

# **Oracle Tuxedo Application Runtime for Batch**

User Guide

12c Release 1 (12.1.1)

September 2013

**ORACLE®**

Copyright © 2010, 2013, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

# Contents

## Introduction

Purpose .....	1-1
Organization .....	1-1
See Also .....	1-2

## Overview of the Batch Runtime Environment

Oracle Tuxedo Application Runtime for Batch Presentation and Structure .....	2-1
Technical Functions .....	2-1
High-Level Functions .....	2-1
Interface-Level Functions .....	2-2
Script Execution Phases .....	2-2

## Using Batch Runtime

Configuration files .....	3-1
BatchRT.conf .....	3-2
Messages.conf .....	3-2
FunctionReturnCode.conf .....	3-2
ReturnCode.conf .....	3-2
Setting Environment Variables .....	3-2
Configuring Batch Runtime in MP Mode .....	3-4
Creating a Script .....	3-5
General Structure of a Script .....	3-5
Script Example .....	3-7

Defining and Using Symbols . . . . .	3-9
Creating a Step That Executes a Program . . . . .	3-10
Creating a Procedure . . . . .	3-11
Using a Procedure . . . . .	3-13
Modifying a Procedure at Execution Time . . . . .	3-13
Controlling a Script's Behavior . . . . .	3-16
Conditioning the Execution of a Step . . . . .	3-16
Controlling the Execution Flow . . . . .	3-18
Changing Default Error Messages . . . . .	3-19
Using Files . . . . .	3-19
Creating a File Definition . . . . .	3-19
Assigning and Using Files . . . . .	3-20
Using a Generation File (GDG) . . . . .	3-21
Using an In-Stream File . . . . .	3-23
Using a Set of Concatenated Files . . . . .	3-23
Using an External "sysin" . . . . .	3-24
Deleting a File . . . . .	3-24
RDB Files . . . . .	3-24
Using an RDBMS Connection . . . . .	3-25
Submitting a Job using INTRDR facility . . . . .	3-27
Submitting a Job With EJR . . . . .	3-27
LOG File Structure . . . . .	3-27
Using Batch Runtime With a Job Scheduler . . . . .	3-29
Executing an SQL request . . . . .	3-30
Simple Application on COBOL-IT / BDB . . . . .	3-30

## Best Practices

Adapting z/OS Capabilities on a UNIX/Linux Environment . . . . .	4-1
--	-----

Defining Paths for Procedures, Includes and Programs . . . . .	4-1
Prohibiting the Use of UNIX Commands . . . . .	4-2
Avoiding the Use of File Overriding . . . . .	4-2

## Using Tuxedo Job Enqueueing Service (TuxJES)

Overview. . . . .	5-1
Requirements. . . . .	5-1
TuxJES Components . . . . .	5-2
Configuring a TuxJES System . . . . .	5-2
Setting up TuxJES as an Oracle Tuxedo Application . . . . .	5-2
Setting up TuxJES in MP mode. . . . .	5-7
Using TuxJES . . . . .	5-7
Submitting a Job . . . . .	5-7
Displaying Job Information. . . . .	5-8
Holding a Job. . . . .	5-9
Releasing a Job . . . . .	5-9
Canceling a Job . . . . .	5-9
Purging a Job . . . . .	5-10
Displaying/Changing ARTJESINITIATOR Configuration. . . . .	5-10
Event Subscribing/Unsubscribing . . . . .	5-10
See Also . . . . .	5-11



# Introduction

## Purpose

The aim of the following guide is to help users understand and write Korn-Shell scripts to be used with the Batch Runtime, and how to use Tuxedo Job Enqueueing Service (TuxJES).

The guide covers the usual tasks that are performed within Korn-Shell scripts, whether they are the result of a conversion from z/OS JCL or newly written for the target platform. The guide also covers the usage of TuxJES.

## Organization

This guide is divided into four main chapters:

- **Overview of the Batch Runtime:** This chapter introduces the general principles of the Batch Runtime.
- **Using the Batch Runtime:** This chapter describes, through various examples, how to perform the usual tasks required to implement the Batch Runtime. This section describes how the different Oracle Tuxedo Application Runtime for Batch high-level functions can be assembled in order to create a single "step", and then how the different steps are assembled in order to create a complete Korn shell script.
- **Best Practices:** This chapter provides guidance for preserving z/OS capabilities on the target platform.
- **Using TuxJES:** This chapter provides guidance for configuring and executing TuxJES.

## See Also

For more information about Batch Runtime, specifically on how to code the different functions, see [Oracle Tuxedo Application Runtime Reference Guide](#).



# Overview of the Batch Runtime Environment

## Oracle Tuxedo Application Runtime for Batch Presentation and Structure

The purpose of the Batch Runtime is to provide functions enabling a robust production environment on a UNIX/Linux platform.

Oracle Tuxedo Application Runtime for Batch is composed of:

- Technical functions
- High-level functions
- Interface-level functions

### Technical Functions

The technical level contains simple one-action functions: easy to write, easy to maintain and easy to debug. For example, GDG (Generation Data Group) management belongs to this level. This technical level is the robust base of the Batch Runtime.

### High-Level Functions

The high-level functions provide entry points to the Batch Runtime. This level homogenizes the behavior of functions, in order for them to be called in a production script. A high-level function follows a skeleton which provide robust logical workflow (execution on/off, options check, predefined return codes ...).

In this level, we find functions to:

- Manage files (creation, copy, assignation...)
- Launch programs (COBOL, executable ...)
- Access Databases (connection/disconnection/commit/rollback for program, SQL execution)
- Produce reports
- Run utilities

## Interface-Level Functions

The interface level allow users to interact with the Batch Runtime job management: submission, holding and releasing, class management, reporting, monitoring ...

