_CODES.

DBXLAT(~?..?) retrieves the shop number based on the value indicated by ?..? where valid values include

R=Register Number, O=Origin/Department, I=Inventory Location

DBXLAT(\*) locates the group associated with the filename passed in from the output file history.

IFELSE($1;$2;$3) this is a conditional string evaluation. $1 contains the condition that is evaluated as a

numeric. If the value is true, then the system returns the results of evaluating the string

expression presented by $2. Otherwise, it evaluates and returns $3.

FIND (e.g., FIND(IE;0) or FIND(IE;[ITEM\_TYPE]==’F’;0)) returns sequence code

CHAINUP (e.g., “CHAINUP(1..1;IE-2..2;IG-3..3)”) returns the sequence code for the top most record in

the chain. “1..1” identifies the chain field in the current

record; the balance of the one or more parameters  
identifies potential top level record types along with the

field used to start the chain.

SHIFT($..$;#..#) makes a shift in time for a specified number of days. The first parameter $..$ is the basis

of the time shift which should be a standard date of the form YYYYMMDD. The second

numeric parameter is the number of days you want to shift the result. A positive number

moves time forward with a negative value moving to the past. The resulting string is in

the form YYYYMMDD.

DOW($..$;#;?..?) where $..$ is a calculation that results in a base date in the form YYYYMMDD; # is the

day of week index where 0=Sunday;…;6=Saturday; and ?..? is optionally an output date

format. The format string is based on the STRFTIME standards. The default format is

YYYYMMDD. DOW can NOT be used with dates before 1/1/2000.

XMLSAFE(?..?) where $..$ is the source string. This function removes unsafe characters from a string

being used in an XML file. For example, “ are removed from the string.

CCGAURD($..$;?..?) where $..$ is the account number from the credit card and ?..? is the expiration date

date in the form MMYY. The value returned by this function is the credit card

reference number.

CCGET($..$;?..?) where $..$ is the retrieval reference number from the credit card and ?..? is optionally the

expiration date n the form MMYY. The value returned by this function is the credit card

reference number.

+ (Both treated as strings)

Field [] $..$ ##:$..$ $..$^$..$ (In Record; By Sequence; Referential Field) In Record and referential

always use the lowest record in the instance list. Normally this is

the current record only. By sequence, evaluates the string in the field

position to a final string.

System Variable {…} DATETIME{format} SEQ RECTYPE TRANCODE TRANKEY INVER

$… included FILECNT, FILEKEY, DETAILCNT,COLCNT,COLKEY,GROUPCNT,BUNDLEID,  
 BASECON

$@RECCNT#### where “####” is a zero filled record number this returns the number of records of

this type contained within the specific file. The “@” after the $ sign causes a

numeric value to be the default returned.

$+##:???:\*\*\* where ## is the output record ID, ??? is a field check (a field check consists of a field

name followed by a value in paranthesis), and \*\*\* is the field name

BASE(1..1;2..2;3..3) converts a value to a new numbering system such as base 16. The 1..1 is an expression which is the value to be converted. 2..2 is the new base to be used for example hexidecimal would have a value of 16 in this field. 3..3 contains the width of the resulting field. This conversion will be zero field with right justrification.

#### Numeric Results

Level I

String

Numeric

Parenthesis

LEFT

RIGHT

MID

ITOA

NULL

NOTNULL

STR evaluates the expression within its parameters as a string rather than a numeric

Field [] $..$ ##:$..$ $..$^$..$ (In Record; By Sequence; Referential Field) In Record and referential

always use the lowest record in the instance list. Normally this is

the current record only. By sequence, evaluates the string in the field

position to a final string.

System Variable {…} DATETIME{format} SEQ RECTYPE TRANCODE TRANKEY INVER

$… included FILECNT, FILEKEY, DETAILCNT,COLCNT,COLKEY,GROUPCNT,BUNDLEID,  
 BASECON, TRANSIZE

$@RECCNT#### where “####” is a zero filled record number this returns the number of records of

this type contained within the specific file. The “@” after the $ sign causes a

numeric value to be the default returned.

$+##:???:\*\*\* where ## is the output record ID, ??? is a field check (a field check consists of a field

name followed by a value in paranthesis), and \*\*\* is the field name

ABS

ATOI

CBC evaluates a series of expressions case-by-case from left to right. The system stops evaluation as soon

as any one of the evaluations fail (e.g., “CBC({RECTYPE}==’IE’;NOTNULL([TAX\_CODE]))”).

XLAT (e.g., “<INPUT>XLAT(1=Y;0=N)” a compare of “\*DFLT\*” sets a value if nothing matches)

DBXLAT (e.g., “<INPUT>DBXLAT($..$;#)” where $..$ is the code set and #=Field Index is used)

DBXLAT($?..?) where ?..? is a location table variable: B=Business Area, S=Source Company,

H=Charge Company, P=LOB Profit Center, C=LOB Cost Center, R=Register Number,

I=Inventory Location, O=Operating Origin, D=Department, N=Store Number,

A=Name, V=Selling Device, L=Line of Business, G=Sales Organization, F=Division,

X=Participant

DBXLAT($?..?#) where ?..? is a location table variable (P=Profit Center and C=Cost Center) and # is the

single character line of business item type as defined in the batch specification or a string

expression that evaluates to a line of business item type. The system attempts to locate

the value from the following sources in order: center defined for the specified LOB, for

the location’s LOB, and finally for the first alphabetic LOB entry.

DBXLAT($M#) retrieves the merchant ID corresponding to the card type identified by the value in #

where valid values are those specified in the definition of HLC\_MERCHANT\_CODES.

DBXLAT(~?..?) retrieves the shop number based on the value indicated by ?..? where valid values include

R=Register Number, O=Origin/Department, I=Inventory Location

DBXLAT(\*) locates the group associated with the filename passed in from the output file history.

DBXLAT(@) locates the outbound filename associated with an FTP filename from output file history.

FIND (e.g., FIND(IE;0) or FIND(IE;[ITEM\_TYPE]==’F’;0)) returns sequence code

EXISTS (e.g., “EXISTS(‘IE’)” or “EXISTS(IE;[ITEM\_TYPE]==’F’)”)

SUM (e.g., “SUM(IE;$..$)” or “SUM(IE;[ITEM\_TYPE]==’F’;$..$)” where $..$ is the calculation to sum)

The record type can be replaced by a list of types separated by colons (e.g., SUM(IE:IG;$..$)

MIN (e.g., “MIN(IE;$..$)” or “MIN(IE;[ITEM\_TYPE]==’F’;$..$)” where $..$ is the calculation)

The record type can be replaced by a list of types separated by colons (e.g., MIN(IE:IG;$..$)

MAX (e.g., “MAX(IE;$..$)” or “MAX(IE;[ITEM\_TYPE]==’F’;$..$)” where $..$ is the calculation)

The record type can be replaced by a list of types separated by colons (e.g., MAX(IE:IG;$..$)

AVG (e.g., “AVG(IE;$..$)” or “AVG(IE;[ITEM\_TYPE]==’F’;$..$)” where $..$ is the calculation)

The record type can be replaced by a list of types separated by colons (e.g., AVG(IE:IG;$..$)

PIT (e.g., “PIT($..$;&..&;X1;X2)” where $..$ is the calculation that is being spread and &..& is the field

within the current record type that is the basis of the spread. Values

X1 and X2 are optional parameters that indicate the range of sequence

codes to which this applies).

DISC (e.g., “DISC(%..%;$..$;#1;#2;#3;#4)” where %..% is an expression that results in the sequence

code of the record to be evaluated; $..$ is the label used to search the translation table for item

configuration; #1 is the record type for discount records; #2 is the field name that contains discount

amount in the transaction discount record; #3 is the name of the field with the lowest sequence value;

and #4 is the name of the field for the highest sequence value.

COUNT (e.g., “COUNT(IE)” or “COUNT(IE:[ITEM\_TYPE]==’F’)”)

INDEX (e.g., (“<SEQ>INDEX(IE;IS;IG)”) the function returns a zero based offset for the record identified

by the sequence code to the left of the operator. The offset is

calculated by examining the records in order while only

considering record’s whose types are in the provided list.

DBEXC($..$) returns true or false based on whether the key provided through the expression $..$ exists in

the exception database – status is not reviewed by this function only that it exists.

Level II (Numeric Only)

\*

/

Level III (Numeric Only)

+

-

Level IV (Mix only use numerical compare if both NUMERIC)

= =

!= or <>

<

<=

>

>=

IN (e.g., <INPUT>IN(1;2;3;4) where 1,2,3,4 are the valid values)

DBIN($..$) where $..$ is a translation SET\_CODE for lookup

DBIN($?..?) where ?..? is a location table variable and the function returns true if the field contains

data. Valid variables include: B=Business Area, S=Source Company, H=Charge

Company, R=Register Number, I=Inventory Location, O=Operating Origin, D=Department,

N=Store Number

DBIN($?..?#) where ?..? is a location table variable (P=Profit Center and C=Cost Center) and # is the single

character line of business item type as defined in the batch specification or a string

expression that evaluates to a line of business item type. The system attempts to locate the

value from the following sources in order: center defined for the specified LOB, for the

location’s LOB, and finally for the first alphabetic LOB entry.

DBXLAT($M#) retrieves the merchant ID corresponding to the card type identified by the value in #

where valid values are those specified in the definition of HLC\_MERCHANT\_CODES.

DBXLAT(~?..?) retrieves the shop number based on the value indicated by ?..? where valid values include

R=Register Number, O=Origin/Department, I=Inventory Location

DBIN(\*) returns true if the filename passed in exists in the output file history.

DBIN(@) returns true if the outbound filename associated with an FTP filename from output file history.

RRCHK($..$) returns true if the value $..$ is a valid retrieval reference number.

Level V (Numeric Only)

|| or “OR”

&& or “AND”

#### Filename Results

String

Numeric

Parenthesis

LEFT

RIGHT

MID

ITOA

CHR (e.g., “CHR(39)” will generate a string of a single quote)

LINE (e.g., LINE(2) returns third line in the file) – Used to retrieve a line out of the transfer file

LOCATE (e.g., LOCATE(0;5;XYZ) returns first line with “XYZ” in the fifth column) ZERO=Index

TOKEN (e.g., TOKEN(LINE(0);4;|) returns the fifth field on the first line of the file where fields are

Separated by the “|” character) no separator causes it to assume “,” delimited.

XLAT (e.g., “<INPUT>XLAT(1=Y;0=N)” a compare of “\*DFLT\*” sets a value if nothing matches)

DBXLAT (e.g., “<INPUT>DBXLAT($..$;#)” where $..$ is the code set and #=Field Index is used)

DBXLAT (e.g., “<INPUT>DBXLAT($..$;#)” where $..$ is the code set and #=Field Index is used)

DBXLAT($?..?) where ?..? is a location table variable: B=Business Area, S=Source Company,

H=Charge Company, P=LOB Profit Center, C=LOB Cost Center, R=Register Number,

I=Inventory Location, O=Operating Origin, D=Department, N=Store Number,

A=Name, V=Selling Device, L=Line of Business, G=Sales Organization, F=Division,

X=Participant

DBXLAT($?..?#) where ?..? is a location table variable (P=Profit Center and C=Cost Center) and # is the

single character line of business item type as defined in the batch specification or a string

expression that evaluates to a line of business item type. The system attempts to locate

the value from the following sources in order: center defined for the specified LOB, for

the location’s LOB, and finally for the first alphabetic LOB entry.

DBXLAT($M#) retrieves the merchant ID corresponding to the card type identified by the value in #

where valid values are those specified in the definition of HLC\_MERCHANT\_CODES.

DBXLAT(~?..?) retrieves the shop number based on the value indicated by ?..? where valid values include

R=Register Number, O=Origin/Department, I=Inventory Location

DBXLAT(\*) locates the group associated with the filename passed in from the output file history.

DBXLAT(@) locates the outbound filename associated with an FTP filename from output file history.

DOW($..$;#;?..?) where $..$ is a calculation that results in a base date in the form YYYYMMDD; # is the

day of week index where 0=Sunday;…;6=Saturday; and ?..? is optionally an output date

format. The format string is based on the STRFTIME standards. The default format is

YYYYMMDD. DOW can NOT be used with dates before 1/1/2000.

+ (Both treated as strings)

System Variable {…} DATETIME{format}

$… included INFILE, INPATH, BUNDLEID, HOSTCODE, CREATEDATE

#### Conditional Results

Level I

String

Numeric

Parenthesis

LEFT

RIGHT

MID

ITOA

NULL

NOTNULL

LINE (e.g., LINE(2) returns third line in the file) – Used to retrieve a line out of the transfer file

LOCATE (e.g., LOCATE(0;5;XYZ) returns first line with “XYZ” in the fifth column) ZERO=Index

TOKEN (e.g., TOKEN(LINE(0);4;|) returns the fifth field on the first line of the file where fields are

Separated by the “|” character) no separator causes it to assume “,” delimited.

XLAT (e.g., “<INPUT>XLAT(1=Y;0=N)” a compare of “\*DFLT\*” sets a value if nothing matches)

DBXLAT (e.g., “<INPUT>DBXLAT($..$;#)” where $..$ is the code set and #=Field Index is used)

DBXLAT (e.g., “<INPUT>DBXLAT($..$;#)” where $..$ is the code set and #=Field Index is used)

DBXLAT($?..?) where ?..? is a location table variable: B=Business Area, S=Source Company,

H=Charge Company, P=LOB Profit Center, C=LOB Cost Center, R=Register Number,

I=Inventory Location, O=Operating Origin, D=Department, N=Store Number,

A=Name, V=Selling Device, L=Line of Business, G=Sales Organization, F=Division,

X=Participant

DBXLAT($?..?#) where ?..? is a location table variable (P=Profit Center and C=Cost Center) and # is the

single character line of business item type as defined in the batch specification or a string

expression that evaluates to a line of business item type. The system attempts to locate

the value from the following sources in order: center defined for the specified LOB, for

the location’s LOB, and finally for the first alphabetic LOB entry.

DBXLAT($M#) retrieves the merchant ID corresponding to the card type identified by the value in #

where valid values are those specified in the definition of HLC\_MERCHANT\_CODES.

DBXLAT(~?..?) retrieves the shop number based on the value indicated by ?..? where valid values include

R=Register Number, O=Origin/Department, I=Inventory Location

DBXLAT(\*) locates the group associated with the filename passed in from the output file history.

DBXLAT(@) locates the outbound filename associated with an FTP filename from output file history.

DOW($..$;#;?..?) where $..$ is a calculation that results in a base date in the form YYYYMMDD; # is the

day of week index where 0=Sunday;…;6=Saturday; and ?..? is optionally an output date

format. The format string is based on the STRFTIME standards. The default format is

YYYYMMDD. DOW can NOT be used with dates before 1/1/2000.

+ (Both treated as strings)

System Variable {…} DATETIME{format}

$… included INFILE, INPATH, BUNDLEID, HOSTCODE, CREATEDATE

Level II (Mix only use numerical compare if both NUMERIC)

= =

!= or <>

<

<=

>

>=

IN (e.g., <INPUT>IN(1;2;3;4) where 1,2,3,4 are the valid values)

DBIN($..$) where $..$ is a translation SET\_CODE for lookup

DBIN($?..?) where ?..? is a location table variable and the function returns true if the field contains

data. Valid variables include: B=Business Area, S=Source Company, H=Charge

Company, R=Register Number, I=Inventory Location, O=Operating Origin, D=Department,

N=Store Number

DBIN($?..?#) where ?..? is a location table variable (P=Profit Center and C=Cost Center) and # is the single

character line of business item type as defined in the batch specification or a string

expression that evaluates to a line of business item type. The system attempts to locate the

value from the following sources in order: center defined for the specified LOB, for the

location’s LOB, and finally for the first alphabetic LOB entry.

DBXLAT($M#) retrieves the merchant ID corresponding to the card type identified by the value in #

where valid values are those specified in the definition of HLC\_MERCHANT\_CODES.

DBXLAT(~?..?) retrieves the shop number based on the value indicated by ?..? where valid values include

R=Register Number, O=Origin/Department, I=Inventory Location

DBIN(\*) returns true if the filename passed in exists in the output file history.

DBIN(@) returns true if the outbound filename associated with an FTP filename from output file history.

Level III (Numeric Only)

|| or “OR”

&& or “AND”

#### Input Results

String

Numeric

Parenthesis

LEFT

RIGHT

MID

ITOA

CHR (e.g., “CHR(39)” will generate a string of a single quote)

XLAT (e.g., “<INPUT>XLAT(1=Y;0=N)” a compare of “\*DFLT\*” sets a value if nothing matches)

DBXLAT (e.g., “<INPUT>DBXLAT($..$;#)” where $..$ is the code set and #=Field Index is used)

DBXLAT (e.g., “<INPUT>DBXLAT($..$;#)” where $..$ is the code set and #=Field Index is used)

DBXLAT($?..?) where ?..? is a location table variable: B=Business Area, S=Source Company,

H=Charge Company, P=LOB Profit Center, C=LOB Cost Center, R=Register Number,

I=Inventory Location, O=Operating Origin, D=Department, N=Store Number,

A=Name, V=Selling Device, L=Line of Business, G=Sales Organization, F=Division,

X=Participant

DBXLAT($?..?#) where ?..? is a location table variable (P=Profit Center and C=Cost Center) and # is the

single character line of business item type as defined in the batch specification or a string

expression that evaluates to a line of business item type. The system attempts to locate

the value from the following sources in order: center defined for the specified LOB, for

the location’s LOB, and finally for the first alphabetic LOB entry.

DBXLAT($M#) retrieves the merchant ID corresponding to the card type identified by the value in #

where valid values are those specified in the definition of HLC\_MERCHANT\_CODES.