Oracle Tuxedo Application Runtime for Batch offers robust and useful production functions. With these functions, you can easily emulate JCL and JES2 features, and have extra features like "no exec mode", return code predefinition (customizable), internationalization.

Oracle Tuxedo Application Runtime for Batch uses a native shell interpreter for high level functions. This approach enables you to add new runtime functions for specific production needs

## Script Execution Phases

When submitted for execution within the Batch Runtime, a Korn shell script is processed through three separate phases:

### Input Phase

In this phase, the JOB card parameters are analyzed.

### Conversion Phase

During this phase, the Batch Runtime performs the following actions:

- Expand all the external Korn shell scripts (procedures and/or includes) that are used within the script so as to produce a single complete script.
- Resolve all the symbols that are used in the script replacing them by their current values.

### Execution Phase

The script is executed by the Batch Runtime.

# Using Batch Runtime

This chapter contains the following topics:

- [Configuration Files](#)
- [Setting Environment Variables](#)
- [Configuring Batch Runtime in MP Mode](#)
- [Creating a Script](#)
- [Controlling a Script's Behavior](#)
- [Different Behaviors from z/OS](#)
- [Using Files](#)
- [Submitting a Job Using INTRDR Facility](#)
- [Submitting a Job With EJR](#)
- [User-Defined Entry/Exit](#)
- [Batch Runtime Logging](#)
- [Using Batch Runtime With a Job Scheduler](#)
- [Executing an SQL Request](#)
- [Simple Application on COBOL-IT / BDB](#)
- [Dynamic JCL Job Execution](#)

- [Network Job Entry \(NJE\) Support](#)
- [File Catalog Support](#)

## Configuration Files

The Configuration files are implemented in the directory CONF of the RunTime Batch.

### BatchRT.conf

This file contains variables definition.

These variables must be set before using the RunTime Batch.

### Messages.conf

This file contains messages used by RTBatch.

The messages may be translated in a local language.

### FunctionReturnCode.conf

This file contains internal codes associated with a message.

### ReturnCode.conf

This file contains return codes associated with a message and returned to the KSH script.

## Setting Environment Variables

Some variables (such as ORACLE\_SID, COBDIR, LIBPATH, COBPATH ...) are shared variables between different components and are not described in this current document.

For more information, see [Rehosting Workbench Installation Guide](#).

[Table 3-1](#) lists the environment variables that are used in the KSH scripts and must be defined before using the software.

**Table 3-1 KSH Script Environment Variables**

Variable	Usage
DATA	Directory for permanent files.
TMP	Directory for temporary application files.
SYSIN	Directory where the sysin are stored.
MT_JOB_NAME	Name of the job, managed by the Batch Runtime.
MT_JOB_PID	PID (process id) of the job, managed by the Batch Runtime.

[Table 3-2](#) lists the environment variables that are used by Batch Runtime and must be defined before using the software.

**Table 3-2 Oracle Tuxedo Application Runtime for Batch Environment Variables**

Variable	Usage
PROCLIB	Directory for PROC and INCLUDE files, used during the conversion phase.
MT_ACC_FILEPATH	File concurrency access, directory that contains the files AccLock and AccWait. These files must be created empty before running the Batch Runtime (see the <a href="#">BatchRT.conf</a> configuration file).
MT_COBOL	Depending on the used COBOL, must contain: <ul style="list-style-type: none"> <li>- “COBOL_MF” for MicroFocus</li> <li>- “COBOL_IT” for CobolIT</li> <li>- “COBOL_NONE” if users neither have any COBOL programs to run nor use any COBOL product; besides, with this setting, only GDG, LSEQ, Fixed length SEQ, and PDS files are supported.</li> </ul> (See the <a href="#">BatchRT.conf</a> configuration file)
MT_CTL_FILES	Directory where the control file (CTL) used by the function m_DBTableLoad (sqlldr with ORACLE, load and export with UDB).

**Table 3-2 Oracle Tuxedo Application Runtime for Batch Environment Variables**

Variable	Usage
MT_DB	Depending on the target data base, must contain : - “DB_ORACLE” for ORACLE - “DB_DB2LUW” for UDB (See the <a href="#">BatchRT.conf</a> configuration file)
MT_DB_LOGIN	Database connection user.
MT_FROM_ADDRESS_MAIL	From-Address used by the function m_SendMail when the option “-f” is omitted.
MT_FTP_TEST	Variable used by the function m_Ftp to do the tranfer or not (test mode).
MT_GENERATION	A mandatory environment variable which indicates the directory to GDG technical functions.  The default is directory GENERATION_FILE. To manage GDG files in database, you need to set the value to GENERATION_FILE_DB and configure MT_GDG_DB_ACCESS appropriately. If the value is specified as NULL or with an incorrect directory name, error occurs when using this environment variable.
MT_KSH	Path of the used “ksh” (pdksh or ksh88 only)  <b>Note:</b> For more information about pdksh, please refer to <a href="http://www.cs.mun.ca/~michael/pdksh/">http://www.cs.mun.ca/~michael/pdksh/</a> .
MT_LOG	Logs directory (without TuxJes).
MT_ROOT	Directory where the Batch Runtime application has been installed. (See the <a href="#">BatchRT.conf</a> configuration file)
MT_SMTP_PORT	Port used by the functions m_Smtp and m_SendMail (localhost by default).
MT_SMTP_SERVER	Server used by the functions m_Smtp and m_SendMail (25 by default).

**Table 3-2 Oracle Tuxedo Application Runtime for Batch Environment Variables**

Variable	Usage
MT_SORT	Depending on the used SORT, must contain: - “SORT_MicroFocus” for MicroFocus Sort Utility - “SORT_SyncSort” for SyncSort Sort Utility - “SORT_CIT” for citsort utility (See the <a href="#">BatchRT.conf</a> configuration file)
MT_SYSOUT	Sysout directory (without TuxJes).
MT_TMP	Directory for temporary internal files (See the <a href="#">BatchRT.conf</a> configuration file).
MT_EXCI	EXCI Interface (Default is Oracle Tuxedo). (See the <a href="#">BatchRT.conf</a> configuration file)
MT_JESDECRYPT	MT_JESDECRYPT must be set to jesdecrypt object file. (See the <a href="#">BatchRT.conf</a> configuration file)
MT_EXCI_XA	Name of the resource manager for XA. (See the <a href="#">BatchRT.conf</a> configuration file)
MT_EXCIGRPNAME	TUXEDO SRVGRP value of the ARTDPL server. (See the <a href="#">BatchRT.conf</a> configuration file)

[Table 3-3](#) lists optional environment variables used by Batch Runtime.

**Table 3-3 Oracle Tuxedo Application Runtime for Batch Environment Variables (Optional)**

Variable	Usage
MT_ACC_WAIT	Retry interval (seconds) to acquire file lock when a job tries to access a file that locked by other jobs.
MT_ACC_MAXWAIT	Maximum wait time (seconds) to acquire file lock. If the lock is not acquired within such time, relevant file operation will fail.

**Table 3-3 Oracle Tuxedo Application Runtime for Batch Environment Variables (Optional)**

Variable	Usage
MT_UTILITY_LIST_UNSU PPORT	<p>A list of executable programs, programs that don't exist but users don't want to fail any jobs because of them. When <code>m_ProgramExec</code> invokes nonexistent programs, JOB will continue if those programs are specified in this list. For example:</p> <pre>MT_UTILITY_LIST_UNSUPPORT=IEHINITT, IEHLIST, IEHMOVE, IEH STATR, IEHPROGM, IEBCOMPR, IEBEDIT, IEBIMAGE, IEBUPDTE, IEBD G, IEBTPCH</pre>
MT_EXCI_PGM_LIST	<p>A list of executable programs. The programs are invoked by <code>runbexci</code> instead of <code>runb</code>. For each program in this list, whether or not <code>-n</code> is specified by <code>m_ProgramExec</code>, the program is invoked only by <code>runbexci</code>.</p> <p>The default value is empty; programs are separated by commas. For example:</p> <ul style="list-style-type: none"> <li>• <code>MT_EXCI_PGM_LIST=PGM1, PGM2</code></li> </ul>
MT_GDG_DB_ACCESS	<p>A variable used with valid database login information to access Oracle Database for GDG management. For example, <code>user/password@sid</code>.</p> <p><b>Note:</b> It's mandatory if <code>MT_GENERATION</code> is set to <code>GENERATION_FILE_DB</code>.</p>
MT_GDG_USEDCB	<p>A variable used to enable DCB support function for GDG. For example:</p> <ul style="list-style-type: none"> <li>• <code>MT_GDG_USEDCB=Y</code>: Create <code>.dcb</code> file for GDG (default behavior)</li> <li>• <code>MT_GDG_USEDCB=N</code>: Don't create <code>.dcb</code> file for GDG</li> </ul>
MT_WB_HOSTNAME	<p>The host name (or IP address), where Workbench is installed to be invoked to convert JCL job to KSH job. The value of <code>MT_WB_HOSTNAME</code> is null if Workbench is in localhost. User name is optional to be added. For example:</p> <ul style="list-style-type: none"> <li>• <code>MT_WB_HOSTNAME=host1</code>: Set the value of <code>MT_WB_HOSTNAME</code> to <code>host1</code></li> <li>• <code>MT_WB_HOSTNAME=user1@host1</code>: Set the value of <code>MT_WB_HOSTNAME</code> to <code>user1@host1</code></li> </ul> <p><b>Note:</b> It is required to be set if Workbench is deployed on the remote machine while ARTJESCONV server is deployed on another machine.</p>



**Table 3-3 Oracle Tuxedo Application Runtime for Batch Environment Variables (Optional)**

Variable	Usage
MT_REFINEDIR	The full install path of Workbench <code>refine</code> , which will be invoked to convert a JCL job to a KSH job. For example: <ul style="list-style-type: none"><li>• <code>MT_REFINEDIR=/newspace/public/WB_Test/wb12110/refine</code></li></ul>
MT_REFINEDISTRIB	The value of environment variable <code>REFINEDISTRIB</code> , which is used when Workbench converts a JCL job. For example: <ul style="list-style-type: none"><li>• <code>MT_REFINEDISTRIB = Linux64</code>: Set <code>REFINEDISTRIB</code> to <code>Linux64</code></li><li>• <code>MT_REFINEDISTRIB = Linux32</code>: Set <code>REFINEDISTRIB</code> to <code>Linux32</code></li></ul>

## Configuring Batch Runtime in MP Mode

Batch Runtime (EJR) will need to be specially configured so as to work well in MP mode if users want to either use EJR to run jobs, which may share resources (normally files), from different machines or configure a MP mode TuxJES domain and submit jobs from any node through the utility provided by TuxJES.

In the latter case, the job submitted from node A may be run by node B and the execution sequence is totally random. Similarly, these jobs submitted from different nodes may share resources.

This section clarifies the details of configuring Batch Runtime (EJR) to support MP mode.

1. All the resources should be put on a shared storage (NFS), which should have the same mount point on all machines in the domain, to ensure any file has the same path from the view of each node, because any job submitted from one machine may be run by another machine. For example, if users prefer to store all files under environment variable `DATA` described in above section, `${DATA}` should point to the shared root directory where files are located and have the same value on all machines.
2. `MT_ACC_FILE_PATH` should be located on shared storage (NFS), which should have same mount point on all machines in the domain, since the control files for file locking are put in this directory; in addition, users need to make sure `AccLock` and `AccWait` files under this directory can be read / written by the effective user of the process running the jobs.
3. NLM (Network Lock Manager) needs to be enabled on the NFS server and all machines in the domain since some shared resources, which are located on NFS, need to be locked to prevent jobs from corrupting them. The configuration is not directly related to Batch Runtime but has close relationship in MP mode.