DBXLAT(~?..?) retrieves the shop number based on the value indicated by ?..? where valid values include

R=Register Number, O=Origin/Department, I=Inventory Location

+ (Both treated as strings)

System Variable {…} DATETIME{format}

$… included INLINE, SOURCE, TRANCODE

#### Input Numeric Results

String

Numeric

Parenthesis

LEFT

RIGHT

MID

ATOI

XLAT (e.g., “<INPUT>XLAT(1=Y;0=N)” a compare of “\*DFLT\*” sets a value if nothing matches)

DBXLAT (e.g., “<INPUT>DBXLAT($..$;#)” where $..$ is the code set and #=Field Index is used)

DBXLAT (e.g., “<INPUT>DBXLAT($..$;#)” where $..$ is the code set and #=Field Index is used)

DBXLAT($?..?) where ?..? is a location table variable: B=Business Area, S=Source Company,

H=Charge Company, P=LOB Profit Center, C=LOB Cost Center, R=Register Number,

I=Inventory Location, O=Operating Origin, D=Department, N=Store Number,

A=Name, V=Selling Device, L=Line of Business, G=Sales Organization, F=Division,

X=Participant

DBXLAT($?..?#) where ?..? is a location table variable (P=Profit Center and C=Cost Center) and # is the

single character line of business item type as defined in the batch specification or a string

expression that evaluates to a line of business item type. The system attempts to locate

the value from the following sources in order: center defined for the specified LOB, for

the location’s LOB, and finally for the first alphabetic LOB entry.

DBXLAT($M#) retrieves the merchant ID corresponding to the card type identified by the value in #

where valid values are those specified in the definition of HLC\_MERCHANT\_CODES.

DBXLAT(~?..?) retrieves the shop number based on the value indicated by ?..? where valid values include

R=Register Number, O=Origin/Department, I=Inventory Location

DBXLAT(\*) locates the group associated with the filename passed in from the output file history.

DBXLAT(@) locates the outbound filename associated with an FTP filename from output file history.

+ (Both treated as strings)

System Variable {…} DATETIME{format}

$… included INLINE, SOURCE, TRANCODE

### User

The user tables are designed to control the behavior of the Web based user interface. These tables are only used within the batch process to control the automated paging and assignment of processing exceptions. Figure 21 shows the basic record structure along with the high-level organization. Linkage to exception management and the host/location configuration tables are made through the fields that begin with “RGI\_”. These fields contain a numeric value that corresponds to the desired “GROUP\_NUMBER” defined in the USR\_GROUP table. There is not a special class defined for access to these tables within the Unix environment.



Figure 21.User Table Structure

**USR\_PROFILE Data Definition**

This table contains a single row for each valid user of the system. Information in this table is used exclusively to identify users. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | ID\_NUMBER | Numeric | Contains a numeric value that uniquely identifies a user within the system. This value ideally should never be reused. Although reuse is permissible after one year. |
|  | USER\_ID | Char(12) | Contains the user ID that is used to access the system. This value should ideally match the user’s Outlook or LAN IDs. |
|  | PASSWORD | Char(32) | Contains the encrypted password required to access the Web based user interface. |
|  | FIRST\_NAME | Char(25) | Contains the first name of the user this ID represents. |
|  | LAST\_NAME | Char(40) | Contains the last name of the user this ID represents. |
|  | EMAIL\_ADDRESS | VarChar2 (64) | Contains an internet style e-mail address for delivering e-mail to this user. |
|  | ADMIN\_FLAG | Char(1) | Contains a flag indicating if the associated user is a system administrator. Normally this flag is set to “N” limiting a users access to data associated with their locations and/or hosts. If set to “Y” the user interface allows the user access to failure information for all groups and hosts defined within the system. |
|  | REVOKED\_  STATUS | Char(1) | Contains a Y/N flag indicating if this ID has been revoked. A value of “N” allows user access. |
|  | ATTEMPTS | Numeric | Indicates the number of times this ID was used unsuccessfully to access the system. |
|  | LAST\_LOGIN | Date/Time | Contains the date and time that the system was last successfully accessed by the user. |
|  | GROUP\_NUMBER | Numeric | Contains a number that identifies the group of which this user is a member. Access control, list content, and responsibility is controlled by this value through the other “USR\_” tables. |

**USR\_GROUP Data Definition**

This table contains a single row for each valid group identified within the system. Groups are used within the RTP system to identify levels of access, identify responsible users, and customize the user interface for the specific user group. Every user belongs to one and only one group. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | GROUP\_NUMBER | Numeric | Contains a numeric value that uniquely identifies a group of business or technical users. This value is used to key all related table information. |
|  | DESCRIPTION | Char(32) | Contains a text-based description of the users within the group. This is used for screens and logs to create a human readable form. |
|  | GROUP\_TYPE | Char(1) | Contains a flag that indicates if this is a group of business or technical users. A value of “B” indicates a business group while “T” is a technical group. |

**USR\_ACCESS Data Definition**

This table contains one row for each unique area and group combination. Data within this table is used to control the type of access users within a group have to the web interface to the RTP system. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | AREA | Char(8) | Contains an eight-character key that identifies the type of security that this entry affords the system. For example, if this protects exception resubmission then the value might be “EXCEPT”. The user interface will have a call to validate this areas availability for the logged in user. Valid values for this field will be defined within the user interface design documents. |
|  | GROUP\_NUMBER | Numeric | Contains a numeric value that uniquely identifies a group of business or technical users. This value is used to key all related table information. |
|  | SECURITY | Char(1) | Contains a code that indicates the type of access that the user is provided. Valid values include:   * “N”o Access * “V”iew Only Access * “U”pdate Access * “A”dministrative Access |
|  | USR\_NO\_  GRANTER | Numeric | Contains the “ID\_NUMBER” associated with the user that granted this level of access to the associated group. This value is captured for review purposes. |

**USR\_LOG Data Definition**

This table contains data about either successful or failed access to specific areas of the system. Information in this table is maintained for at least 90 days and is used to monitor the security of the system. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | ID\_NUMBER | Numeric | Contains a numeric value that uniquely identifies a user that caused this log event. This value must exist within the USR\_PROFILE table. |
|  | AREA | Char(8) | Contains an eight-character key that identifies the type of security that was either accessed or violated in this process. This value must exist within the USR\_ACCESS table. |
|  | DATETIME | Date/Time | Contains the date and time when this event occurred. |
|  | SUCCESS\_FLAG | Char(1) | Contains a flag that is set to “Y” if this record indicates successful access to this area of the system. “N” indicates that this is recording a security violation where the user attempted to access something that they did not have access to. A password error will also result in a “N” value. |
|  | IP\_ADDRESS | Char(15) | Contains the IP address from which this request originated. |
|  | MESSAGE | VarChar  (64) | Contains a free form message that the application can write to the log if desired. |

**USR\_PREFERENCE Data Definition**

This table contains one row for each unique preference and user ID number combination. Data within this table is used to maintain user based preferences and settings. The specific preference is dynamic and identified by the “name” field used within the application code. The following is a description of the fields contained within the table:

|  |  |  |  |
| --- | --- | --- | --- |
| **DB Name** | **Name** | **Type** | **Description** |
|  | ID\_NUMBER | Numeric | Contains a numeric value that uniquely identifies the user for which this specific setting value is maintained. This value must exist within the USR\_PROFILE table. |
|  | NAME | Char(8) | Contains an eight-character key that identifies the type of preference or setting that this setting value represents. Definition of these values is completely at the discretion of the user interface. |
|  | SETTING | VarChar  (64) | Contains up to 64 characters of information that represent the setting associated with this “name”. Numeric values should be translated to strings before being written to the database and must be decoded upon retrieval. |

# File Summary

## Source Code – Core Program Modules

| **Filename** | **Description** | **Used In** |
| --- | --- | --- |
| RTPINIT.cpp | Contains “main” along with initialization code for the RTPINIT executable. | RTPINIT |
| RTPINIT.h | Contains the constant and type definitions used uniquely within the program RTPINIT. | RTPINIT |
| RTPSTART.cpp | Contains the source code used to start the processes of the daemon and recover the shared resources into a known state. | RTPINIT |
| RTPCPROC.cpp | Contains the source code that the command processor process uses to execute the commands received on the FIFO. | RTPINIT |
| RTPSCREEN.cpp | Contains the source code used for the SCREEN process. This module includes the main module along with all the supporting subroutines. | RTPINIT |
| RTPLOG.cpp | Contains the source code used for the Logging process. This module includes the main module along with all the supporting subroutines. | RTPINIT |
| RTPINLOAD.cpp | Contains the source code used for loading a new Input Format version into shared memory. | RTPINIT |
| RTPDEBUG.cpp | Contains “main” along with initialization code for the RTPDEBUG executable. | RTPDEBUG |
| RTPDEBUG.h | Contains the constant and type definitions used uniquely within the program RTPDEBUG. | RTPDEBUG |
| RTPTRANS.cpp | Contains the primary module source used for the *Transaction* stage of the RTP system. | RTPBATCH |
| RTPTRANS.h | Contains the constant and function definitions used uniquely for entry into the *Transaction* stage of the RTP system. | RTPBATCH |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Source Code – Common Modules

| **Filename** | **Description** | **Used In** |
| --- | --- | --- |
| RTPGENERIC.h | Contains generic type definitions, constant definitions, and structures used throughout the RTP system. It is recommended this is included in all source modules. | All Modules |
| RTPDIRMGMT.h | Contains the class and constant definitions used for accessing a directory based on path and pattern information. | RTPINIT  RTPBATCH |
| RTPDIRMGMT.cpp | Contains the method source code used for accessing a directory based on path and pattern information. | RTPINIT  RTPBATCH |
| RTPSUPPORT.h | Contains the class and constant definitions used for accessing the support infrastructure which provides core functionality such as logging, process control, database log access, and access to shared memory segments for common data such as input format information. | All Programs |
| RTPSUPPORT.cpp | Contains the method source code used for accessing the support infrastructure which provides core functionality such as logging, process control, database log access, and access to shared memory segments for common data such as input format information. | All Programs |
| RTPNMPIPE.h | Contains the class and constant definitions used for implementing pipe communication between processes. | All Programs |
| RTPNMPIPE.cpp | Contains the method source code used for implementing pipe communication between processes. | All Programs |
| RTPMASTER.h | Contains the class and constant definitions used for accessing and creating the master shared memory segment used for control and statistic information shared by all RTP processes. | All Programs |
| RTPMASTER.cpp | Contains the method source code used for accessing and creating the master shared memory segment used for control and statistic information shared by all RTP processes. | All Programs |
| RTPINIDECODE.h | Contains the class and constant definitions used for accessing and decoding INI files throughout the RTP system. | RTPINIT |
| RTPINIDECODE.cpp | Contains the method source code for the class used to access and decode INI files through the RTP system. | RTPINIT |
| RTPCMDLINE.h | Contains the class and constant definitions used for parsing command line parameters, validating parameters, and providing text based help. | RTPINIT  RTPBATCH  RTPDEBUG  RTPEMU  RTPDTRAN |
| RTPCMDLINE.cpp | Contains the method source code for parsing command line parameters, validating parameters, and providing text based help. | RTPINIT  RTPBATCH  RTPDEBUG  RTPEMU  RTPDTRAN |
| RTPTRANSFILE.h | Contains the class and constant definitions used for accessing a standard batch transaction input file. | RTPBATCH |
| RTPTRANSFILE.cpp | Contains the method source code used for validating and accessing a standard batch transaction input file. | RTPBATCH |
| RTPLOGICALBATCH.h | Contains the class and constant definitions used for accessing transaction information in a logical batch format. | RTPBATCH |
| RTPLOGICALBATCH.cpp | Contains the method source code used for accessing transaction information in a logical batch format. | RTPBATCH |
| RTPINFORM.h | Contains the class and constant definitions used for building and accessing input format control information. | RTPINIT  RTPBATCH |
| RTPINFORM.cpp | Contains the method source code used for building and accessing input format control information. | RTPINIT  RTPBATCH |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Source Code – Data Structure Definition Modules

| **Filename** | **Description** | **Used In** |
| --- | --- | --- |
| RTPMASTERSEG.h | Contains the structures used for access to the master shared memory segment. | RTPINIT  RTPBATCH  RTPDEBUG |
| RTPCMDFIFO.h | Contains the structures and constants used for communication on the command FIFO that feeds the daemon. | RTPINIT  RTPBATCH  RTPDEBUG |
| RTPLOGFIFO.h | Contains the structures and constants used for communication with the logging process through the log FIFO. | All Programs |
| RTPTRANSBASE.h | Contains constant and structure definitions used throughout the system for analyzing basic transaction information. | RTPBATCH |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Source Code – Database Modules

| **Filename** | **Description** | **Used In** |
| --- | --- | --- |
| RTPCFGHOST.h | Contains the class and constant definitions used for accessing the host configuration information in the Oracle database. | RTPINIT  RTPDTRAN |
| RTPCFGHOST.cpp | Contains the method source code used for accessing host configuration information in the Oracle database. | RTPINIT  RTPDTRAN |
| RTPCFGLOCATION.h | Contains the class and constant definitions used for accessing the location configuration information in the Oracle database. | RTPINIT  RTPBATCH |
| RTPCFGLOCATION.cpp | Contains the method source code used for accessing location configuration information in the Oracle database. | RTPINIT  RTPBATCH |
| RTPCFGINFORM.h | Contains the class and constant definitions used for accessing input format configuration information in the Oracle database except for the ITF\_VERSION table. | RTPINIT |
| RTPCFGINFORM.cpp | Contains the method source code used for accessing input format configuration information in the Oracle database except for the ITF\_VERSION table. | RTPINIT |
| RTPCFGINFORMVER.h | Contains the class and constant definitions used for accessing input format version configuration information in the Oracle database. | RTPINIT |
| RTPCFGINFORMVER.cpp | Contains the method source code used for accessing input format version configuration information in the Oracle database. | RTPINIT |
| RTPDBLOPS.h | Contains the class and constant definitions used for accessing and updating the operational log tables. | RTPBATCH |
| RTPDBLOPS.cpp | Contains the method source code used for accessing and updating the operational log tables. | RTPBATCH |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Code Generation

| **Filename** | **Description** | **Creates** |
| --- | --- | --- |
| UXRTPINIT.mak | Contains the make script that can be used on either Linux or HP-UX to generate the *Daemon Initialization* program. | RTPINIT |
| UXRTPDEBUG.mak | Contains the make script that can be used on either Linux or HP-UX to generate the *Debugging Tool* program. | RTPDEBUG |
| UXRTPBATCH.mak | Contains the make script that can be used on either Lunux or HP-UX to generate the *Batch Transaction* processing program. | RTPBATCH |
|  |  |  |
|  |  |  |
|  |  |  |

# Error Definitions

**RTPINIT**

RTP1022E:CINIDecode Class failed to allocate during daemon start.

RTP1023E:Error installing signal handler for “%s” in the daemon start up.

RTP1024E:SLEEP function call detected an unexpected error.

RTP1025E:TIME-OUT waiting for daemon shutdown.

RTP1026E:An error occurred changing the working directory (rc=%d).

RTP1400E:Error forking the daemon task during start up (rc=%d).

RTP1401E:Error forking the “%s” process (rc=%d). Name of Process, errno

RTP1402E:Unable to install signal handler “%s” (rc=%ld) Signal ID errno

RTP1403E:Error opening the STDOUT file “%s” (rc=%ld). Filename, errno

RTP1404E:Error writing to STDOUT file (rc=%ld). Errno

RTP1405W:Error forking additional child process \”%s\” (rc=%d). name errno

RTP1406W:Error setting signal mask for process fork (rc=%d). errno

RTP1407W:Error resetting signal mask after process fork (rc=%d). errno

RTP1408W:Error “unlink”ing file – file not removed (rc=%d). errno

RTP1409E:Error getting file time information loading the directory (rc=%d) errno

RTP1410E:Error allocating shared memory segment (size=%lu; rc=%ld). Size, errno

RTP1411E:Unable to attach to input format shared segment (rc=%ld).

RTP1412E:Unable to locate the database control binary file.

RTP1413E:GETHOSTNAME failed in logging process(rc=%ld).

RTP1414E:Failure “unlinking” aged bundling archive file (rc=%ld).

RTP1415E:Error queuing DCD\_TERMINATE command – will hard fail (rc=%ld).

RTP1416W:Invalid failure action code received on child termination (Code=%d).

RTP1417E:ERROR - Process “%s” has failed – daemon will continue limited operation.

RTP1418E:Too many arguments associated with a task based process.

RTP1419E:Restart attempts exceeded restart limit for process %s.

RTP1420E:Unable to locate process control record for %s.

RTP1421E:Process RESTART failed for %s.

RTP1422W:A failure occurred adjusting the process priority (errno=%ld).

RTP1423E:Active process limit exceeded for task %s.

RTP1424E:Error opening the task timer event control file (errno=%d): %s

RTP1425E:Error seeking task timer event control file (errno=%d).

RTP1426E:Error reading task timer event control file (rc=%ld/errno=%ld).

RTP1427E:Missing CR from line end in task timer event control file - Line ignored.

RTP1428E:Error writing to task timer event control file (rc=%ld/errno=%ld).

RTP1429E:Missing or invalid parameters received on CLEAN line: %s

RTP1430E:Error writing to inbound temporary file – transfer aborted (rc=%ld/errno=%ld).

RTP1431E:Body packet received without having an open storage file.

RTP1432E:Last packet received without having an open storage file.

RTP1433E:The screen file has exceeded its anticipated storage.  
RTP1434E:Error opening business logic dump file (errno=%ld): %s

RTP1435E:Error seeking to the start of the BL dump file (errno=%ld).

RTP1436E:Invalid operational state for business logic dump (State=%ld).