4. ARTJESADM server should be configured and started on each node in the MP domain to check, by other nodes, whether a job on this node is running or not. This is a part of the file lock mechanism in Batch Runtime. If either ARTJESADM server on one node dies abnormally or the node itself dies abnormally, the file lock owned by the job running on this node won't be released automatically; in this case, the utility `artjescleanlock` can be used to release the inactive file lock. For details of `artjescleanlock`, see [Using Tuxedo Job Enqueueing Service \(TuxJES\)](#).

## Creating a Script

### General Structure of a Script

Oracle Tuxedo Application Runtime for Batch normalizes Korn shell script formats by proposing a script model where the different execution phases of a job are clearly identified.

Oracle Tuxedo Application Runtime for Batch scripts respect a specific format that allows the definition and the chaining of the different phases of the KSH (JOB).

Within Batch Runtime, a *phase* corresponds to an activity or a step on the source system.

A phase is identified by a label and delimited by the next phase.

At the end of each phase, the `JUMP_LABEL` variable is updated to give the label of the next phase to be executed.

In the following example, the last functional phase sets `JUMP_LABEL` to `JOBEND`: this label allows a normal termination of the job (exits from the phase loop).

The mandatory parts of the script (the beginning and end parts) are shown in bold and the functional part of the script (the middle part) in normal style as shown in [Table 3-4](#). The optional part of the script must contain the labels, branching and end of steps as described below. The items of the script to be modified are shown in italics.

**Table 3-4 Script Structure**

<b>Script</b>	<b>Description</b>
<code>#!/bin/ksh#</code>	
<code>m_JobBegin -j JOBNAME -s START -v 2.00</code>	m_JobBegin is mandatory and must contain at least the following options: <ul style="list-style-type: none"> <li>• -j: internal job name</li> <li>• -s: name of the first label to begin execution (usually should be START)</li> <li>• -v: Minimum version number of Batch Runtime required for this script (upward compatible).</li> </ul>
<code>while true ;do</code>	The "while true; do" loop provides a mechanism to simulate the movement from one step to the next.
<code>m_PhaseBegin</code>	m_PhaseBegin enables parameters to be initialized at the beginning of a step.
<code>case \${CURRENT_LABEL} in</code>	The case statement enables a branching to the current step.
<code>(START)</code>	The start label (used in the -s option of m_JobBegin)
<code>JUMP_LABEL=STEP1</code>	JUMP_LABEL is mandatory in all steps and gives the name of the next step.
<code>;;</code>	;; ends a step and are mandatory.
<code>(STEP1)</code>	A functional step begins with (LABEL); where LABEL is the name of the step.
<code>m_* m_*</code>	A typical step continues with a series of calls to Batch Runtime functions.
<code>JUMP_LABEL=STEP2</code>	There is always a branching to the next step (JUMP_LABEL=)
<code>;;</code>	And always the ;; at the end of each step.
<code>(PENULTIMATESTEP)</code>	
<code>m_* m_*</code>	The last functional step has the same format as the others, except...
<code>JUMP_LABEL=END_JOB ;; (END_JOB)</code>	For the label, which must point to END_JOB. The _ is necessary, because the character is forbidden on z/OS.

**Table 3-4 Script Structure**

Script	Description
<pre>break ;; (*)</pre>	This step enables the processing loop to be broken.
<pre>m_RcSet \${MT_RC_ABORT:-S999} "Unknown label : \${CURRENT_LABEL}" break ;; esac</pre>	This is a catch-all step that picks-up branching to unknown steps.
<b>m_PhaseEnddone</b>	m_PhaseEnd manages the end of a step including file management depending on disposition and return codes.
<b>m_JobEnd</b>	m_JobEnd manages the end of a job including clearing-up temporary files and returning completion code to job caller.

## Script Example

[Listing 3-1](#) shows a Korn shell script example.

**Listing 3-1 Korn shell Script Example**

```
#!/bin/ksh
#@(#)-----
#@(#) -
m_JobBegin -j METAW01D -s START -v 1.00 -c A
while true ;
do
    m_PhaseBegin
    case ${CURRENT_LABEL} in
(START)
```

```

# -----
# 1) 1st Step: DELVCUST
#   Delete the existing file.
# 2) 2nd Step: DEFVCUST
#   Allocates the Simple Sample Application VSAM customers file
# -----
#
# -Step 1: Delete...
      JUMP_LABEL=DELVCUST
      ;;
(DELVCUST)
      m_FileAssign -d OLD FDEL ${DATA}/METAW00.VSAM.CUSTOMER
      m_FileDelete ${DD_FDEL}
      m_RcSet 0
#
# -Step 2: Define...
      JUMP_LABEL=DEFVCUST
      ;;
(DEFVCUST)
# IDCAMS DEFINE CLUSTER IDX
      m_FileBuild -t IDX -r 266 -k 1+6 ${DATA}/METAW00.VSAM.CUSTOMER
      JUMP_LABEL=END_JOB
      ;;
(ABORT)
      break
      ;;
(END_JOB)

```

## Using Batch Runtime

```
        break
    ;;
(*)
    m_RcSet ${MT_RC_ABORT} "Unknown label : ${JUMP_LABEL}"
    break
    ;;
esac
m_PhaseEnd
done
m_JobEnd
#@(#)-----
```

---

## Defining and Using Symbols

Symbols are internal script variables that allow script statements to be easily modifiable. A value is assigned to a symbol through the `m_SymbolSet` function as shown in [Listing 3-2](#). To use a symbol, use the following syntax: `${symbol}`

**Note:** The use of brackets ([]) instead of braces ({} ) is to clearly distinguish symbols from standard Korn shell variables.

### Listing 3-2 Symbol Use Examples

---

```
(STEP00)
    m_SymbolSet VAR=40
    JUMP_LABEL=STEP01
    ;;
(STEP01)
    m_FileAssign -d SHR FILE01 ${DATA}/PJ01DDD.BT.QSAM.KBST00${VAR}
    m_ProgramExec BAI001
```

---

## Creating a Step That Executes a Program

A step (also called a phase) is generally a coherent set of calls to Batch Runtime functions that enables the execution of a functional (or technical) activity.

The most frequent steps are those that execute an application or utility program. These kind of steps are generally composed of one or several file assignment operations followed by the execution of the desired program. All the file assignments operations must precede the program execution operation shown in [Listing 3-3](#)

### Listing 3-3 Application Program Execution Step Example

---

```
(STEPPR15)

    m_FileAssign -d SHR INFIL ${DATA}/PJ01DDD.BT.QSAM.KBPRO099
    m_FileAssign -d MOD OUTFIL ${DATA}/PJ01DDD.BT.QSAM.KBPRO001
    m_OutputAssign -c "*" SYSOUT
    m_FileAssign -i LOGIN

IN-STREAM DATA

_end

    m_FileAssign -d MOD LOGOUT ${DATA}/PJ01DDD.BT.QSAM.KBPRO091
    m_ProgramExec BPRAB001 "20071120"

    JUMP_LABEL=END_JOB

;;
```

---

## Application Program Abend Execution

ILBOABNO, an abend routine, can be called from a running program to force it to abort and return the abend code to KSH script. ILBOABNO is supplied as both source and binary gnt file. It can be called directly by any user-defined cobol program.

### Listing 3-4 Application Program Abend Execution Example (KSH)

---

```
(STEPPR15)

m_ProgramExec USER
JUMP_LABEL=END_JOB

;;
```

---

### Listing 3-5 USER.cbl Example

---

```
PROCEDURE DIVISION.

PROGRAM-BEGIN.

    DISPLAY "USER: HELLO USER" .

    MOVE 2 TO RT-PARAM.

    CALL "ILBOABNO" USING RT-PARAM.

    DISPLAY "USER: CAN'T REACH HERE WHEN ILBOABNO IS CALLED" .

PROGRAM-DONE.

...
```

---

## Creating a Procedure

Oracle Tuxedo Application Runtime for Batch offers a set of functions to define and use "procedures". These procedures follow generally the same principles as z/OS JCL procedures.

The advantages of procedures are:

- Write a set of tasks once and use it several times.
- Make this set of tasks dynamically modifiable.



Procedures can be of two types:

- **In-stream Procedures:** Included in the calling script, this kind of procedure can be used only in the current script.
- **External Procedures:** Coded in a separate source file, this kind of procedure can be used in multiple scripts.

## Creating an In-Stream Procedure

Unlike the z/OS JCL convention, an in-stream procedure must be written after the end of the main JOB, that is: all the in-stream procedures belonging to a job must appear after the call to the function `m_JobEnd`.

An in-stream procedure in a Korn shell script always starts with a call to the `m_ProcBegin` function, followed by all the tasks composing the procedure and terminating with a call to the `m_ProcEnd` function. [Listing 3-6](#) is an example.

### Listing 3-6 In-stream Procedure Example

---

```
m_ProcBegin  PROCA
            JUMP_LABEL=STEPA
            ;;
(STEPA)
            m_FileAssign -c "*" SYSPRINT
            m_FileAssign -d SHR SYSUT1
            ${DATA}/PJ01DDD.BT.DATA.PDSA/BIEAM00${SEQ}
            m_FileAssign -d MOD SYSUT2 ${DATA}/PJ01DDD.BT.QSAM.KBIE0005
            m_FileLoad ${DD_SYSUT1} ${DD_SYSUT2}
            JUMP_LABEL=ENDPROC
            ;;
(ENDPROC)
m_ProcEnd
```

---

## Creating an External Procedure

External procedures do not require the use of the `m_ProcBegin` and `m_ProcEnd` functions; simply code the tasks that are part of the procedure shown in [Listing 3-7](#).

In order to simplify the integration of a procedure's code with the calling job, always begin a procedure with:

```
JUMP_LABEL=FIRSTSTEP  
  
;;  
  
(FIRSTSTEP)
```

and end it with:

```
JUMP_LABEL=ENDPROC  
  
;;  
  
(ENDPROC)
```

### Listing 3-7 External Procedure Example

---

```
JUMP_LABEL=PR2STEP1  
  
;;  
  
(PR2STEP1)  
  
m_FileAssign -d SHR INFIL ${DATA}/PJ01DDD.BT.QSAM.KBPRI001  
m_FileAssign -d MOD OUTFIL ${DATA}/PJ01DDD.BT.QSAM.KBPRO001  
m_OutputAssign -c "*" SYSOUT  
m_FileAssign -d SHR LOGIN ${DATA}/PJ01DDD.BT.SYSIN.SRC/BPRAS002  
m_FileAssign -d MOD LOGOUT ${DATA}/PJ01DDD.BT.QSAM.KBPRO091  
m_ProgramExec BPRAB002  
JUMP_LABEL=ENDPROC  
  
;;  
  
(ENDPROC)
```

---

## Using a Procedure

The use of a procedure inside a Korn shell script is made through a call to the `m_ProcInclude` function.

As described in [Script Execution Phases](#), during the Conversion Phase, a Korn shell script is expanded by including the procedure's code each time a call to the `m_ProcInclude` function is encountered. It is necessary that after this operation, the resulting expanded Korn shell script still respects the rules of the general structure of a script as defined in the [General Structure of a Script](#).

A procedure, either in-stream or external, can be used in any place inside a calling job provided that the above principals are respected shown in [Listing 3-8](#).