RTP1437E:Error writing to business logic dump file (iRet=%ld / errno=%ld / Section=%s).

RTP1438E:Error reading from business logic dump file (iRet=%ld / errno=%ld / Section=%s).

RTP1439E:Error seeking to the %s in the BL dump file (rc=%ld / errno=%ld).

RTP1440W:Marker in business logic dump does not match (%s).

RTP1441W:Header Check for business logic dump does not match (%s). {VIFPD}

RTP1442E.Error determining the dump integrity control value (RC=%ld / errno=%ld).

RTP1443E:Unexpected exit code (%ld) from %s.

RTP1444E:An unknown error occurred during the load of input version information (iRet=%d).

RTP1445E:An unknown error occurred accessing the input format config class (iRet=%d).

RTP1446E:Unexpected error accessing the Operational Archive Database.

RTP1447E:Error accessing the control information file (errno=%ld).

RTP1448E:Error Control File class not initialized.

RTP1449E:Control File read failed (Size=%ld / rc=%ld / errno=%ld).

RTP1450E:Encryption key index out of range (0 to %lu – requested %lu).

RTP1451E:Request received to set current to an inactive encryption key (Key=%ld).

RTP1452E:Encryption control entry not identified for: %s – System terminated.

RTP1453E:Error randomizing encryption keys – No ACTIVE keys available.

RTP1454E:Potential endless loop locating the current encryption key (lBase=%ld / lCount=%ld)

RTP1455E:Control information verification failed.

RTP1456E:No changes available for writing to the control file…

RTP1457E:Unknown error with INI file initialization...

RTP1458E:Encryption key requested out of range (%s Request=%ld – Max=%ld).

RTP1459E:Control File write failed (Size=%ld / rc=%ld / errno=%ld).

RTP1460E:General Encryption failure working with %s.

RTP1461E:Select key is already either INACTIVE or in use as a special key (%c).

RTP1462E:Invalid encryption entry type selected (EKC\_ = %ld).

RTP1463E:Failure in requesting a reset of the NVTR list function.

RTP1464E:DateList class was not properly initialized.

RTP1465W:Warning an excessive number of blank lines has come into the screen file!

RTP1466E:Error %s the TASK QUEUE file (errno=%ld/%s).

RTP1467E:Error starting QUEUE’d Task (%s).

RTP1468E:Error establishing read to child 32 Bit Process.

RTP1469E:Error writing to parent talk back pipe (errno=%ld).

RTP1470E:Duplicate key information on add – entry dropped.

RTP1471E:Key entry does not exist.

RTP1472E:Error reading the Comm Manager MQ key control file (lRC=%ld / errno=%ld / Target=%ld).

RTP1473E:Comm Manager MQ Key Class not initialized.

RTP1474E:Comm Manager MQ Key out of range (ID=%ld).

RTP1475E:Comm Manager MQ Key not loaded (ID=%ld).

RTP1476E:Looped once looking for Comm Manager MQ Key – write failing!

RTP1477E:Error writing Comm Manager MQ Key file (ID=%ld / lRC=%ld / Size=%ld / errno=%ld).

RTP1478E:Error opening file for verification – OPEN ERROR (errno=%ld).

RTP1479E:Error seeking the end of file for PRODMOVE (rc=%ld / errno=%ld).

RTP1480E: Error on file stat call (errno=%ld).

RTP1481E:Error on file open (errno=%ld).

RTP1482E:Error on reading file (rc=%ld / errno=%ld).

RTP1483E:Error on writing to file (rc=%ld / errno=%ld).

RTP1484E:Error changing permissions (Mod=%ld / errno=%ld).

RTP1485E:Unexpected internal timing event!

RTP1486E:Error opening Task Dump output file (errno=%ld).

RTP1487E: Error streaming Task Dump output file (errno=%ld).

RTP1488E:Unrecognized error accessing encryption key.

RTP1489E:Key ID write requested WITHOUT valid key codes.

RTP1490E:Error writing Key ID information (lRC=%ld / lLeft=%ld / errno=%ld).

RTP1491E:Error reading Key ID information (lRC=%ld / lLeft=%ld / errno=%ld).

RTP1492E:Unexpected error loading the DB control keys.

RTP1493E:Unexpected error writing out the files under a new encryption key.

RTP1494E:Error writing out revised keys please revert the file (New Key=%ld).

RTP1950E:Unable to randomize key – looped more than maximum number of times.

RTP3030E:Error occurred loading host configuration information during initialization.

RTP3031E:Error occurred loading location configuration information during initialization.

RTP8100W:Unexpected embedded logging command detected (Code=%ld). character code

RTP8101W:Unexpected message type received by logging process (Type=%ld). Type Code

RTP8102E:Missing marker in log packet – logging process will fail.

RTP8103W:Invalid Daemon Command Code Received (Code=%ld). Code Number

RTP8104W:Unexpected response code received from the daemon (Code=%ld). code

RTP8105E:Request consisted of an input version already loaded into memory.

RTP8106E:Too many input file versions have already been loaded into memory.

RTP8107W:Daemon received an unsupported command code.

RTP8108W:Command was missing required additional information - Command canceled.

RTP8109W:Daemon is not in an appropriate state to accept this command.

RTP8110E:Unexpected database error occurred during command processing.

RTP8111E:Unexpected operating system error occurred during command processing.

RTP8112E:Background process failed to start.

RTP8113E:Internal class error received attempting to submit the command.

RTP8114E:An unexpected internal error occurred.

RTP8115E:Unexpected HOST TRANSLATOR status during initialization.

RTP8116W:Time-out detected waiting for the Translator Children to terminate.

RTP8117E:Error loading and decrypting DB connect information.

RTP8118E:Error loading PCI Compliance state code.

RTP8119E:Unexpected shutdown state – daemon will hard terminate.

RTP8120E:Unknown error in maintenance database class…

RTP8121W:Task Dump received without required response process Id.

RTP9000E:Multiple daemon action commands requested (S, T, R, and P must be individually used).

RTP9001E:Daemon is not available – Command Canceled. (ALSO used by RTPDEBUG)

RTP9002I:Daemon is terminating.

RTP9003I:Daemon has started.

RTP9004E:Invalid time range contained in Task file – Configuration will cause a failure.

RTP9005E:Unable to locate special run task in configuration – “%s”.

RTP9006E:WAITPID failed unexpectedly for special child (errno=%ld).

RTP9007E:Unepected return code from special child process (%ld) – daemon terminating.

RTP9008E:RTPQIDX automated recover exhausted start up execution attempts – daemon terminating.

RTP9009I:Task Queue was required for \”%s\”

RTP9010E:Specified CLEAN up task does not exist (Index=%ld).

RTP9011I:Forced Shutdown required from State Code %ld.

RTP9012I:Hard down initiated from controlled down with State=%ld.

RTP9013I:Term Signal forced down %s core task.

RTP9014E:Security Audit CONTROL has been manipulated – Notify SECURITY IMMEDIATELY!

RTP9015E:Passcode error deferred processing of Maintenance with ID=\”%s\”.

RTP9016E:FAILure building the source filename for %s.

RTP9017E:Timestamp on file does not match release documentation (Target=%s/Got=%s).

RTP9018E: File size on file does not match release documentation (Target=%ld/Got=%ld).

RTP9019E:File build failed – fully qualified source exceeded limit (Limit=%ld / Size=%ld).

RTP9020E:File sub-directory selection was invalid (%s).

RTP9021E:Failure in UNLINK of prod move file – please handle recovery!

RTP9022E:Failure will require MANUAL ON-CALL effort to restart the system (%s).

RTP9023E:Conflicting COMMANDS – Inbound hold was issued with the inbound shutdown command.

RTP9024E:Conflicting COMMANDS - Inbound hold was issued without a start command.

RTP9025I:Inbound processors were stopped.

RTP9026W: User twice requested inbound shutdown.

RTP9027E:Configuration compatibility issue between input flag and start mode (%s).

RTP9028E:Secondry key has not been assigned (Read=%ld / Write=%ld).

RTP9029E:Pending key change is already outstanding.

**RTPMON**

RTP1250E:Error opening monitor control file (errno=%ld).

RTP1251E:Error reading from monitor control file (errno=%ld).

RTP1252E:Error writing to monitor control file (lRC=%ld / errno=%ld).

RTP1253E:Error lock was not acquired on monitor control file.

RTP1254E:Error seeking to start of the control file (errno=%ld).

RTP1255E:Error truncating control file length (Len=%ld / errno=%ld).

RTP1256E:Monitor Control Class not initialized.

RTP1257E:Out of space for monitor control information.

RTP1258E:Host code is not the appropriate length of %ld characters.

RTP1259E:Error removing monitor control file (errno=%ld).

RTP9100E:MONITOR Attach Failed: %s

RTP9101E:Redirect File change failed to occur...

RTP9102E:Host Translator Suspension Count Changed: Curr=%d Last=%d

RTP9103E:Call to SYSTEM failed with errno=%d.

RTP9104E:Error locating the start character in the disk utilization string.

RTP9105E:WARNING Disk Usage climbed from %ld to %ld between checks.

RTP9106E:WARNING Disk Utilization has exceeded the threshold of %ld (Current=%ld).

RTP9107E:WARNING DAILY Disk Utilization has increased from %ld to %ld.

RTP9108E:Error opening the stream for command string counting (errno=%ld).

RTP9109E:ALERT unexpected growth in the FTP directory - Check Shuttle.

RTP9110E:Error – attempt to use Last Tick pointer beyond the array size (Val=%ld / Max=%ld).

RTP9111E:WARNING - the system may have a DTRAN(%s) stuck in start up.

RTP9112E:Host Code %s already exists in the control table.

RTP9113E:Host Code %s does not exist in the control table.

RTP9114E:Host Code exclusion list exceeded buffer space (Need=%ld / Limit=%ld).

RTP9115E:Error attempting to execute system command: %s

**RTPDTRAN**

RTP2100E:Invalid state code in DTRAN master loop (State=%d).

RTP2101I:DTRAN catastrophic failure is causing a shutdown.

RTP2102I:DTRAN catastrophic configuration failure is causing a shutdown.

RTP2103E:Unexpected record build failure will cause the ABNORMAL termination of DTRAN.

RTP2104E:MISMATCH in buffer sizes build BUNDLE entry (Actual=%ld Expected=%ld).

RTP2105E:Failure detected evaluating numeric expression: %s

RTP2106E:Failure detected evaluating expression(%c): %s

RTP2107E:Unexpected data type (%c) detected in field: %s

RTP2108E:Unexpected final Token type detected in evaluation (Type=%d).

RTP2109E:Unexpected VERB token within an evaluation (Type=%d).

RTP2110I:Failure Action(%c) - State=%d / Process=%d / Ret=%d

RTP2112W:Numeric evaluation resulted in a string result: %s

RTP2113E:Numeric Conversion OVERFLOW (iPos=%ld; iSize=%ld).

RTP2114E:ATOLL Conversion failed due to string content: %s

RTP2115I:Passed once through NumericEvaluation without a change.

RTP2116E:Detected potential loop in evaluation of: %s

RTP2117I:Passed once through EvaluateString without a change.

RTP2118E:Failure detected creating the Transaction Wait semaphore.

RTP2119E:Failure initializing the semaphore value for Transaction Wait.

RTP2120E:Unexpected TRIGGER MODE detected (%c).

RTP2121E:Unknown Field in CalculateKeySize: ID=%ld - Field="%s".

RTP2122E:Create Bundle File was called without needing a file open.

RTP2123E:Error creating BUILD file (errno=%d): %s

RTP2124E:The file (%s) prefill failed which will result in system termination.

RTP2125E:Error writing filler information to output build file (RC=%ld/errno=%ld).

RTP2126E:Final calculated file position does not match anticipated size for: %s

RTP2127E:Collect calculated file position does not match anticipated size for: %s

RTP2128E:Error seeking anticipated position within the BUILD file (errno=%d).

RTP2129E:Error updating BUILD file record (RC=%ld/errno=%ld).

RTP2130E:Error insufficient space allocated in planning for ORD=%s

RTP2131E:SPACE CONFLICT between planning and population pass: %s

RTP2132E:Error insufficient space allocated in planning for COLLECT.

RTP2133E:Error reading a host record into memory (lRC=%ld/errno=%ld).

RTP2134E:Unexpected error attempting to pull configuration (%ld) for SumRecord.

RTP2135E:Host file validation failed resulting in the suspension of transmission.

RTP2136E:Error seek to the end of the host output file (errno=%ld).

RTP2137E:Error reading buffer during host output file verification (errno=%ld).

RTP2138E:Error seek back to the start of the host output file (RC=%ld/errno=%ld).

RTP2139E:Error renaming the build file for FTP (errno=%d) - %s

RTP2140E:Error removing bad build file (errno=%d): %s

RTP2141E:Unknown Record ID for bundle header/trailer (ID=%ld).

RTP2142W:Unable to locate a system calculated variable (%s) – NULL assumed.

RTP2143E:Unable to link failed BUNDLE file: %s

RTP2144E:Error converting sequence number for overlay.

RTP2145W:Error locating () in system variable summation condition string \”%s\”.”

RTP2146W:Error locating FIELD in system variable summation condition string \”%s\”.”

RTP2147E:Incompatible destination and source fields in system summation variable: %s

RTP2148W:WARNING: Numeric value conversion error detected – Record Ignored (%s).

RTP2149E:The index file (%s) prefill failed which will result in system termination.

RTP2150W:Index file unlink failed on (%s) with RC=%d.

RTP2151E:Error creating BUILD INDEX file (errno=%d): %s

RTP2152E:Error writing filler information to output build index file (RC=%ld/errno=%ld).

RTP2153E:Error seeking anticipated position within the BUILD INDEX file (errno=%d).

RTP2154E:Error updating BUILD INDEX file record (RC=%ld/errno=%ld).

RTP2155E:Original BUILD unlink failed (%s) with RC=%d.

RTP2156E:New BUILD rename failed (%s) with RC=%d.

RTP2157E:Error seeking to the start of the index file (errno=%d).

RTP2158E:Error reading the index control file during COMMA translation (errno=%d).

RTP2159E:Error reading a transaction out of the build file during delimiting (RC=%ld/errno=%ld).

RTP2160E:Error writing delimited transaction to new build file (RC=%ld/errno=%ld).

RTP2161E:Delimited Temporary Buffer OVERFLOW at field: %s

RTP2162E:Error seeking to the start of the first build file (errno=%d).

RTP2163E:Error evaluating string for sequence code – NO CODE assumed(%s).

RTP2164E:Error renaming the build file for DEBUG (errno=%d) - %s

RTP2165E:Failure detected in BuildSumm (Fld=%s).

RTP2166E:DIVIDE BY ZERO error in numeric evaluation.

RTP2167E:Error opening clean file for bundle clean up (errno=%d).

RTP2168E:Error opening archive file for bundle clean up (errno=%d).

RTP2169E:Error writing to clean/archive file (RC=%d/errno=%d): %s

RTP2170I:Clean Bundle Completed: %s

RTP2171E:Error opening existing archive file for copy (errno=%d): %s

RTP2172E:Error reading the prior archive file for copy (RC=%d/errno=%d): %s

RTP2173E:Error writing to the new archive file for copy (RC=%d/errno=%d): %s

RTP2174E:System failure FINALIZING bundle clean – unable to reopen the host bundle file.

RTP2175E:ERROR renaming the original host BUNDLE file in clean process (errno=%d).

RTP2176E:ERROR renaming the original archive BUNDLE file in clean process (errno=%d).

RTP2177E:Error opening the bundle file for clean check – process terminating (errno=%d): %s

RTP2178E:Error unlinking an old clean temporary file (errno=%d): %s

RTP2179E: Error renaming clean bundling file (errno=%d): %s

RTP2180E: Error unlinking an old archive temporary file (errno=%d): %s

RTP2181E:Error renaming archive file (errno=%d): %s

RTP2182E:Unexpected failure attempting to load collection list.

RTP2183E:Unexpected failure attempting to load summary data.

RTP2184I:Passed once through EvaluateFilename without a change.

RTP2185E:Error evaluating profit/cost center item line of business value: %s

RTP2186W:Tanami Center Item LOB equation improperly used with "Filename" – Default Assumed.

RTP2187W:Conditional evaluation resulted in a string result: %s

RTP2188I: Passed once through ConditionEvaluation without a change.

RTP2189E:Error reading the base build file while removing EOR (errno=%ld).

RTP2190E:Error writing file while removing EOR (rc=%ld / errno=%ld).

RTP2191E:Error reseting file pointer for BUILD file during RemoveEOR (errno=%ld).

RTP2192E:Error getting the NVTR file size (errno=%ld):%s

RTP2193E:Error checking Host Wait Status – DTRAN is terminating.

RTP2194E:ERROR page point on host wait for DTRAN - %s.

RTP2195E:Error requesting size of NVTR based on STAT (errno=%ld).

RTP2196E:ERROR group level %ld defined without locations attached…

RTP2197E:Error in HOST FILE validation – Offset=%ld.

RTP2198E:Table name not specified for custom updates…

RTP2199E:Invalid state detected during CUSTOM insertion (State=%ld).

RTP2200E:SQL Error on custom table (code=%ld):

RTP2201E:Custom table update stuck in a loop… %s

RTP2202E:Record Type List exceeded available field size…

RTP2203E:Error writing XML Version string to build file (rc=%ld/errno=%ld).

RTP2204E:Error writing XML Label string to build file (rc=%ld/errno=%ld).

RTP2205E:Organize Key for SEQUENCE exceeded storage (Len=%ld).

RTP2206E:Unable to locate adjustment point for discount (Variance %ld).

RTP2207E:Error opening the NextScan control file (errno=%ld).