### Listing 3-8 Call to the `m_ProcInclude` Function Example

---

```
...  
(STEPPR14)  
    m_ProcInclude BPRAP009  
    JUMP_LABEL=STEPPR15  
...
```

---

## Modifying a Procedure at Execution Time

The execution of the tasks defined in a procedure can be modified in two different ways:

- Modifying symbols and/or parameters
- Symbols can be used inside a procedure and the values of these symbols can be specified when calling the procedure.

[Listing 3-9](#) and [Listing 3-10](#) are examples.

### Listing 3-9 Defining Procedure Example

---

```
m_ProcBegin PROCEDURE  
    JUMP_LABEL=STEPE
```

## Using Batch Runtime

```
;;
(STEPE)
  m_FileAssign -d SHR SYSUT1 ${DATA}/DATA.IN.PDS/DTS${SEQ}
  m_FileAssign -d MOD SYSUT2 ${DATA}/DATA.OUT.PDS/DTS${SEQ}
  m_FileLoad ${DD_SYSUT1} ${DD_SYSUT2}
  JUMP_LABEL=ENDPROC
;;
(ENDPROC)
m_ProcEnd
```

---

### Listing 3-10 Calling Procedure Example

---

```
(COPIERE)
  m_ProcInclude PROCE SEQ="1"
  JUMP_LABEL=COPIERF
;;
```

---

## Using Overrides for File Assignments

As specified in [Best Practices](#), this way of coding procedures is provided mainly for supporting Korn shell scripts resulting from z/OS JCL translation and it is not recommended for Korn shell scripts newly written for the target platform.

The overriding of a file assignment is made using the `m_FileOverride` function that specifies a replacement for the assignment present in the procedure. The call to the `m_FileOverride` function must follow the call to the procedure in the calling script.

[Listing 3-11](#) shows how to replace the assignment of the logical file `SYSUT1` using the `m_FileOverride` function.

### Listing 3-11 m\_FileOverride Function Example

---

```
m_ProcBegin PROCEDURE
    JUMP_LABEL=STEPE
    ;;
(STEPE)
    m_FileAssign -d SHR SYSUT1 ${DATA}/DATA.IN.PDS/DTSS${SEQ}
    m_FileAssign -d MOD SYSUT2 ${DATA}/DATA.OUT.PDS/DTSS${SEQ}
    m_FileLoad ${DD_SYSUT1} ${DD_SYSUT2}
    JUMP_LABEL=ENDPROC
    ;;
(ENDPROC)
m_ProcEnd
```

---

### Listing 3-12 m\_FileOverride Procedure Call:

---

```
(COPIERE)
    m_ProcInclude PROCEDURE SEQ="1"
    m_FileOverride -i -s STEPE SYSUT1
Overriding test data
_end
    JUMP_LABEL=COPIERF
    ;;
```

---

# Controlling a Script's Behavior

## Conditioning the Execution of a Step

### Using `m_CondIf`, `m_CondElse`, and `m_CondEndif`

The `m_CondIf`, `m_CondElse` and `m_CondEndif` functions can be used to condition the execution of one or several steps in a script. The behavior is similar to the z/OS JCL statement constructs `IF`, `THEN`, `ELSE` and `ENDIF`.

The `m_CondIf` function must always have a relational expression as a parameter as shown in [Listing 3-13](#). These functions can be nested up to 15 times.

#### Listing 3-13 `m_CondIf`, `m_CondElse`, and `m_CondEndif` Example

---

...

```
(STEPIF01)
    m_FileAssign -d SHR INFIL ${DATA}/PJ01DDD.BT.QSAM.KBIF000
    m_FileAssign -d MOD OUTFIL ${DATA}/PJ01DDD.BT.QSAM.KBIF001
    m_ProgramExec BAX001
    m_CondIf "STEPIF01.RC,LT,5"
    JUMP_LABEL=STEPIF02
    ;;
(STEPIF02)
    m_FileAssign -d SHR INFIL ${DATA}/PJ01DDD.BT.QSAM.KBIF001
    m_FileAssign -d MOD OUTFIL ${DATA}/PJ01DDD.BT.QSAM.KBIF002
    m_ProgramExec BAX002
    m_CondElse
    JUMP_LABEL=STEPIF03
    ;;
(STEPIF03)
```

```
m_FileAssign -d SHR INFIL ${DATA}/PJ01DDD.BT.QSAM.KBIF000
m_FileAssign -d MOD OUTFIL ${DATA}/PJ01DDD.BT.QSAM.KBIF003
m_ProgramExec BAX003
m_CondEndif
```

---

## Using m\_CondExec

The `m_CondExec` function is used to condition the execution of a step. The `m_CondExec` must have at least one condition as a parameter and can have several conditions at the same time. In case of multiple conditions, the step is executed only if all the conditions are satisfied.

A condition can be of three forms:

- Relational expression testing previous return codes:

```
m_CondExec 4,LT,STEPEC01
```

- **EVEN**: Indicates that the step is to be executed even if a previous step terminated abnormally:

```
m_CondExec EVEN
```

- **ONLY**: Indicates that the step is to be executed only if a previous step terminated ab-normally:

```
m_CondExec ONLY
```

The `m_CondExec` function must be the first function to be called inside the concerned step as shown in [Listing 3-14](#).

### Listing 3-14 m\_CondExec Example with Multiple Conditions

---

...

```
(STEPEC01)
```

```
m_FileAssign -d SHR INFIL ${DATA}/PJ01DDD.BT.QSAM.KBIF000
m_FileAssign -d MOD OUTFIL ${DATA}/PJ01DDD.BT.QSAM.KBIF001
m_ProgramExec BACC01
JUMP_LABEL=STEPEC02
```

```
;;
(STEPEC02)
  m_FileAssign -d SHR INFIL ${DATA}/PJ01DDD.BT.QSAM.KBIF001
  m_FileAssign -d MOD OUTFIL ${DATA}/PJ01DDD.BT.QSAM.KBIF002
  m_ProgramExec BACC02
  JUMP_LABEL=STEPEC03
;;
(STEPEC03)
  m_CondExec 4,LT,STEPEC01 8,GT,STEPEC02 EVEN
  m_FileAssign -d SHR INFIL ${DATA}/PJ01DDD.BT.QSAM.KBIF000
  m_FileAssign -d MOD OUTFIL ${DATA}/PJ01DDD.BT.QSAM.KBIF003
```

---

## Controlling the Execution Flow

The script's execution flow is determined, and can be controlled, in the following ways:

- The start label specified by the `m_JobBegin` function: this label is usually the first label in the script, but can be changed to any label present in the script if the user wants to start the script execution from a specific step.
- The value assigned to the `JUMP_LABEL` variable in each step: this assignment is mandatory in each step, but its value is not necessarily the label of the following step.
- The usage of the `m_CondExec`, `m_CondIf`, `m_CondElse` and `m_CondEndif` functions: see [Conditioning the Execution of a Step](#).
- The return codes and abnormal ends of steps.

## Changing Default Error Messages

If Batch Runtime administrator wishes to change the default messages (to change the language for example), this can be done through a configuration file whose path is specified by the environment variable: `MT_DISPLAY_MESSAGE_FILE`.



This file is a CSV (comma separated values) file with a semicolon as a separator. Each record in this file describes a certain message and is composed of 6 fields:

1. Message identifier.
2. Functions that can display the message (can be a generic name using '\*').
3. Level of display.
4. Destination of display.
5. Reserved for future use.
6. Message to be displayed.

## Different Behaviors from z/OS

On z/OS, before one job is executed, JES checks its syntax. If any error is found, JES reports it and runs nothing of the job. For example, if there is a JCL statement applying “NEW” on generation(0) of a GDG, because NEW is not allowed to be applied to existing files, JES reports this error and does not run the job.

However, in ART for Batch, JCL job is converted to ksh job by Oracle Tuxedo ART Workbench at first, and ART for Batch only checks ksh script syntax in the converted ksh job. Grammar errors, if any, are detected when this statement runs, resulting in the fact that statements after the wrong statement are executed but statements before it are executed without being affected.

## Using Files

### Creating a File Definition

Files are created using the `m_FileBuild` or the `m_FileAssign` function.

Four file organizations are supported:

- Sequential file
- Line sequential file
- Relative file
- Indexed file

You must specify the file organization for the file being created. For indexed files, the length and the primary key specifications must also be mentioned.

## m\_FileBuild Examples

- Definition of a line sequential file

```
m_FileBuild -t LSEQ ${DATA}/PJ01DDD.BT.VSAM.ESDS.KBIDO004
```

- Definition of an indexed file with a record length of 266 bytes and a key starting at the first bytes and having a size of 6 bytes.

```
m_FileBuild -t IDX -r 266 -k 1+6 ${DATA}/METAW00.VSAM.CUSTOMER
```

## m\_FileAssign examples

- Definition of a new sequential file with a record length of 80 bytes.

```
m_FileAssign -d NEW -t SEQ -r 80 ${DATA}/PJ01DDD.BT.VSAM.ESDS.KBIDO005
```

## Assigning and Using Files

When using Batch Runtime, a file can be used either by a Batch Runtime function (for example: `m_FileSort`, `m_FileRename` etc.) or by a program, such as a COBOL program.

In both cases, before being used, a file must first be assigned. Files are assigned using the `m_FileAssign` function that:

- Specifies the DISP mode (Read or Write)
- Specifies if the file is a generation file
- Defines an environment variable linking the logical name of the file (IFN) with the real path to the file (EFN).

The environment variable defined via the `m_FileAssign` function is named: `DD_IFN`. This naming convention is due to the fact that it is the one used by Micro Focus Cobol to map internal file names to external file names.

Once a file is assigned, it can be passed as an argument to any of Batch Runtime functions handling files by using the `${DD_IFN}` variable.

For COBOL programs, the link is made implicitly by Micro Focus Cobol.

### Listing 3-15 Example of File Assignment

---

```
(STEP01)
    m_FileAssign -d SHR INFIL ${DATA}/PJ01DDD.BT.QSAM.KBIDI001
    m_FileAssign -d SHR OUTFIL ${DATA}/PJ01DDD.BT.VSAM.KBIDU001
    m_FileLoad ${DD_INFIL} ${DD_OUTFIL}
...

```

---

### Listing 3-16 Example of Using a File by a COBOL Program

---

```
(STEP01)
    m_FileAssign -d OLD INFIL ${DATA}/PJ01DDD.BT.QSAM.KBIFI091
    m_FileAssign -d MOD OUTFIL ${DATA}/PJ01DDD.BT.QSAM.KBIFO091
    m_ProgramExec BIFAB090
...

```

---

## Concurrent File Accessing Control

Batch Runtime provides a lock mechanism to prevent one file from being written simultaneously in two jobs.

To enable the concurrent file access control, do the following:

1. Use environment variable `MT_ACC_FILEPATH` to specify a directory for the lock files required by concurrent access control mechanism.
2. Create two empty files, `AccLock` and `AccWait`, under the directory specified in step 1.

Make sure the effective user executing jobs has read/write permission to these two files.

#### Notes:

- The file names of `AccLock` and `AccWait` are case sensitive.

- When accessing generation files, a GDG rather than a generation file is locked. That is, a GDG is locked as a whole.
- Following two lines in `ejr/CONF/BatchRT.conf` should be commented out:

```
${MT_ACC_FILEPATH}/AccLock  
${MT_ACC_FILEPATH}/AccWait
```

## Using Generation Data Group (GDG)

Oracle Tuxedo Application Runtime for Batch allows you to manage Generation Data Group (GDG) files either based on file or based on database (DB). In file-based management way, Batch Runtime manages GDG files in separate `*.gens` files, and one `*.gens` corresponds to one GDG file. In DB-based management way, ART for Batch allows users to manage GDG information in Oracle database or DB2 database.