RTP2208E:Error reading NextScan control file (lRC=%ld / errno=%ld).

RTP2209E:Error reseting to the start of the NextScan file (errno=%ld).

RTP9900W:Field truncated due to insufficient record length: %s (%ld)

RTP9901W:Field DROPPED due to insufficient record length: %s (%ld)

RTP9902E:Unrecognized token start (%c) at offset %d in expression.

RTP9903W:Unable to locate field information (%s) – NULL assumed.

RTP9904E:Error evaluating within parenthesis – terminating conversion.

RTP9905E:String Function failed during command evaluation - terminating conversion.

RTP9906W:Unable to extract STRING FUNCTION parameters (%s) - NULL assumed.

RTP9907W:WARNING: Transaction failure caused it to be IGNORED. (Key=%s)

RTP9908E:Unable to complete XLAT token – Evaluation failed.

RTP9909E:XLAT token requires an input value to its left.

RTP9910E:Translate could not find a match: %s

RTP9911E:INset token requires an input value to its left.

RTP9912E:Unable to complete “In Set” token – Evaluation failed.

RTP9913W:Transaction key “%s” has been dropped from processing.

RTP9914E:Invalid, missing, or a failure was detected evaluating a system variable.

RTP9915E:Unexpected System Variable Name: %s

RTP9916E:BuildSum has attempted to operate past the end of the record (Len=%ld/Limit=%ld).

RTP9917E:Planned file length does not match anticipated file length (Plan=%lu/Actual=%lu).

RTP9918W:Build process failure detected – removing “%s”.

RTP9919E:Invalid, missing, or a failure was detected evaluating an \"existance\" check.

RTP9920W:Build Index detected - removing \"%s\".

RTP9921E:Invalid, missing, or a failure was detected evaluating a \"summation\".

RTP9922W:Conditional calculation failure in SumMinMax: %s

RTP9923W:Calculation failure in SumMinMax: %s

RTP9924W:SumMinMax received an INVALID token.

RTP9925E:Invalid, missing, or a failure was detected evaluating a \"count\".

RTP9926E:Unable to complete “Database In Set” token – Evaluation failed.

RTP9927E:DBINset token requires an input value to its left.

RTP9928E:Sequence Code FIND failed during command evaluation – terminating conversion.

RTP9929W:Conditional calculation failure in FindSequence: %s

RTP9930E:PIT base calculation failed: %s

RTP9931E:PIT failed to capture current sequence pointer: %s

RTP9932E:Unable to complete DBXLAT token – Evaluation failed.

RTP9933E:DOW-Day Of Week failed during command evaluation - terminating conversion.

RTP9934E:DOW day index was %ld which is invalid – terminating conversion.

RTP9935E:Base date string calculation failed: %s

RTP9936E:Base date string is not in the form YYYYMMDD: %s

RTP9937E:Unable to complete INDEX token – Evaluation failed.

RTP9938E:INDEX token requires an input value to its left.

RTP9939E:INDEX failed because there was not an associated instance.

RTP9940E:INDEX failed to locate the target sequence code.

RTP9941E:Sequence Code CHAINUP failed during command evaluation – terminating conversion.

RTP9942W:WARNING unable to climb a CHAINUP linkage – NULL data assumed.

RTP9943E:Invalid CHAINUP parameter(s): %s

RTP9944E:CHAINUP failed missing the associated instance definition.

RTP9945W:CHAINUP failed to locate a top level record for: %

RTP9946E:CHAINUP pass failed to locate a sequence code (%s).

RTP9947E:ITOA failed during command evaluation - terminating conversion.

RTP9948W:ITOA value calculation failure: %s

RTP9949W:Calculation failure in Case-By-Case on: %s

RTP9950E:LINE failed during command evaluation - terminating conversion.

RTP9951E:LINE value calculation failure: %s

RTP9952E:LINE value was either negative or a non-integer: %s

RTP9953E:LINE error opening input file (errno=%ld).

RTP9954E:LINE error reading the input file (errno=%ld).

RTP9955E:LOCATE failed during command evaluation - terminating conversion.

RTP9956E:LOCATE index was either negative or a non-integer: %s

RTP9957E:LOCATE index calculation failure: %s

RTP9958E:LOCATE offset was either negative or a non-integer: %s

RTP9959E:LOCATE offset calculation failure: %s

RTP9960E:LOCATE offset parameter does not exist.

RTP9961E:LOCATE comparison value evaluation failure: %s

RTP9962E:LOCATE comparison parameter does not exist.

RTP9963E:TOKEN failed during command evaluation - terminating conversion.

RTP9964E:TOKEN failed evaluation of base string equation: “%s”

RTP9965E:TOKEN index was either negative or a non-integer: %s

RTP9966E:TOKEN index calculation failure: %s

RTP9967E:TOKEN index parameter does not exist.

RTP9968W:TOKEN field does not exist within the input: %s – NULL assumed.

RTP9969W:TOKEN field search hit the string end without a result: %s - NULL assumed.

RTP9970E:CHR failed during command evaluation - terminating conversion.

RTP9971E:CHR failed evaluation of base string equation: "%s"

RTP9972E:PIT based low range calculation failed: %s

RTP9973E:PIT based high range calculation failed: %s

RTP9974E:Error retrieving current record type for PIT calculation: %s

RTP9975E:Error no instance provided on a PIT – calculation failed.

RTP9976E:Error instance(%s) not found in the transaction - calculation failed.

RTP9977E:Error evaluating discount calculation – calculation failed.

RTP9978E:DISC target sequence calculation failed: %s

RTP9979E:DISC missing in-line parameter (%s) – calculation failed.

RTP9980E:DISC missing Transaction Discount record type –OR- amount field name.

RTP9981E:DISC missing configuration parameter (%s) on line %ld - calculation failed.

RTP9982E:DISC could not locate field %s in record type %s – calculation failed.

RTP9983E:DISC could not locate a discount record sequence code (%s).

RTP9984E:DISC lost configuration for record type during Item Discount (%s).

RTP9985W:DISC could not locate field %s in record type %s – Default assumed.

RTP9986E:YEAR failed during command evaluation – calculation terminated.

RTP9987E:YEAR failed evaluation of base string equation: "%s"

RTP9988W:Calculation failure in IfElse on: %s

RTP9989E:Credit Card Gaurd (CCGAURD) failed during command evaluation - terminating conversion.

RTP9990E:Missing TimeShift number of days as a parameter – calculation failed.

RTP9991E:Calculate Total Size is in conflict with Host Delivery Mode… Translator Suspended.

RTP9992E:Size of base conversion is overflowing the available space – S/W error (Size=%ld).

RTP9993W:Unable to evaluate EXT parameters (%s) - NULL assumed.

RTP9994W: Unable to locate field from EXT (%s) - NULL assumed.

RTP9995W: EXT extract exceeded storage capacity (iLen=%ld / Max=%ld) - NULL assumed.

RTP9996W:EXT extract exceeded ORIG buffer (iLen=%ld / iOff=%ld / Max=%ld) - NULL assumed.

**RTPBATCH/RTPUBATCH**

RTP1600E:Error installing signal handler for “%s” in the batch start up.

RTP1601E:Unexpected return code from the exception class (rc=%ld).

RTP1603E:RTPTRANS timed-out waiting for shared Memory to go OPERATIONAL.

RTP1604E:Field storage exceeded during encryption – operation aborted.

RTP1605E:Error clearing transaction storage – operation aborted.

RTP1606E:Unknown error while locating Exception Management flag (Loc=%s).

RTP1607E:CHARGE\_DESC field not large enough for TXN\_REF.

RTP1608E:Unexpected UNLOAD configuration error.

RTP1609E:Undefined error accessing field data from dynamic query results.

RTP9200E:Missing batch input file pattern detected – processing failed.

RTP9201E:Transaction Submission Failed (rc=%ld). ID

RTP9202W:Base sequence missing from input transaction (Len=%ld).

RTP9203W:Extended sequence missing from input transaction (Len=%ld).

RTP9204W:Buffer size does not match base sequence information (Buff=%ld – Base=%lu).

RTP9205E:Transaction key could not be detected for an ORPHAN.  
RTP9206E:Missing or invalid source system code – conversion terminating.

RTP9207E:Duplicate Logical Batch ID received (%s) – File dropped.

RTP9208E:ARCHIVE\_DIRECTORY has not been established – batch will terminate.

RTP9209E:No data generated by unload process (%s).

RTP9210E: Date is out of range - processing failed.

**RTPTRANS**

RTP2000E:Error unblocking the signal handlers for the DTRAN instance (rc=%d).

**RTPTLOAD**

RTP2200E:Error installing signal handler for “%s” in start up.

RTP2201E:Error opening the input file (rc=%d).

RTP2202E:Error reading data from the translation load file (rc=%d).

RTP9950E:Missing translation set code name - processing failed.

RTP9951E:The “P” and “X” command line options are mutually exclusive – Command Canceled.

**RTPSUPP/RTPSUPIF**

RTP1850E:MQCONN connection failure (CompCode=%d/Reason=%d).

RTP1851E:MQOPEN queue open failed (CompCode=%ld/Reason=%ld).

RTP1852E:MQGET detected an unexpected failure (CompCode=%ld/Reason=%ld).

RTP1853E:MQPUT detected an unexpected failure (CompCode=%ld/Reason=%ld).

RTP1854E:Transaction Process failed because a required class was not initialized.

RTP1855E:FAILURE detected on reset of the MQ Trigger (CompCode=%ld/Reason=%ld).

RTP1856E:Unable to establish INIPATH environment variable for the application.

RTP1857E:Unable to locate the transaction key within the exception database.

RTP1858E:Unexpected transaction analysis error retrieving an exception transaction.

RTP1859E:MQGET Timed-Out waiting for a response.

RTP1860E:Error creating pipe handle pair (%s). errno=%ld

RTP1861E:Error writing to response pipe for the client (rc=%ld/errno=%ld).

RTP1862E:Error reading header from the inbound pipe (rc=%ld/errno=%ld).

RTP1863E:Error reading data from the inbound pipe (rc=%ld/errno=%ld).

RTP1864E:Invalid %s Version ID received (%s).

RTP1865E:Batch header for file transfer did not match expected size (Got=%ld / Trgt=%ld).

RTP1866E:Invalid control marker received within the request packet (%s).

RTP1867E:Invalid request received by the logging process (%ld).

RTP1868E:Unexpected error reloading the transaction buffer.

RTP1869E:Unexpected error building the corresponding input buffer.

RTP1870E:Unexpected database error reseting the statistics cursor.

RTP1871E:Index out of range for SUPIF retrieval.

RTP1872E:Retrieval must be made through IDX=0 call.

RTP1873E:Error occurred locate data marker in string (%s).

RTP1874E:Unexpected database error retrieving reference information.

RTP1875E:Unexpected error sending anc RTPINIT command transaction.

RTP1876E: Unexpected database error setting host password.

RTP1877E: Unexpected error retrieving monitor control information.

RTP1878E:Unexpected error updating monitor control information.

RTP1879E:Packet received on an unexpected connection (Type=%ld).

RTP1880E:Unexpected error on opening of CCS database.

RTP1881E:Unexpected error loading the Security Control file.

RTP1882E:No Routing Entry Found…

RTP1883E:Unable to locate MQ Encryption key.

RTP1884E:Unexpected error in PROM class.

RTP1885E:Invalid passcode/target date combination.

RTP9980E:Invalid MQ Series Version detected in the INI file (%s).

RTP9981W:Default Queue Manager Version assumed.

RTP9982E:Invalid packet received on Support Interface Queue (%ld) – Packet Dumped and Ignored.

RTP9983W:Default response TTL established at 30 seconds.

RTP9984E:Improper packet length received on the logging queue (Size=%ld).

RTP9985E:Buffer size (%ld) received was less than expected (%ld).

RTP9986E:Invalid command requested for a production environment.

RTP9987E:Query and/or interface not properly initialized.

RTP9988E:Requested response entry is out of range (Idx=%ld / Max=%ld).

RTP9989W:Missing second parameter for Passcode Verification.

RAP1500E:Failure loading instance reference matrix (Idx=%ld / Lim=%ld).

RAP1501E:Duplicate header record detected in matrix build.

RAP8000E:Invalid transaction command code detected.

RAP8001E:Insufficient information to accept a log record (Size=%d).

RAP8002E:Transaction processor rejected the transaction (RC=%ld).

RAP8003E:Support Interface class was not properly initialized.

RAP8004E:Instance index out of range (%d - %d) – request failed.

RAP8005E:Field index out of range (%d - %d) for Instance %s – request failed.

RAP8006E:Invalid User ID/Password combination – command cancelled.

RAP8007E:Unexpected internal software error detected in Get Login.

RAP8008E:Field Name (%s) does not exist within the record index %ld.

RAP8009E:Field data type (%c) does not have an appropriate conversion routine.

RAP8010E:Date field does not contain eight or six digits as required.

RAP8011E:Change flag not enabled for retrieved transaction.

**RTPMQIF.dll**

RTP1930E:Connection handle not properly initialized.

RTP1931E:Invalid Queue ID specified on connect (%ld).

**RTPMQSRV/RTPMQINB/RTPQ/RTPCM/RTPCMTCP/RTPCMGM/RTPMQ32IF/RTPKEYSRV/RTPKEYIF**

RTP1800E:Error writing to server application input pipe (RC=%ld/errno=%ld).

RTP1801E:Error reading header from server application (RC=%ld/errno=%ld).

RTP1802E:Error reading %ld Byte Response from server application (RC=%ld/errno=%ld).

RTP1803E:Error committing an MQ Unit of Work (Comp=%ld/Reason=%ld).

RTP1804E:Error rolling back an MQ Unit of Work (Comp=%ld/Reason=%ld).

RTP1805E:Unexpected packet received on the BATCH inbound queue (Type=%ld).

RTP1806E:Unexpected error received removing the temporary transfer file (errno=%ld).

RTP1807W:Debug rename failed which resulted in the loss of debugging data.

RTP1808E:Error renaming failed partial transfer file for debugging (errno=%ld).

RTP1809E:Header received prior to receipt of a “last packet” (Type=%ld).

RTP1810E:Invalid header packet received for inbound batch (Length=%ld).

RTP1811E:Body packet received without corresponding header record – File dropped.

RTP1812E:LAST packet received without corresponding header record – File dropped.

RTP1813E:Error opening batch inbound temporary file (errno=%ld).

RTP1814E:Invalid format code received on inbound file (%d) – File being dumped…

RTP1815E:Error renaming inbound file to destination file/path (errno=%ld).

RTP1816E:Error sending task start to RTPINIT (iRC=%ld/Ext=%ld).

RTP1817E:Error opening client file for queue transmission (errno=%ld).

RTP1818E:ERROR renaming the sent file after send (errno=%d).

RTP1819E:Error reading the input file in RTPQ (errno=%ld).

RTP1820E:Error placing BATCH queue packet on the queue (CompCode=%ld/Reason=%ld).

RTP1821I:No input file detected for RTPQ request (%s).

RTP1822E:Batch Queue Connection not initialized.

RTP1823E:Error extracting the file size from the file system (errno=%ld).

RTP1824W:Error finding the first directory match on NT (LastError=%ld).

RTP1825W:Error scanning the directory on NT (LastError=%ld).

RTP1826I:RTPQ(NT) Requested transmission of file: %s

RTP1827E:Error generating termination signal (Task=%s / pid=%ld): errno=%ld

RTP1828E:Unknown child process (pid=%ld) terminated with exit code %ld.

RTP1829E:Unknown error access configuration – process will fail.

RTP1830E:Unexpected Time EVENT CODE received – Code=%ld.

RTP1831E:ID already exists – entry canceled.

RTP1832E:SOCKET command failed (errno=%ld).

RTP1833E:Address resolution failed – socket connectivity was not established.

RTP1834E:Unexpected error received from SocketSet class.

RTP1835E:Assigned connection ID is out of range (ID=%ld).

RTP1836E:Selected connection ID does not exist (ID=%ld).

RTP1837E:Selected connection ID does not support writes (ID=%ld / Type=%ld).

RTP1838E:Termination Signal received during wait event.

RTP1839E:Error deleting closed socket connection (ID=%ld).

RTP1840E:Error reading from socket (Hnd=%ld) with errno=%ld.

RTP1841E:Secure communication is not supported on a RAW channel (ID=%ld).

RTP1842E:Packet writes are not supported on a RAW channel (ID=%ld).

RTP1843E:RAW %s requested on a packet pipe – request rejected (ID=%ld).

RTP1844E:PACKET %s requested on a raw pipe – request rejected (ID=%ld).

RTP1845E:No data available on requested connection (ID=%ld).

RTP1846E:Connection broken or dead without connect.

RTP1847E:Unexpected internal software error...

RTP1848E:Name not assigned for this connection source (ID=%ld)

RTP1849E:Unexpected EVENT code received from a socket set (Code=%ld).

RTP1850E:Error sending command to RTPCM – Command Processor (%s).

RTP1851E:Duplicate key used in linked list – insertion failed (ID=%ld).

RTP1852E:Connection has already been registered to group “%s”.

RTP1853E:Grouping Label missing from server registration.

RTP1854E:End of connection list.

RTP1855E:Duplicate PORT assignment detected – connection rejected (%s/Port=%ld).

RTP1856E:CM Interface class not properly initialized.

RTP1857E:CM Interface already has a request pending – please try again.

RTP1858E:Unexpected event received during internal processing (EVC=%ld).

RTP1859E:CM Interface does not have an outstanding CMR request response available.

RTP1860E:Unexpected CM error – no additional information available (%s).

RTP1861E:Error retrieving the system assigned port number (errno=%ld).

RTP1862E:Failed to reconnect to CM Router – application failing.

RTP1863E:Reply received from CMR after unknown command (LastCmd=%ld).

RTP1864E:Bad return code on %s caused the connection to collapse (Code=%ld).

RTP1865E:Nothing found.

RTP1866E:Invalid/Unexpected Event Code received on Connection.

RTP1867E:UNEXPECTED error occurred in the GetEvent function.

RTP1868E:You can not force the CMR connection closed – request ignored.

RTP1869E:No service ports available for the requested service unit.

RTP1870E:Unexpected error connecting to the server – command failed.

RTP1871E:Exceeded number of credit cards supported in a single request.

RTP1872E:Incomplete result returned from the server – data integrity compromised (%ld).

RTP1873E:Cross reference request received with insufficient length @%s – Length=%ld.

RTP1874E:Unexpected error in generating TXN Reference number…

RTP1875E:AF2 was not long enough to contain an account number (Len=%ld).

RTP1876I:Recycling Gold connection…

RTP1877E:Invalid GOLD LIST internal ID received – connection failed…

RTP1878E:Unexpected message type seen from Gold (%s).

RTP1879E:Invalid and/or retired TLF format received (%c).

RTP1880E:Unexpected encryption key detected on TLF interface (Key=%ld).

RTP1881E:Unexpected error occurred retrieving Account Reference Number.

RTP1882E:Bad packet received from socket connection (%s).

RTP1883E:Unexpected return code from SUPP interface (RC=%ld).

RTP1884E:Submitted application packet could not be routed – packet dropped.

RTP1885E:Unexpected error in the CCS class…

RTP1886E:Unexpected error occurred setting time-out in the CMINTERFACE class.

RTP1887W:Duplicate insert into the retrieval reference number table (%s).

RTP1888E:Unknown error sending logging data.

RTP1889W:Packet received with insufficient length (%ld – Expected %ld).

RTP1890E:Invalid packet code received – Code=%ld.

RTP1891E:Error reading from MQ32IF interchange pipe (IN)(lRC=%ld / errno=%ld).

RTP1892E:End Of File received on Queue – Interface will terminate.

RTP1893E:Unknown error received opening the queue.

RTP1894E:GETMSG Packet received when connection was not READY.

RTP1895E:GETMSG did not send on sufficient message length.

RTP1896E:Communication Manager class not properly initialized.

RTP1897E:Retrieval Packet size NOT divisable by 8 (Len=%ld) – encryption failure.

RTP1898E:Subsitution requested for unknown value (%ld).

RTP1899E:Failed to get the host name from the system (%ld).

RTP1900E:Unknown error talking to the TCPIF2 class.

RTP1901E:Unable to attach to an CMR port – system is not active.

RTP1902E:Error forking task on RESTART – errno=%ld.

RTP1903E: Unable to attach to at least one CMR port.

RTP1904E:Unknown error attempt to close a connection (ID=%ld).

RTP1905E:Error adding pending listener port.

RTP1906E:Insufficient command data for PortAdd - command ignored.

RTP1907E:Port number OUT OF RANGE - Command ignored.

RTP1908E:Duplicate request to activate a port (%s).

RTP1909E:Error opening file for transfer (errno=%ld):%s

RTP1910E:Error opening pipe to push file content to PGP (errno=%ld).

RTP1911E:Error forking PGP child process for encryption (errno=%ld).

RTP1912E:Error on “waitpid” for PGP process (pid=%ld / errno=%ld).

RTP1913E:Error reading the source file (errno=%ld).

RTP1914E:Error writing to the PGP pipe (errno=%ld).

RTP1915E:Error creating the PGP output file (errno=%ld).

RTP1916E:Error unlinking temporary PGP file (errno=%ld).

RTP1917E:Connection not in appropriate state – State=%ld.

RTP1918E:Configuration error generating legacy encryption key.

RTP1919E:Unable to access the key service for %c - %ld.

RTP1920E:Server Error (%s) rc=%ld.

RTP1921E:Unexpected error in connection process.

RTP1922E:Error writing to key server socket (lRC=%ld / errno=%ld).

RTP1923E:Error reading from key server socket (lRC=%ld / errno=%ld).

RTP1924E:Error parsing key server buffer (cStatus=%c).

RTP1925E:Invalid key length - lLen=%ld.

RTP1926E:Error retrieving key from the key server.

RTP1927E:Error converting text key to hexadecimal.

RTP1928E:Invalid offset in server search (Index=%ld / Count=%ld).

RTP9720E:Invalid operational mode selected.

RTP9721E:Windows Socket Initialization Failed.

RTP9722E:TCP Interface Class was not properly initialized.

RTP9723E:Connection is already operational - Connect request ignored.

RTP9724E:Invalid connection parameters provided for CMR connection (%s / %ld).

RTP9725E:Connect is already active – second attempt refused.

RTP9726E:No active connection to disconnect.

RTP9727E:SOCKET call failed with RC=%ld.

RTP9728E:CONNECT call failed with RC=%ld.

RTP9729E:Unable to make contact with Communication Manager.

RTP9730E:Communication Dialog creation failed.

RTP9731E:Error creating dialog signal event (rc=%ld).

RTP9732E:Event signal from dialog window failed (rc=%ld).

RTP9733E:Modal wait on the communication dialog failed (rc=%ld).

RTP9734E:Error WSAAsyncSelect failed with last error code = %ld.

RTP9735E:Error receiving data from the CMR socket(%s) (rc=%ld).

RTP9736E:Error retrieving CMR connection structure.

RTP9737E:Time-out attempting %s.

RTP9738E:Unexpected error occurred during decryption process.

RTP9739E:No matching interface found under Communication Manager.

RTP9740E:Error code received from server process (Code=%ld).

RTP9741E:Unable to get contact information from CMR (%s).

RTP9742E:Service response did not contain a reply code (Len=%ld).

RTP9743E:Transaction buffer is already in use – update dropped.

RTP9744E:Invalid connection ID selected.

RTP9745E:Connection already established – region can not change.

RTP9746E:Invalid region specified – region must be P, D, or 1-4.

RTP9747E:Invalid port number received on configuration load.

RTP9748E:Task start data exceeded line limit (Len=%ld / Max=%ld).

RTP9749E:WARNING CM Inputs are still suppressed.

RTP9750E:Error analyzing the provide interface identifier.

RTP9751E:Key Interface was not initialized.

RTP9752E:Invalid command code received (%ld).

RTP9753E:Unable to locate working key server for index %ld.

RTP9754E:Invalid encryption key specified for key server channel (%ld).

**RTPSHUTL**

RTP1775E:Error opening control pipe for file distribution main thread (errno=%ld).

RTP1776E:Error sending signal to shuttle process (errno=%ld).

RTP1777E:Error reading from control pipe (Left=%ld/errno=%ld).

RTP1778E:Error setting USR1 signal for shuttle during restart (errno=%ld).

RTP1779E:Error opening single thread lock file (errno=%ld).

RTP1780E:Unexpected error locking singularity file (errno=%ld).

RTP1781I:Shuttle program is already running – Lock file unavailable.

RTP1782E:Error releasing the singularity lock on the shuttle program (errno=%ld).

RTP1783E:Internal software failure is causing a system failure.

RTP1784E:Error renaming transfer file after processing (errno=%ld).

RTP1785E:Error executing “SYSTEM” call to initiate FTP transfer (errno=%ld).

RTP1786E:Error renaming transfer file into a TIP file for transfer (errno=%ld).

RTP1787E:Unexpected error reading the inbound STDOUT fifo.

RTP1788E:Unable to locate transfer confirmation from FTP application.

RTP1789E:Error opening the output stream for capturing FTP output (errno=%ld).

RTP1790E:Error unlinking the FTP output file (errno=%ld).

RTP1791E:FGETS failed reading information out of the STDOUT file (errno=%ld).

RTP1792E:Error setting USR1 signal for next process check (errno=%ld).

RTP1793E:SHUTTLE THREAD(%s) Failed - File transfer may have aborted (Exit=%d).

RTP1794E:TIP files detected during restart – Please verify transfers and adjust accordingly (Cnt=%ld).

RTP1795E:Shuttle thread has exceeded its allotted time by %ld seconds.

RTP1796E:Error opening SOCKET for TCP/IP transfer (errno=%ld).

RTP1797E:Error CONNECTing to the port for TCP/IP transfer (errno=%ld).

RTP1798E: Error opening source file for TCP/IP transfer (errno=%ld).

RTP1799E:Unexpected error reading data from input file for TCP/IP transfer (errno=%ld).

RTP1800E:Unexpected error writing to the TCP/IP socket (rc=%ld / errno=%ld).

RTP1801E:Get host by name failed looking for %s (errno=%ld).

RTP1802E:Overflow populating record size for socket transfer.

RTP1803E:Error opening outbound file to count lines (errno=$ld).

RTP8990W:Unexpected command packet on internal FDS pipe (%ld).

RTP8991E:Shuttle process ID not available – Signal dropped.

RTP8992E:Shuttle for host %s has been suspended due to a lack of configuration.

RTP8993E:Transfer working directory matches the archive directory – initialization failed.

RTP8994E:Invalid host transfer type retrieved for %s – Code: %c

RTP8995I:Warning file generated for an inactive transport (%s).

RTP8996W:Warning host file did not have a host code for initialization.

RTP8997E:Failure detected generating a host delivery filename (bEvalErr=%ld).

RTP8998E:Failure detected evaluating a host delivery filename condition (bEvalErr=%ld).

RTP8999E:MANUAL Intervention required to handle failed DEMAND Task (%s).

**RTPDEBUG/RTPAMEM/RTPRENG/RTPRENGQ/RTPMAIL/RTPKEYSRV**

RTP1450E:Unable to allocate storage for the command list (Size=%ld). Storage Size

RTP1451E:Error getting command line from the console (RC=%d). errno

RTP1452E:Daemon did not achieve shutdown stage within designated time interval (Level=%u).

RTP1453E:Unable to terminate daemon which is required to update BUNDLE offset.

RTP1454E:Error reading from the source file (ferror=%d).

RTP1455E:Error signaling a process %ld – errno=%ld

RTP1456E:Unexpected error accessing the reporting engine database tables.

RTP1457E:REQ Database Access class did not initialize.

RTP1458E:Error opening the temporary reporting file (errno=%ld).

RTP1459E:Error unlinking the temporary file to the system (errno=%ld).

RTP1460E:Error writing to the output file (errno=%ld).

RTP1461E:Error seeking in the output file (errno=%ld).

RTP1462E:Error reading from the output file (errno=%ld).

RTP1463E:FAULT inserting intp the except report list.

RTP1464W:Message removal request received without file information.

RTP1465E:Non-reporting child process failed unexpectedly (TID=%ld / ExitCode=%ld).

RTP1466E:Error locating the process within the task list (TID=%ld / ExitCode=%ld).

RTP1467E:Unexpected exit code returned from child process (PID=%ld / ExitCode=%ld).

RTP1468E:Error writing to primary process command pipe (lRC=%ld / errno=%ld).

RTP1469E:Unexpected path request type (%s).

RTP1470E:Error retrieving %s path from NVTR management.

RTP1471E:SIU Database Access class did not initialize.

RTP1472E:Failed opening %s mail file (errno=%ld).

RTP1473E:Error getting a line from the source file (Line=%ld / errno=%ld).

RTP1474E:Error writing a line to the working file (Line=%ld / errno=%ld).

RTP1475E:Error on system command to invoke mail service (Code=%ld).

RTP6750E:SQL error encountered updating the REQ tables (sqlcode=%ld):%s

RTP6751E:SQL error encountered querying the REQ tables (sqlcode=%ld):%s

RTP6752E: SQL Issue List Get Max Sequence failed[%d]:%s

RTP6753E:SQL error updating the REQ tables (sqlcode=%ld):%s

RTP6754E:Unexpected SQL error accessing the REQ tables.%s

RTP6755E:SQL error encountered querying the CCS tables (%s - sqlcode=%ld):

RTP6756E:SQL error updating the location-reference table (sqlcode=%ld):

RTP6757E:MIN and MAX values from the reference master do not match (Min=%s / Max=%s).

RTP6758E:Error accessing the REQ\_ACTIVELIST for new item (Bad State = %ld).

RTP6759E:Error updating REQ\_ACTIVELIST (sqlcode=%ld):%s

RTP6760E:Error updating REQ\_RESPONSE\_LIST (sqlcode=%ld):%s

RTP6761E:SQL error encountered inserting into CCS\_TXNREF (sqlcode=%ld):%s

RTP6762E:SQL error querying the SIU\_CODES (sqlcode=%ld):%s

RTP6763E:SQL error fetching from cursor OBW\_LIST\_CURSOR (sqlca.sqlcode=%ld):%s

RTP9050E:Tool started in command-line mode without a list of interactive commands.

RTP9051E:Error extracting "command" value from the command line (Idx=%d). Index

RTP9052W:Invalid main menu command detected.

RTP9053W:Invalid or missing “Run” argument detected (%s) Argument

RTP9054W:Invalid or missing “Display” argument detected (%s) Argument

RTP9055W:Invalid or missing “Set” argument detected (%s) Argument

RTP9056W:Missing assignment value from the set command – Command canceled.

RTP9057E:Error opening memory tracking input file (errno=%d): %s

RTP9058W:Invalid reporting code requested (%s).

RTP9059E:Exception Aging Line Item Expression is missing from configuration.

RTP9060W:Maximum active reporting thread limit has been reached.

RTP9061E:Command MUST be run in an interactive mode – it CAN NOT be used in batch.

RTP9062E:User ID/Password combination does not match existing values – please verify case…

RTP9063E:Required entry not received – command canceled….

RTP9064E:New ID and/or Password is required for this action – command canceled.

RTP9065E:Verification of entry failed – command canceled.

RTP9066E:IOCTRL failure occurred during ECHO control (%s / errno=%ld).

RTP9067E:ERROR Multiple Encryption Keys are in use for reference numbers.

RTP9068E:Unknown error initializing the reference database.

RTP9069E:ERROR reference database needs refreshed to restart the system(DB=%ld / CTRL=%ld).

RTP9070E:ERROR exceeded buffer storage for decryption (lOut=%ld / lSize=%ld).

RTP9071E:State ERROR updating the REF/LOC table (iState=%ld).

RTP9072E:Argument count for NVTR Management Exceeded (Limit=9) – Excess IGNORED.

RTP9073E:Missing/Invalid reporting code – a valid command code is required to submit a report.

RTP9074E:Either a user ID or group name must be provided for report delivery.

RTP9075E:User ID and group name specified - the report queue program will only handle one or the other.

RTP9076E:Report priority must be I, H, M, L, or D.

RTP9077E:Error determining the requesting user key for a report request (%s).

RTP9078E:Reporting format was not selected for Ticket Retrieval Report (Code=%ld).

RTP9079E:Unknown error accessing the reporting engine configuration file.

RTP9080E:Unload commands can not be executed from the command line.

RTP9081E:Unknown error interrogating the Monitor Control File.

RTP9082E:Command MUST be called interactively – Command Canceled.

RTP9900I:Exporting File(TLTS):%s

RTP9901E:WARNING Host Monitoring Suspended for %s

RTP9902E:Missing required parameter for Transaction Report – Report aborted.

RTP9903E:Communication Manager interface verification MUST be run on-line.

RTP9904E:WARNING Host Bundling Suspended for %s

RTP9905E:DTRAN Archive File Unloaded:%s

RTP9906E:User ID/Password combination is invalid.  
RTP9907E:Login rejected - please log on the web-site for more details.

RTP9908E:Key randomize exceeded safety net – change aborted.

RTP9909E:Send Mail class was not properly initialized.

RTP9910E:Invalid audit group identifier provided – report canceled(%s).

RTP9911E: Invalid date requested - report canceled(%s).

RTP9912E:No listener ports defined for the key server.

**RTPGSA/RTPTLTS/RTPWRAP/RTPEDS/RTPEXCEPT/RTPEXFIX/RTPSQL/RTPQIDX/ RTPCCVT/ RTPESET/RTPSECURE**

RTP1050E:Error opening EDS input file (rc=%d).

RTP1051E:Error renaming the processed input file during EDS conversion (rc=%d).

RTP1052E:Error removing output file after EDS processing failure (rc=%d).

RTP1053E:Error renaming the final output file to its destination name (rc=%d).

RTP1054E:Unexpected error locking an EDS input file (rc=%d).

RTP1055E:Error removing temporary output file after EDS processing failure (rc=%d).

RTP1056E:Error creating the temporary build file for EDS conversion (rc=%ld).

RTP1057E:Error reading input buffer for EDS conversion (rc=%ld / errno=%ld).

RTP1058E:Unexpected error opening the result file (errno=%ld).

RTP1059E:Missing DS record at the beginning of the input file – process canceled.

RTP1060E:Error writing to the output file (rc=%ld / errno=%ld).

RTP1061E:Partial EDS transaction detected at the end of the input file.

RTP1062E:Mixed register deposit detected (First=%s / Second=%s).

RTP1063E:Unable to locate register number for shop translation.

RTP1064E:Unexpected database error from lease class – Conversion terminated.

RTP1065E:Lease release failed which could impact data integrity.

RTP1066E:SQL ERROR detected updating leasing information (sqlcode=%ld):%s

RTP1067E:Unexpected error opening the %s conversion class.

RTP1068E:FATAL error processing conversion - conversion terminated.

RTP1069E:Error seeking to start of input file before merge (errno=%ld).

RTP1070E: Error reading from source for file merge (errno=%ld).

RTP1071E: Error writing to master output for file for merge (%ld of %ld / errno=%ld).

RTP1072E:Error creating a build control file for conversion (errno=%ld).

RTP1073E:Transaction fragment detected in the conversion process (lOff=%ld / lBuff=%ld).

RTP1074E:Transaction missing a business control date (lOff=%ld / lBuff=%ld).

RTP1075E: Invalid transaction length detected in the conversion process (lOff=%ld / ulSize=%lu).

RTP1076E:Transaction write to build file failed (lRC=%ld / errno=%ld).

RTP1077E:Error opening conversion source file (errno=%ld).

RTP1078E:WRAP program INI special field type code is invalid (Source=%s / Type=%s).

RTP1079W:Conversion program did not produce any results – conversion canceled.

RTP1080E:Error WRAP conversion class was not initialized.

RTP1081E:Error reading from input file (errno=%ld).

RTP1082E:Error loading GLOBAL VARIABLE for evaluation (%s).

RTP1083E:Error evaluating %s field for WRAP evaluation.

RTP1084E:Missing or invalid parameter in Terminal ID automatic assignment (%s).

RTP1085E:Terminal ID automatic assignment parameter is out of range (%s - %ld to %ld).

RTP1086E:Error renaming input file to include date (errno=%ld).

RTP1087E:Unexpected error opening rename file for WRAP date change (errno=%ld).

RTP1088E:Ready List CURSOR open failed[%d]:%s

RTP1089E:Ready List CURSOR declaration failed[%d]:%s

RTP1090E:Ready List CURSOR fetch failed[%d]:%s

RTP1091E:Error updating status information [%d]:%s

RTP1092E:Error updating instance status information [%d]:%s

RTP1093E:Left over deposit detail or DST records in an EDS source file.

RTP1094E:Unable to locate the DST corresponding to a set of DS records. Deposit file dropped.

RTP1095E:Configuration error retrieving User ID and Password for SQL Plus.

RTP1096E:SQL Plus may be stuck – please review the progress of process.

RTP1097E:Error setting header MEAL TYPE and COURSE CODE for food item.

RTP1098E:Unknown error encountered accessing the RTPSUPPORT.INI file.

RTP1099E:Error opening the index control file for processing (rc=%ld)

RTP1100E:Error seeking to the start of the control file (Ret=%ld / errno=%ld).

RTP1101E: Error reading from the control file (Target=%ld / Ret=%ld / errno=%ld).

RTP1102E:Could not locate expected comma within the control file.

RTP1103E:Error writing to the control file (Target=%ld / Ret=%ld / errno=%ld).

RTP1104E:Error truncating the length of the control file (errno=%ld).

RTP1105E:Error locating the oldest NVTR – Nothing Found.

RTP1106E:Insufficient caller storage for NVTR path/file (Need=%ld / Provided=%ld).

RTP1107E:Error reading from stream during file conversion (errno=%ld).

RTP1108E: Error writing to the destination file for delimiting conversion (errno=%ld / RC=%ld).

RTP1109E:Error opening the output file for CCR conversion (errno=%ld).

RTP1110E:Error reading CCR input file (errno=%ld) on line %ld.

RTP1111E:Error pulling parameters out of CCR input file.

RTP1112E:Error writing to conversion output file (errno=%ld).

RTP1113E:Unexpected error from VALUATION function in the transaction class.

RTP1114E:Unexpected error detected opening the TLTS Source File.

RTP1115E:SQL Error accessing the GSA Control Table (sqlcode=%ld):%s

RTP1116E:SQL Error accessing the GSA Control Error Table (sqlcode=%ld):%s

RTP1117E:SQL Error adding data to the GSA Control Error Table (sqlcode=%ld):%s

RTP1118E:ERROR invalid location state returned (Ctrl=%ld / State=%ld).

RTP1119E:Error building %s transaction – this represents a general system failure.

RTP1120E:SQL Error adding data to the Pending Txn Table (sqlcode=%ld):%s

RTP1121E:Error accessing the pending txn tables for update (Bad State = %ld).

RTP1122E:Error constructing transaction key… posting will fail.

RTP1123E:Key construction called with a sequence number... posting will fail.

RTP1124E:Error updating the GSA Control Tables(%s) (sqlcode=%ld):%s

RTP1125W:WARNING shop send set called without transactions (%s).

RTP1126E:Unable to VOID VPV transaction - %s(%s).

RTP1127E:Error ARCHIVING path was not established… TLTS will fail to load.

RTP1128E:Invalid command line request to secure utility (%s).

RTP1129E:File opened failed during protection (errno=%ld).

RTP1130E:ERROR occurred unlinking old source file (errno=%ld).

RTP1131E:Buffer overflow during encryption process – PROTECT command terminated.

RTP1132E: WRITE FAILED to encrypted file - iRC=%ld / errno=%ld.

RTP1133E:Error updating transaction in PENDING – State has already passed entry (%s).

RTP1134E:Decryption source has overflowed input buffer (Buff=%ld / Needed=%ld).

RTP1135E:Count discrepancy in DBATCH application (Ctrl=%ld / Count=%ld).

RTP1136E:CheckIssue Query [%d]:%s

RTP1137E:Invalid rebuild request – you must specify {YEAR}{JULIAN DAY}{SEQUENCE}.

RTP1138E:Unexpected internal error during rebuild…

RTP1139E:Write error to new NVTR file (rc=%ld / errno=%ld).

RTP1140E:Open error to new NVTR file (errno=%ld).

RTP1141E:Error in size check of the new NVTR (errno=%ld).

RTP1142E:Error in seek to start of the new NVTR (errno=%ld).

RTP1143E:Error in ZONED DECIMAL export of input buffer.

RTP1144E:Unexpected error from EXCEPTion class.

RTP1145E:Error in FDOPEN for stream conversion (errno=%ld).

RTP1146E:Error reading from stream – FGETS failed (errno=%ld).

RTP1147E:Buffer OVERFLOW in ReadSecure(cBuff=%ld / lOut=%ld).

RTP1148E:Error reseting to the start of the file (errno=%ld).

RTP1149E:Error converting to STREAM I/O (errno=%ld).

RTP1150E:Error configuration file is not properly secured.

RTP1151E:Error creating compiled output format (errno=%ld).

RTP1152E:Error writing compiled output format (%s/ Len=%ld / RC=%ld / errno=%ld).

RTP1153E:Error reading compiled output format (%s/ Len=%ld / RC=%ld / errno=%ld).

RTP1154E:Secure INI file accessed without appropriate decryption tools.

RTP9075E:Duplicate header detected trying to dump RTT transaction.

RTP9076W:Unhandled string contained on a sales transaction (%lu/%lu/%lu – ID=%lu).

RTP9077E:Unable to locate FORMAT 0 – RTP Compliant.

RTP9078E:EDS Process was started to reprocess a file with an existing output file.

RTP9079W:Unhandled tender type contained on a sales transaction (%lu/%lu/%lu – ID=%lu).

RTP9080W:Unexpected ITEM level discount without associated item (%lu/%lu/%lu).

RTP9081E:Unexpected string type 8 detected in a transaction Code=%04x (%lu/%lu/%lu).

RTP9082W:%s process ran without an input file.

RTP9083E:Exception Conversion not initialized.

RTP9084E:Missing ORIGINAL BUSINESS DATE from an OO record (Shop=%s).

RTP9085W:Unexpected tax control string type 8 detected in a transaction Code=%04x (%lu/%lu/%lu).

RTP9086E:Missing or invalid parameter for ESET command (%s) – Command Canceled.

RTP9087E:Error locating the “.” in an NVTR filename to parse out control field.

RTP9088E:Unexpected error updating NVTR index table…

RTP9089E:Quick Index Control class not properly initialized.

RTP9090E:Index control file is empty.

RTP9091E:Unexpected/Undefined error finding oldest NVTR…

RTP9092E:Unexpected error in accessing %s control information – System Failing!

RTP9093E:Unexpected error during initialization of catch up (%s).

RTP9094E:Unexpected error during initialization of exception roll forward (%s).

RTP9095W:Termination signal detected.

RTP9096E:Invalid fix code requested for exception management (Valid Codes: %s)

RTP9097E:Unexpected error in the report request class…

RTP9098E:Could not determine controller number from filename:%s

RTP9099E:GSA Controller Management class not initialized.

RTP9100W:Duplicate store open request received from controller %ld.

RTP9101E:Transaction class was not properly initialized.

RTP9102E:Store close received without store open (Store=%s).

RTP9103E:INCONSISTENCY detected in second submission from TLTS processing.

RTP9104E:Could not locate 15S1/2 for a stored value 6 string (Position=%ld).

RTP9105E:ERROR S1/S2 combos orphaned – assumed gift card.

RTP9106E:ERROR S%ld string found when expecting an S1 for gift card – string dropped.

RTP9107E:Missing and/or invalid CASH OUT SKU configuration:%s

RTP9108E:Invalid Default Tanami Set code requested (%c).

RTP9109E:SYNTAX error – Host Transfer requires a source and destination host code.

RTP9110E: Parameter data provided for a fix code that requires none (Codes w/Data: I).

RTP9111E:Item Batch Fix did not have correct parameters (\”%s\”).

**CCmdLine**

RTP1020E:Memory allocation failed (%u bytes) w/rc=%ld reading command line.", Size, errno

RTP1021E:Memory allocation failed (%lu bytes) w/rc=%ld storing the non-option definition.", Size, errno

RTP9010E:Duplicate command line parameter definition (\"%s\").", pcParameter

RTP9011E:Duplicate command line parameter received (\"%s\").", pcParameter

RTP9012E:Missing required command line parameter: %s", pcParameter

RTP9013E:Numeric value out of range (%ld to %ld) for parameter: %s" Low, High, Switch

RTP9014E:Missing numeric value with command line parameter: %s”, Switch

RTP9015E:Missing string value with command line parameter: %s", Switch

RTP9016E:Multiple "%ss" detected on a command line requiring a unique argument.", Switch

RTP9017E:Missing required non-option command line argument: %s", Switch

RTP9018E:Duplicate non-option command line parameter definition (\"%s\").", Switch

RTP9019E:Unexpected command line value detected: “%s” , Parameter

RTP9020E:Unexpected command line option: %s , Parameter

**CDirMgmt**

RTP1550W:Error opening directory scan (rc=%d). errno

RTP8060W:End of directory listing.

**CINIDecode**

RTP1000E:Memory Allocation Failed requesting %lu bytes (rc=%ld).", sizeof(INILINKLIST), errno); -------------(ALSO USE IN RTPINIT, RTPDEBUG, Csupport, CNmPipe, CDirMgmt)

RTP1100E:File stream I/O error reading the INI file (rc=%ld)", m\_lLastError);

RTP1104E:Error opening INI file (%s).", cTemp);

RTP8000E:Invalid format - missing \"=\" in the input file (Line=%d).", m\_lLineCnt);

RTP8001W:Error locating INI variable (%s)", cTemp);

RTP8002W:Field retrieval requested without the appropriate line get.

RTP8003W:Field %d from variable %s was not found.", iIndex, m\_pLast->cVariable);

RTP8004W:Past the end of the source file.

**CTransact**

RTP1900E:Insufficient storage to export transaction key.

RTP1901E:Key Field "%s" does not exist.

RTP1902E:Unreported data error detected during transaction analysis.

RTP1903E:Transaction Buffer Load Mismatch (Hdr=%lu – Offset=%lu).

RTP1904E:Unable to locate the record sequence number (%s).

RTP1905E:Buffer storage exceeded during transaction input buffer rebuild (Size=%ld / Limit=%ld).

RTP1906E:Unexpected error from input configuration class.

RTP1907E:Error AddRecord exceeded sequence number capabilities.

RTP8900E:Data Failure: %s

RTP8901E:Unknown message reference provided in input verification (ID=%ld).

RTP8902E:Transaction class not properly initialized.

RTP8903E:Invalid Field Type detected in the input format (Type=%lu).

RTP8904E:Unable to locate sequence %s during input rebuild.

RTP8905E:Date field contains neither eight or six digits as required.

RTP8906E: Field data type (%c) does not have an appropriate conversion routine.

RTP8907E: Field Name (%s) does not exist under sequence %s.

RTP8908E:Error evaluating %s equation from within the class.

**CAudit**

RTP1700E: Unknown system error within the CAudit Class.

RTP1701E:Conditional Numeric Evaluation Failed.

RTP1702E:Conditional Numeric Evaluation Error Detected (MsgID=%ld).

RTP1703E:Equation Numeric Evaluation Failed.

RTP1704E:Equation Numeric Evaluation Error Detected (MsgID=%ld).

RTP1705W:WARNING potential Equation Numeric Evaluation Failed.

RTP8950E:Unknown message reference provided in audit verification (ID=%ld).

**CInForm**

RTP1550E:Error attaching to the Input Format Shared Memory Segment (rc=%d).

RTP1551E:Error loading and/or locating the Input Format Shared Memory Segment.

RTP1552W:Wait released due to system shutdown.

RTP1553E:CM Shared Memory Version not as expected (Current=%ld / Application=%ld).

RTP8200E:Error input format load wait time out of range (15 to 3600). Value=%ld

RTP8201E:Input Format buffer was not initialized – Segment load aborted (Ptr=%lu; Size=%ld).

RTP8202E:No record types were defined for input format.

RTP8203E:Input Format buffer not initialized.

RTP8204E:Transaction Index out of range - Internal S/W error.

RTP8205W:Transaction Code not defined or missing record definitions (Code=%s).

RTP8206E:Input Format buffer not initialized or retrieval not successfully reset.

RTP8207E:Record Index out of range - Internal S/W error.

RTP8208W:Record type not defined for the give transaction (Code=%s – Type%s).

RTP8209W:Edit Code is out of range (%lu).

RTP8210E:Software error caused class to walk off end of edit list.

RTP8211W:Message Code is out of range (%ld).

RTP8212E:LocateField called for a non-unique record type (Tran=%s/Rec=%s/Field=%s).

RTP8213W:Could not locate field (Tran=%s/Rec=%s/Field=%s).

RTP8214E:Requested Host Identifier does not exist (%s).

RTP8215E:FAILed creating a new entry in the GOLD table - Table FULL (%s).

**CNVTR**

RTP1750E:Error opening the Transaction Repository (rc=%d). errno

RTP1751E:Error checking the size of the Transaction Repository (rc=%d).

RTP1752E:Error attempting to lock the Transaction Repository (rc=%d).

RTP1753E:Error attempting to unlock the Transaction Repository (rc=%d).

RTP1754E:Error truncating an invalid Transaction Repository (rc=%d).

RTP1755E:Error writing to the Transaction Repository (Size=%ld - rc=%d).

RTP1756E:Error seeking %ld in the Transaction Repository (rc=%d).

RTP1757E:Error reading the header from the Transaction Repository (rc=%d).

RTP1758E:Error opening prior NVTR file (rc=%d): %s

RTP1759E:Error reading NVTR header information (rc=%d/NULL=EOF).

RTP1760E:Error writing a transaction to the Transaction Repository (Size=%ld - rc=%d).

RTP1761E:Error key retrieval required on entry that had an offset (Offset=%ld).

RTP1762E:Error reading previous transaction size from the repository (rc=%ld/errno=%d).

RTP1763I:Scan required of the NVTR file series to resynchronize Host Translator.

RTP1764E:Error reading transaction information out of the repository (rc=%ld/errno=%d).

RTP1765E:Error CheckPriorKey was asked to move out of range (Offset=%ld).

RTP1766E:Error accessing the directory to locate the next NVTR file.

RTP1767W:Forced NVTR pointer bump due to prior system failure.

RTP1768E:Error CheckCurrentKey was asked to move out of range (Offset=%ld).

RTP1769E:Error sending signal to host DTRAN (rc=%d).

RTP1770E:Error seeking to end of PRIOR file (rc=%ld/errno=%d).

RTP1771E:Error seeking to load PRIOR adjustment (rc=%ld/errno=%d).

RTP1772E:Error reading PRIOR adjustment (rc=%ld/errno=%d).

RTP1773E:Error accessing the directory to locate the prior NVTR file.

RTP1774E:Error locating the current NVTR file on the file system.

RTP1775W:Invalid offset detected during resynchronization (lPos=%ld).

RTP1776E:Error in Prior Transaction – prior file get that may result in shutdown.

RTP3100E:Error attempting to retrieve the transaction buffer from CTransact.

RTP3101E:Fatal error occurred attempting to synchronize with bundle for host.

RTP3102E:UNKNOWN Fatal error occurred re-reading matched transaction.

RTP8600W:Unexpected NVTR file size – File truncated and re-established (Size=%ld).

RTP8601W:File truncation required on the NVTR file (%ld down to %ld).

RTP8602W:File integrity adjustment required on the NVTR file (%ld down to %ld).

RTP8603E:Filename error handling new physical file creation (%s).

RTP8604E:Number of repositories created within a single day exceeded 99 file limit.

RTP8605E:Transaction Repository file not initialized.

RTP8606E:Transaction Repository file and path information is not available.

RTP8607W:NVTR Scan terminated due to abort signal from CSUPPORT.

**CBundle**

RTP1950E:Error seeking %ld in Host Specific Bundling File (rc=%d).

RTP1951E:Error opening the Bundling Control File (rc=%d).

RTP1952E:Error attempting to lock the Bundling Control File (rc=%d).

RTP1953E:Error attempting to unlock the Bundling Control File (rc=%d).

RTP1954E:Error reading the host structure from the Bundling Control File (rc=%d).

RTP1955E:Error seeking %ld in the Bundling Control File (rc=%d).

RTP1956E:Error writing a host structure to the Bundling Control File (Size=%ld - rc=%d).

RTP1957E:Host Structure mismatch within the Bundling Control File (read=%ld – rc=%d – offset=%ld).

RTP1958E:Error opening the Host Specific Bundling File (rc=%d).

RTP1959E:Error attempting to lock the Host Specific Bundling File (rc=%d).

RTP1960E: Bundling class not properly initialized.

RTP1961W:A partial bundle write was deleted from file (Size=%ld).

RTP1962E:Error truncating an invalid Host Specific Bundling File (Size=%ld/rc=%d).

RTP1963E:File truncation required but not performed for the Bundling File.

RTP1964E:Error reading the Host Specific Bundling File (iRC=%ld/errno=%d).

RTP1965E:Error determining location of the transaction (rc=%d).

RTP1966E:File Stat error determining host bundling file size (rc=%d).

RTP1967E:Unable to locate the host control structure in the bundling system.

RTP1968E:Error seeking cursor during bundle write (Pos=%ld/Rel=%s/rc=%d).

RTP1969E:Integrity issue between bundle header (%ld) and parameter (%ld).

RTP1970E:Error adding a new transaction to the bundle file (Key=%s/Len=%ld/rc=%ld/errno=%d).

RTP1971E:Error writing update to the Host Specific Bundle file (rc=%ld/errno=%d).

RTP1972E:Error seeking start of the Host Specific Bundle file (rc=%ld/errno=%ld).

RTP1973E:Error locating the selected host code – request canceled.

RTP1974E:Unknown error scanning the bundle file (%s).

**CSupport**

RTP1500E:”dup2” failed to redirect standard output/error (rc=%d). errno

RTP1501W:Error occurred sending a signal (ID=%u) to a child process (PID=%lu) – RC=%d

RTP1502W:Input Format Load In Progress (%s).

RTP1503E:Time-out occurred waiting for Daemon to become operational.

RTP1504E:Error locating Sequence, Archive, and Exception – RECOVERY failed (Key=%s).

RTP1505E:Unable to locate prior instance of a managed process (old pid=%ld).

RTP1506E:Encryption algorithm exceeded anticipated buffer size limit (lSize=%ld / lOff=%ld).

RTP7000W:Unexpected Command Processor Response Code (%d/%ld). RespCode ExtError

RTP7001W:Duplicate request received to load input version format information.

RTP7002W:Maximum number of input version formats has been exceeded - load failed.

RTP7003W:Invalid command code received by the command processor (Code=%ld). Code

RTP7004W:Daemon is not in a state from which the requested command can be executed.

RTP7005E:Unexpected database error encountered (rc=%ld). lExtErr

RTP7006E:Unexpected operating system error encountered (rc=%ld). LExtErr

RTP7007E:Error occurred starting the daemon (extended RC=%ld) lExtErr

RTP7008E:An unexpected internal error occurred executing the command.

RTP7009E:Invalid software STATE accessing the NVTR file information (State=%d).

RTP7010E:Missing or invalid parameter provided with command.

RTP8050E:Screen Named Pipe not initialized.

RTP8051E:Logging Named Pipe not initialized.

RTP8052E:Command Named Pipe not initialized.

RTP8053W:Process ID not found in the process control list (pid=%lu). PID

RTP8054E:Attempt was made to remove a core daemon process (pid=%lu) – daemon is failing.

RTP8055W:Command Named Pipe name not initialized.

RTP8055W:Process name retrieval attempted without sufficient working area for the name – request failed.

RTP8056E:Master Shared Memory Segment not initialized.

RTP8057W:Input Format has not yet been loaded into memory (%s).

RTP8058E:Insufficient space to retrieve user ID and password (Req=%d).

RTP8059E:Invalid search type requested for location of Shop Number (Type=%d).

RTP8060E:Unable to locate the requested location within the shared memory business logic (%s).

RTP8061E:Unknown error out of the master shared memory segment class.

RTP8062E:No data found looking for base location information.

RTP8063E:Unexpected error in support module (%s).

RTP8064E:Host Code could not be located on the server (%s).

RTP8065E:Invalid PCI State flag submitted (%ld).

RTP8066E:Error QUEUE’ing command to CPROC.

**CNmPipe**

RTP1030E:Memory allocation failed (%u bytes) w/rc=%ld creating pipe class variable.", Size, errno

RTP1031E:Error making named pipe in the file system (rc=%ld) - %s errno and name

RTP1032E:Error opening named pipe (rc=%ld) - %s errno and name

RTP1033E:Unanticipated pipe creation error.

RTP1034E:Time-Out opening named pipe – %s name

RTP1035E:Error attaching a stream to the open pipe (rc=%d).

RTP1036W:Error getting string off a stream (rc=%d)

RTP1037W:Error writing to the pipe (rc=%d)

RTP1038W:Error reading from the pipe (rc=%d)

RTP1039W:Read past the end of a pipe.

RTP1040E:Unable to switch pipe back to non-blocking mode (rc=%ld).

RTP1041E:Error waiting for a select on a response pipe (rc=%d).

RTP1042W:Time-out occurred waiting for response.

RTP1043E:Error putting string on to stream (rc=%d)

RTP1044E:Error flushing inbound pipe (iRC=%ld / errno=%ld).

RTP8040E:Unable to create stream because pipe is not open.

RTP8041E:Pipe has not been initialized.

RTP8042E:Stream has not been attached to the pipe.

RTP8043E:Response pipe has not been initialized.

**CMaster/CCMMemory**

RTP1300E:Unable to get the master shared memory ID (rc=%d).

RTP1301W:Shared memory segment is unavailable at this time.

RTP1302E:Timed out waiting for OK to access shared memory segment.

RTP1303E:Shared Memory Access Control Semaphore failed.

RTP1304E:Error opening RTPSUPPORT.INI file to load master segment parameters.

RTP1305E:Error creating shared memory ID (ProjID=%d) - %s Project ID and path

RTP1306E:Unable to attach to created master shared memory segment.

RTP1307E:Error creating the access control semaphore.

RTP1308E:Error locking the shared memory segment (rc=%d).

RTP1309E:Error unlocking the shared memory segment (rc=%d).

RTP1310E:Error initializing the access control semaphore.

RTP1311W:Unable to locate requested version ID (%s).

RTP1312E:Timed out waiting for access shared memory segment.

RTP1313E:Error attaching to business logic extension (ID=%ld/errno=%ld).

RTP1314E:Memory Allocation from BusLog Failed requesting %lu bytes

RTP1315E:Unable to get the extension shared memory ID (rc=%d).

RTP1316E:System failed to insert item into SHARED MEMORY link list.

RTP1317E:Incompatible shared memory segment versions (ID=%08x).

RTP1318E:Error on semaphore wait for next transaction (rc=%d).

RTP1319E:Error seeking to start of Business Logic dump file for reload (errno=%d).

RTP1320E:Error reading the “%s” from Business Logic dump file for reload (RC=%ld/errno=%d).

RTP1321E:Time-out waiting for shared memory segment ID.

RTP8030E:Master segment creation failed because parameter “%s” is out of range.

RTP8031E:Master memory segment not initialized.

RTP8032E:AddHost failed for \"%s\" : Length-Name=%d/Program=%d", code , len of name, len of prog

RTP8033E:Host Code Table Overloaded on "%s". , pCode

RTP8034E:Attempted to add a duplicate host code (%s). , pCode

RTP8035E:Host Table Index out of range(%d of %u). index count

RTP8036E:Duplicate Input Format load request received (Code=%s).

RTP8037E:Input Format Table has exceeded its limit – Load canceled (Limit=%u).

RTP8038E:Input Format entry does not exist for update (Code=%s).

RTP8039E:Attempted to add a duplicate location code (%s). , pCode

RTP8040E:Location Code Table Overloaded on "%s". , pCode

RTP8041E:AddLocation failed for \"%s\" : Length-ID=%d

RTP8042E:Device update failed due to an invalid location (%s).

RTP8043E:AddLocationTanami failed for \"%s\" : Length-ID=%d

RTP8044E:AddLocationTanami failed because location \”%s\” doesn’t exist.

RTP8045E:AddLocationTanami failed because of a duplicate LOB Code: %ld.

RTP8046E:Location Tanami Code Table Overloaded for "%s".

RTP8047E:Location Key get failed because index out of range (%d – Limit=%d).

RTP8048E:Space in encryption table exceeded – SZ\_MKEY is too small.

RTP8049E:Invalid selection parameter %ld on Statistic updates.

RTP8050E:CM Memory class not properly initialized.

RTP8051E:Unexpected LOCKING error on CM shared memory segment.

RTP8052E:Process ID does not match expected ID (Mem=%ld / Current=%ld).

RTP8053E:List Index %ld is out of range (%ld to %ld).

**CGSAFile/CGSATLog/CGSAConvert/CTLTSFile**

RTP1650E:Unexpected error locking a GSA input file (rc=%d).

RTP1651E:Error opening a batch GSA input file (rc=%d).

RTP1652E:Class initialization and/or operation failed.

RTP1653E:Missing segments found within the GSA input file.

RTP1654E:Error reading a segment from the GSA input file (rc=%d/errno=%d).

RTP1655E:Base string in a GSA record did not include a type.

RTP1656E:GSA Conversion Class not initialized.

RTP1657W:Unexpected second string on non-sale transaction header (%s).

RTP1658W:Terminal 15S5 record received without a Terminal #1 definition.

RTP1659E:Error attempting to seek through TLTS file (lRC=%ld / errno=%ld).

RTP1660E:At end of TLTS source file…

RTP1661E:Error reading from TLTS source file (lRC=%ld / errno=%ld).

RTP1662E:Error walking file segment list – fell off the end of the file (lBuff=%d / lWrite=%ld).

RTP1663W:Warning a record fragment was pulled from a TLTS tlog file.

RTP8450E:GSA TLOG Class has not been properly initialized.

RTP8451E:Requested String (%u) could not be located.

RTP8452E:Requested Field (%u) within String (%u) could not be located.

RTP8453W:Close received without corresponding open (Ctrl=%ld).

RTP8454W:Controller %ld was not found in the control table – could be missing 15S5.

RTP8455W:Shop %s could not be found during locate - Potential 15S5 issue.

RTP8456W:Shop not defined for Controller=%ld Terminal=%ld – Potential 15S5 issue.

RTP8457W:Terminal %ld was not found for Controller=%ld - Potential 15S5 issue.

RTP8458W:Invalid character in OPTIONAL store indentifier - %ld.

RTP8459E:UserString pull would over run the input field (iOff=%d / iLen=%d / Size=%d).

RTP8460W:Unexpected TENDER TYPE detected in a 15S1 (Type=%ld).

RTP8461W:Error store reopened during the middle of the day (Shop=%s).

RTP8462E:Unable to locate associated VOID PREVIOUS list record (%ld/%ld/%ld/%ld).

RTP8463W:Main string not yet processed – data ignored (Type=%ld / Count=%ld).

RTP8464E:Error adding original order information – transaction dropped.

RTP8465W:Error locating original item discount for void processing.

RTP8466E:Error writing to destination for GetField – Field longer than max allowed (Str=%ld / Fld=%ld).

RTP8467E:Filename does not conform to appropriate standards for TLTS conversion.

**CTransFile**

RTP1010E:Memory allocation failed for transaction of %lu (rc=%d)” size, errno

RTP1011E:Memory allocation failed for definition data of size %ld (rc=%d)", lBuff, errno);

RTP1012E:Memory allocation failed for error information of size %ld (rc=%d)", lBuff, errno);

RTP1200E:Error opening a batch transaction input file (rc=%ld) ERRNO

RTP1201E:Unexpected error locking a transaction input file (rc=%d).

RTP1250E:Error reading extended base sequence (Trans=%ld; rc=%d).", m\_lTransCount, errno);

RTP1251E:Error reading base sequence (Trans=%ld; rc=%d).", m\_lTransCount, errno);

RTP1252E:Error seeking past transaction data block (Trans=%ld; rc=%d).”, m\_lTransCount, errno

RTP1253E:Error reseting the file position to ZERO (rc=%d).", errno);

RTP1254E:Error reading the transaction information (Size=%lu; Ret=%ld; Err=%d).”, size, read RC, errno

RTP1255E:Error reading the current cursor position (Ret=%ld), errno

RTP1256E:Error setting the current cursor position (Pos=%ld; Ret=%d), offset, errno

RTP1257E:Error reading base sequence (rc=%d).", errno);

RTP3000E:Missing Transaction Marker (Trans=%ld)", m\_lTransCount);

RTP3001E:Missing Transaction Marker (Pos=%ld)", Offset);

RTP3010E:Nested Logical Units are Illegal (Trans=%ld)", Offset);

RTP3015E:Logical Unit Mismatch (Start=”$$$”; End=”$$$”).

RTP3016E:Logical Trailer Count Mismatch Caused File Failure (ID=$[$]; TRN=$$; Trailer=%lu;

Actual=%lu) , ID, Proper Filename, Transaction Code, Trailer Count, Actual Count

RTP8010E:Input File Not Open or Validated.

RTP8011E:Invalid SEEK argument requested for transaction input file (Pos=%ld; Ret=%d). , Offset, errno

RTP8012E:Unknown Unit ID specified during file SEEK operation (ID=$[$]). ID String and Filename

**CLogicalBatch**

RTP1015E:Memory allocation failed for internal class (Size=%ld; rc=%d)", size, errno); (ALSO USED IN RTPINIT)

RTP1016E:Memory allocation failed for internal variable (Size=%ld; rc=%d)", size, errno);

RTP1260E:Error renaming processed file (rc=%ld) – %s errno, filename

RTP3020E:Orphan records detected in logical batch file causing failure (Cnt=%lu) – %s Count, Filename

RTP3021E:Record Count Mismatch ID=%s On Transaction Codes: %s ID, transaction code list

RTP8020E:Transaction File class not initialized.

RTP8021E:Transaction File class does not have logical unit markers defined.

RTP8022E:Transaction retrieval attempted prior to logical unit retrieval.

RTP8023W:Get Transaction failed - Get Unit Required (ID=%s).

RTP8025E:Attempted second validation of a transaction file which results in a validation failure.

RTP8026W:Unable to locate the unit information requested (ID=%s).

**CHostConfig/CUnloadConfig**

RTP1890E:Outbound Configuration load has failed resulting in host translator termination.

RTP1891E:Bundle Configuration load has failed resulting in host translator termination.

RTP1892E:SKIP LIST load has failed resulting in host translator termination.

RTP1893E:Duplicate master query record on extract \”%s\”.

RTP1894E:WALKED off end of query list on extract \”%s\”.

RTP1895E:SCAN LIST load has failed resulting in host translator termination.

RTP8800E:Host Configuration Class not properly initialized.

RTP8801E:Unable to locate field definitions for output record ID %ld.

RTP8802E:Group repeat last retrieval field since no prior reset available.

RTP8803E:Indexing CHAR request exceeded system limit (%ld) – Default assumed.

RTP8804E:Invalid Index Configuration based on internal data control.

RTP8805E: Insert Linked List called with a BASE configuration entry.

RTP8806E: Insert Linked List called without a base index.

**CdbLOps/CDbBWork/CDBLOpD/RTPDLOAD/CDBLocOp/CDBPromCtrl**

RTP1350E:Error reading the source file (line=%ld / errno=%ld).

RTP1351E:Parameter (%s) missing from the input string.

RTP6150E:OPL CURSOR open failed[%d]:%s

RTP6151E:OPL CURSOR fetch failed[%d]:%s

RTP6152E:Error on select of archive device entry (sqlcode=%ld).",

RTP6153E:Unexpected error deleting aged records from OPL\_FILE\_RUN (sqlcode=%ld): %s

RTP6154E:Unexpected error deleting aged records from OPL\_LOGICAL (sqlcode=%ld): %s

RTP6155E:Invalid state updating the business logic operational archive (State=%ld).

RTP6156E:Unexpected error updating the archive (sqlcode=%ld):%s

RTP6157E:Unexpected SQL error querying DB State Table (sqlcode=%ld):%s

RTP6158E:Unexpected SQL error updating DB State Table (sqlcode=%ld):%s

RTP6159E:Unexpected SQL error inserting into DB State Table (sqlcode=%ld):%s

RTP6160E:Unexpected SQL error rolling back DB State Table (sqlcode=%ld):%s

RTP6161E:Unexpected SQL error commiting to DB State Table (sqlcode=%ld):%s

RTP6162E:Unexpected SQL error accessing the CONTROL\_KEY data (sqlcode=%ld):%s

RTP6163E:Unexpected SQL error updating the CONTROL\_KEY data (sqlcode=%ld):%s

RTP6164E:SQL error opening a cursor (%s) (sqlcode=%ld):%s

RTP6165E:SQL error updating PRM\_LOG (sqlcode=%ld):%s

RTP8650E:Database Operational Log not initialized.

RTP8651E:Invalid or missing system code – process terminated.

RTP8652E:Missing or invalid location/business date for CDBLOCOP class.

RTP8653E:Database Location Statistics not initialized.

RTP8654E:SQL error encountered updating the location stats (sqlcode=%ld):%s

RTP8655E:Store number not specified for AUDIT logging…

RTP8656E:Unexpected out of sequence submitting log entry – possible security breach.

RTP8657E:Database Maintenance Tables not initialized.

RTP8658E:Invalid state during write information (State=%ld).

RTP8659E:SQL error updating BATCH WORKING table (sqlcode=%ld):%s

RTP8660E:Batch Working Table Class not initialized.

**CDbLog/CDynamicSQL**

RTP6200E:LOG CURSOR open failed[%d]:%s

RTP6201E:LOG CURSOR fetch failed[%d]:%s

RTP6202E:Log entry INSERT failed[%d]:%s

RTP6203E:LOG CURSOR declaration failed[%d]:%s

RTP6204E:LOG CURSOR preparation failed[%d]:%s

RTP6205E:Log entry DELETE failed[%d]:%s

RTP6206E:SQL Error opening the dynamic SQL cursor (sqlcode=%ld):%s

RTP6207E:SQL error on fetch for dynamic SQL (sqlcode=%ld):%s

RTP8300E:Database Logging Class not initialized.

RTP8301E:Dynamic SQL Class not initialized.

RTP8302E:You can not add %s after the query has been prepared.

RTP8303E:At least one field must be added for retrieval – SQL statement build failed.

RTP8304E:SQLSQLDAAllocate failed for %s (sqlcode=%ld).

RTP8305E:You can not prepare a dynamic sql statement twice.

RTP8306E:Bind structure was not initialized.

RTP8307E:Bind structure does not have enough array space.

RTP8308E:FETCH not executed.

RTP8309E:Requested field is out of range (%ld).

**CDbOut/CDbTmpWork**

RTP8322E:Outfile List CURSOR open failed[%d]:%s

RTP8323E:Outfile List CURSOR declaration failed[%d]:%s

RTP8324E:Outfile List CURSOR fetch failed[%d]:%s

RTP8325E:Host code not provided initializing temporary working table class.

RTP8326E:Error cleaning the temporary table (sqlcode=%ld):%s

RTP8327E:Temporary working database class not initialized.

RTP8327E:Error on insert into temporary table (sqlcode=%ld):%s

**CCfgTranslate/CUnloadConfig**

RTP6100E:Translation Count Retrieval Failed[%d]:%s

RTP6101E:OTF\_CODE insertion failed[%d]:%s

RTP6102E:OTF\_CODE update failed[%d]:%s

RTP6103E:Translation delete flag set failed[%d]:%s

RTP6104E:OTF\_TRANSLATE insertion/update failed[%d]:%s

RTP6105E:OTF\_TRANSLATE delete non-updated failed[%d]:%s

RTP6106E:Translation Set/Key retrieval failed[%d]:%s

RTP6107E:Field not found in TRANSLATE request: %ld

RTP6108E:Cache Preload CURSOR declaration failed[%d]:%s

RTP6109E:Cache Preload CURSOR open failed[%d]:%s

RTP6110E:Cache Preload CURSOR fetch failed[%d]:%s

RTP6111E:OTF\_TRANSLATE delete QUERY of non-updated failed[%d]:%s

RTP6112E:OPL\_OUTFILE\_HISTORY query failed to locate file(%ld):%s

RTP6113E:OPL\_OUTFILE\_HISTORY delete failed on file(%ld):%s

RTP8670E:Translation Class not initialized.

RTP8671E:Translation Class not initialized or in an invalid state($..$=%d).

RTP8672E:Type/Index combination does not exist in configuration (%c - %ld).

RTP8673E:No more field information.

**CCache**

RTP2850W:Error locating SET entry on the clean second pass(%s).

**CDbExcept**

RTP3400E:Error on select of overall status (sqlcode=%ld).

RTP3401E:Error allocating %s for CLOB retrieval (sqlcode=%ld).

RTP3402E:SQL Error declaring cursor to retrieve CLOBs (sqlcode=%ld).

RTP3403E:SQL Error opening cursor to retrieve CLOBs (sqlcode=%ld).

RTP3404E:SQL Error fetching cursor to retrieve CLOBs (sqlcode=%ld).

RTP3405E:SQL Get CLOB size on %s failed (sqlcode=%ld).

RTP3406E:SQL Read CLOB data from %s failed (sqlcode=%ld).

RTP3407E:Update exception "process instance" status failed (sqlcode=%ld):%s

RTP3408E:Update exception "transaction" status failed (sqlcode=%ld):%s

RTP3409E:SQL Error inserting EXC\_TRANSACTION record (sqlcode=%ld):%s

RTP3410E:SQL Error updating EXC\_TRANSACTION record (sqlcode=%ld):%s

RTP3411E:SQL Error inserting EXC\_ISSUE record (sqlcode=%ld):%s

RTP3412E:SQL Error updating EXC\_PROCESS\_INSTANCE record (sqlcode=%ld):%s

RTP3413E:SQL Error inserting EXC\_PROCESS\_INSTANCE record (sqlcode=%ld):%s

RTP3414E:SQL Error ALLOCATE for EXC\_PROCESS\_INSTANCE (sqlcode=%ld):%s

RTP3415E:SQL Error writing LOB to EXC\_PROCESS\_INSTANCE record (sqlcode=%ld):%s

RTP3416E:SQL Error querying the next EXC\_PROCESS\_INSTANCE sequence (sqlcode=%ld):%s

RTP3417E:SQL Error updating exception status codes (sqlcode=%ld):%s

RTP3418E:SQL Error counting exception issue status codes (sqlcode=%ld):%s

RTP3419E:SQL Error retrieving the copy information from EXC\_TRANSACTION (sqlcode=%ld):%s

RTP3420E:Error commiting copy transaction to the database (RC=%ld).

RTP3421E:SQL error failed declaring issue retrieval cursor (sqlcode=%ld):%s

RTP3422E:SQL error fetching from copy retrieval cursor (sqlcode=%ld):%s

RTP3423E:SQL error retrieving locator for CORRECTED LOB (sqlcode=%ld).

RTP3424E:SQL Error retrieving the overall exception status code (sqlcode=%ld):%s

RTP3425E:SQL Error updating the CORRECTED buffer (sqlcode=%ld):%s

RTP3426E:SQL Error updating the batch fix table for (sqlcode=%ld):%s

RTP3427E:SQL Error inserting information to EXCEPT2 (sqlcode=%ld):%s

RTP3428E:SQL Error deleting information from EXCEPT2 (sqlcode=%ld):%s

RTP8400E:Exception Management Transaction Key not initialized.

RTP8401E:Exception Management class not initialized.

RTP8402W:WARNING we did NOT find a match for MarkProcessed: %s

RTP8403E:Exception transaction has already been submitted for processing (%c).

RTP8404E:Transaction retrieved CLOB does not meet minimum lengths (Len=%ld).

RTP8405E:UNKNOWN ERROR locating field information…

RTP8406E:Summary Table access class not initialized properly.

**CdbLease/CDBCCS**

RTP5800E:Lease class not initialized or in an invalid state.

RTP5801E:OPL\_LEASE\_PROCESSID delete failed with days=%ld (sqlcode=%ld):%s

RTP5802E:OPL\_LEASE\_PROCESSID retrieval failed on state %d (sqlcode=%ld):%s

RTP5803E:GetLease appears to be stuck in an endless loop – processing terminated.

RTP5804E:GetLease exceeded control limit for terminal assignment (%ld - %ld).

RTP5805E:Credit Guard class not initialized or in an invalid state.

RTP5806E:CCS\_MASTER List CURSOR fetch failed[%d]:%s

RTP5807E:Reference retrieval failed with unknown error.

RTP5808E:Get old reference key failed.

RTP5809E:No old reference key code.

**CSourceSystem**

RTP7770E:Unexpected state during attempt to generate a batch key (lState=%ld).

RTP7771E:SQL Error retrieving the next sequence number for system (%s) – sqlcode=%ld:%s

RTP7772E:SQL Error inserting a new sequence number for system (%s) - sqlcode=%ld:%s

RTP7773E:SQL Error updating the sequence number for system (%s) - sqlcode=%ld:%s

RTP7774E:Loop limit exceeded attempting to update sequence number for %s.

**CCfgInForm**

RTP8500E:Input Format Configuration Class not initialized.

RTP8501E:New Exception State code is not a valid setting (%c - %s).

**CCfgInFormVer**

RTP8550E:Input Format Version Configuration Class not initialized.

RTP8551E:Unable to locate an active input format configuration.

**CCfgHost**

RTP6125E:Host Retrieval CURSOR open failed[%d]:%s

RTP6126E:Host Retrieval CURSOR declaration failed[%d]:%s

RTP6127E:XRef Retrieval CURSOR open failed[%d]:%s

RTP6128E:XRef Retrieval CURSOR declaration failed[%d]:%s

RTP6129E:HLC|CCfgHost Count HLC\_OPERATION XRef|[%d]:%s

RTP6130E:XREF Host Configuration CURSOR fetch failed[%d]:%s

RTP6131E:Host Configuration CURSOR fetch failed[%d]:%s

RTP6132E:Host Activation Date fetch failed[%d]:%s

RTP6133E:Host Control Data fetch failed[%d]:%s

RTP8700E:Host Configuration Class not initialized.

**CCfgOut**

RTP8850E:Unable to locate an active output version format for the host (%s).

RTP8851E:Outbound Configuration Class not initialized.

RTP8852E:XML Action was not N, E, S, W (Value=%c).

**CCfgFDS**

RTP6900E:Error retrieving host file delivery information (sqlcode=%ld):%s

RTP6901E: FDS\_File List CURSOR open failed[%d]:%s

RTP6902E:FDS\_File List CURSOR declaration failed[%d]:%s

RTP6903E:FDS Configuration Class not initialized.

RTP6904E:FDS\_File List CURSOR fetch failed[%d]:%s

RTP6905E:FDS\_File was unable to locate a filename for %s.

RTP6906E:TransferMove Rename Failed (errno=%ld).

RTP6907E:TransferMove CHMOD Failed (errno=%ld).

RTP6908E:Failure getting user ID information (errno=%ld) - CHOWN ignored.

RTP6909E:TransferMove CHOWN Failed (errno=%ld).

RTP6910E: FDS\_Command List CURSOR open failed[%d]:%s

RTP6911E:FDS\_Command List CURSOR declaration failed[%d]:%s

RTP6912E:FDS\_Command List CURSOR fetch failed[%d]:%s

RTP6913E:Password mismatch prevented update for %s.

RTP6914E:New password exceeded field size for %s.

RTP6915E:Error locating Host Code to get FDS information (%s).

**CcfgLocation/CDBSec**

RTP5001E:HLC|CCfgLocation|Count SQL Error[%d]:%s

RTP5002E:Location List CURSOR open failed[%d]:%s

RTP5003E:Location List CURSOR declaration failed[%d]:%s

RTP5004E:HLC|CCfgLocation|INSERT %s SQL Error[%d]:%s

RTP5005E:HLC|CCfgLocation|DELETE %s SQL Error[%d]:%s

RTP5006E:HLC|CCfgLocation|DELETE all SQL Error[%d]:%s

RTP5007E:Location List CURSOR fetch failed[%d]:%s

RTP5008E:Tanami List CURSOR open failed[%d]:%s

RTP5009E:Tanami List CURSOR declaration failed[%d]:%s

RTP5010E:Tanami List CURSOR fetch failed[%d]:%s

RTP5011E:Merchant Code retrieval failed [%d]:%s

RTP5012E:NULL fields contained within a location configuration – location %s ignored.

RTP5013E:SQL Error accessing SEC\_USER (sqlcode=%ld):%s

RTP5014E:SQL Error accessing SEC\_GROUP/SEC\_USERGROUP(sqlcode=%ld):%s

RTP8750E:Location Configuration Class not initialized.

RTP8751E:Requested USER or GROUP does not exist within the system.

**CCfgItem**

RTP5015E:ITM|CCfgItem|Get Type SQL Error[%d]:%s

RTP8755E:Item Configuration Class not initialized.

-------------------------------------------------------------------------------------------------------------------------

1000 – 2999 Operating System Error

3000 – 4999 Data Integrity Error

5000 – 7999 Database Errors

8000 – 9999 User or Input Integrity Error

## Log File Module Name Definitions

| **Name** | **Description** | **Program** |
| --- | --- | --- |
| RTPINIT(IN) | Daemon Initialization program – this message was generated as the system was starting prior to the process taking on a definite role (e.g., Command Processor, Logging, etc…) | RTPINIT |
| RTPINIT(CP) | Daemon Command Processor is the primary process within the daemon. It has ownership of all other processes and shared memory resources. | RTPINIT |
| RTPINIT(SC) | Daemon Screen process handles the routing of all output to a daily console style file. | RTPINIT |
| RTPINIT(LG) | Daemon Logging process which is responsible for processing messages (e.g., warnings, errors, etc…) and reporting them on to the appropriate support system. | RTPINIT |
| RTPINIT(CS) | Daemon Screen Clean process is responsible for removing aged STDOUT files as defined within the RTPSUPPORT.INI file. This process is run once daily just after mid-night. | RTPINIT |
| RTPINIT(LI) | Daemon Input Format Load process is responsible for loading a new input format into a shared memory block for a requesting client. | RTPINIT |
| RTPBATCH | Batch process instance responsible for parsing a batch transaction file into individual transactions and submitting them to the system for processing. | RTPBATCH |
| RTPGSA | GSA batch process instance that is responsible for translating an ISRS polled GSA TLOG into a standard RTP batch input file. | RTPGSA |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. The minimum requirements for the *Logical Batch Open* and *Logical Batch Close* transactions are defined in Chapter 4 of this document. Detailed transaction definitions are provided in the separate document titled *RTP – Batch File Specification*. [↑](#footnote-ref-1)
2. This reference to transactions that have been previously processed includes duplicates within both current business period (e.g., business day) and prior business periods (e.g., duplicate batch). [↑](#footnote-ref-2)
3. *Exception Management* resubmissions are transactions that were previously written to the exception database, have been reviewed/corrected by a user, and are being reprocessed through the system. [↑](#footnote-ref-3)
4. Non-volatile storage referenced here is redundant DASD storage. A database is not necessarily required and may impact overall performance of the system. [↑](#footnote-ref-4)
5. *Business Management Information,* in the case of this reference, includes adding a new Chronological Grouping, Physical Location Grouping, updating the current transaction mode, and updating the sequence number as appropriate. [↑](#footnote-ref-5)
6. Additional information on the contents and operation of the configuration database are provided in section 0 of this document. [↑](#footnote-ref-6)
7. Individual Transaction referenced here is a single type of transaction such as *store open*, *sale*, *return, sign-on,* etc… [↑](#footnote-ref-7)
8. Offset is from the base of the buffer unless used within an array of records (e.g., *Host Specific Control Data Array*). Within array definitions, the offset is from the start of an individual records. [↑](#footnote-ref-8)
9. The CPipe class provides an encapsulation of a bi-directional FIFO by creating a single inbound FIFO with the declared name through which commands are received by the “server” process. Responses are then returned on a similarly named FIFO that has the PID of the client process included as a suffix. [↑](#footnote-ref-9)
10. Additional information about task definitions are provided in section 5.4.3.2 of this document. [↑](#footnote-ref-10)
11. Batch Stage exceptions are limited to file specific errors such as orphan records. Orphan records are only written to the exception management system if the FILE\_BAD\_ WITH\_ ORPHANS is set the NO as described in section 5.4.3.3. [↑](#footnote-ref-11)
12. Integer in the case of this statement does not mean a 16, 32, or 64 bit value but rather indicates that the value contains only whole numbers. [↑](#footnote-ref-12)
13. WARNING use of the numeric types “0” – “9” may in some cases result in the summation of values if instances are grouped for generation of output records. For this reason, a type of “A” with a pattern edit should be used for values that have numeric data but should not be treated as a number. [↑](#footnote-ref-13)
14. Valid characters within the pattern include: #=Numeric, @=Sign or Blank, &=Sign, Blank, or Numeric, ?=Single Any Character, `=Any Non-Blank Character, %=Single Alpha, ^=Single Uppercase Alpha, \_=Single Lowercase Alpha, \* at the end of the string indicates zero or more characters through the end of the string. A ~ in the first position indicates a blank entry is also valid assuming all SPACES in the field. Any other characters contained within the pattern must match exactly as provided. [↑](#footnote-ref-14)
15. The information contained in the EDIT\_DATA for the “Must Exist” edit consists of either a two character record type or a single character followed by a question mark to include all types starting with the given character (e.g., I? will capture IE, IG, etc…). A “~” in front of the type indicates that the field can be filled with spaces if no reference is called for; otherwise, it must contain a valid reference. [↑](#footnote-ref-15)
16. The second character in this field controls the separator field used for “delimited” data fields. A missing value or a literal ‘C’ indicates that the separator should be a ‘,’. [↑](#footnote-ref-16)
17. The transaction code identified for the trigger MUST be set to process for the given host even if there are no resulting output record types defined. Failure to enable this transaction will prevent the system from building bundles. [↑](#footnote-ref-17)
18. Combination of numeric values will be accomplished through simple summation of similar fields across the identified instances. String information is ALWAYS taken from the first or lowest valued instance that matches the given criteria. [↑](#footnote-ref-18)
19. Multiple OTF\_FIELD rows allow the administrator the ability to have a field populated differently depending on evaluation of a conditional expression within the record. Conditionals used in this form MUST provide exclusive results (e.g., one or the other). Unpredictable results will occur if multiple rows are used for a single output record. [↑](#footnote-ref-19)
20. This feature is ONLY supported in delimited files. All other output types treat this field type exactly as a “C”haracter field is treated. [↑](#footnote-ref-20)
21. This field is out of order from the Data Model since it is incorporated into the primary key for the table but follows fields that are not part of the key. For the purpose of INI and load files, the order matches the order in this table rather than in the data model. [↑](#footnote-ref-21)
22. This feature is ONLY supported in delimited files. All other output types treat this field type exactly as a “C”haracter field is treated. [↑](#footnote-ref-22)
23. This field is out of order from the Data Model since it is incorporated into the primary key for the table but follows fields that are not part of the key. For the purpose of INI and load files, the order matches the order in this table rather than in the data model. [↑](#footnote-ref-23)
24. A record header template is inserted at the start of each transmission automatically just prior to being written to the port. A string of “^” in this pattern is replaced with the same size numeric value which represents the size of the data portion of the packet. [↑](#footnote-ref-24)
25. Timing parameters are optional. If excluded the system will not pause between operations. In addition, record set markers are optional. If excluded the system will send up to 8K of data at each time until all data has been transmitted to the destination. [↑](#footnote-ref-25)
26. Related records include ITM\_LOCATION\_REFERENCE or ITM\_ALIAS. Additions, deletions, or changes cause the associated ITM\_MASTER record’s LAST\_MODIFIED field to be updated. [↑](#footnote-ref-26)