### GDG Management Functionalities

In order to emulate the notion of generation files and present on the z/OS mainframe which is not a UNIX standard, Batch Runtime provides a set of functions to manage this type of file. These functions are available to both file-based management and DB-based management.

**Note:** Copying or Renaming GDG is not supported.

### Defining and/or Redefining a GDG

It is required to define a GDG before using it.

A GDG file is defined and/or redefined through `m_GenDefine`. The operation of defining or redefining a GDG is committed immediately and cannot be rolled back.

As shown in [Listing 3-17](#), the first line defines a GDG and sets its maximum generations to 15, the second line redefines the same GDG maximum generations to 30, the third line defines a GDG without specifying `-s` option (its maximum generations is set to 9999), the fourth line defines a GDG implicitly and sets its maximum generations to 9999, the fifth line defines a GDG use model file `$DATA/FILE`, which can be either a GDG file or a normal file.

#### Listing 3-17 Example of Defining and Redefining GDG Files

---

```
m_GenDefine -s 15 ${DATA}/PJ01DDD.BT.FILE1  
m_GenDefine -s 30 -r ${DATA}/PJ01DDD.BT.FILE1
```

```
m_GenDefine ${DATA}/PJ01DDD.BT.FILE2
m_FileAssign -d NEW,CATLG -g +1 SYSUT2 ${DATA}/PJ01DDD.BT.FILE3
m_FileAssign -d NEW,CATLG -g +1 -S $DATA/FILE FILE1 $DATA/GDG
```

---

## Adding Generation Files in a GDG

To add a new generation file (GDS) into a GDG, call `m_FileAssign` with "-d NEW/MOD,..." and "-g +n" parameters. GDS file types can be only LSEQ or SEQ.

There are four key points to add generation files in a GDG.

- Multiple generation files (GDS) can be added in one job or step discontinuously and disorderedly. See [Listing 3-18](#) for an example.
- One generation number (GenNum) can be added only one time in a job. [Listing 3-19](#) shows an incorrect usage.
- The filename of a newly created GDS is generated by the generation number specified in `m_FileAssign` in the format of <current GDS number> + <GenNum>. See [Listing 3-20](#) for an example.
- In a job, if multiple generation files (GDS) are newly created, the GDS with the maximum RGN becomes the *current* GDS after the job finishes. See [Listing 3-21](#) for an example.

Four examples as below elaborate those key points individually.

### Listing 3-18 Example of Adding Multiple Generation Files Discontinuously and Disorderedly

---

```
(STEP1)
m_FileAssign -d NEW,KEEP,KEEP -g +1 SYSUT1 "$DATA/GDG1"
m_FileAssign -d MOD,KEEP,KEEP -g +5 SYSUT2 "$DATA/GDG1"

(STEP2)
m_FileAssign -d NEW,KEEP,KEEP -g +9 SYSUT1 "$DATA/GDG1"
m_FileAssign -d NEW,KEEP,KEEP -g +2 SYSUT2 "$DATA/GDG1"
```

---

The above example adds the following GDS files to GDG.

```
$DATA/GDG1.Gen.0001  
$DATA/GDG1.Gen.0002  
$DATA/GDG1.Gen.0005  
$DATA/GDG1.Gen.0009
```

---

### Listing 3-19 Example of Adding One Generation Number Multiple Times in a Job (Incorrect Usage)

---

```
(STEP1)  
m_FileAssign -d NEW,KEEP,KEEP -g +1 SYSUT1 "$DATA/GDG1"  
m_FileAssign -d NEW,KEEP,KEEP -g +5 SYSUT2 "$DATA/GDG1"  
  
(STEP2)  
m_FileAssign -d NEW,KEEP,KEEP -g +4 SYSUT1 "$DATA/GDG1"  
m_FileAssign -d NEW,KEEP,KEEP -g +5 SYSUT2 "$DATA/GDG1"
```

---

The above example shows an incorrect usage, where generation number (+5) is added two times.

---

### Listing 3-20 Example of Listing GDS Filenames

---

```
m_FileAssign -d NEW,KEEP,KEEP -g +1 SYSUT1 "$DATA/GDG1"  
m_FileAssign -d MOD,KEEP,KEEP -g +5 SYSUT2 "$DATA/GDG1"
```

---

In the above example, suppose \$DATA/GDG1 has three GDS numbered as 1, 2, and 4, respectively. The corresponding GDS files are listed as below.

```
$DATA/GDG1.Gen.0001  
$DATA/GDG1.Gen.0002  
$DATA/GDG1.Gen.0004
```

After the above job runs, \$DATA/GDG1 has five GDS numbered as 1, 2, 4, 5, and 9, respectively. The corresponding GDS files are listed as below.

```

$DATA/GDG1.Gen.0001
$DATA/GDG1.Gen.0002
$DATA/GDG1.Gen.0004
$DATA/GDG1.Gen.0005
$DATA/GDG1.Gen.0009

```

---

### Listing 3-21 Example of Defining the Current GDS

---

```

(STEP1)
m_FileAssign -d NEW,KEEP,KEEP -g +1 SYSUT1 "$DATA/GDG1"
m_FileAssign -d MOD,KEEP,KEEP -g +5 SYSUT2 "$DATA/GDG1"
(STEP2)
m_FileAssign -d NEW,KEEP,KEEP -g +2 SYSUT3 "$DATA/GDG1"

```

---

In the above example, the GDS whose RGN equals +5 becomes the *current GDS*, meaning its RGN becomes 0 after job finishes successfully.

### Referring an Existing Generation Files in a GDG

To refer to an existing generation file (GDS) in a GDG, call `m_FileAssign` with "-d OLD/SHR/MOD,..." and "-g 0", "-g all", or "-g -n" parameters. "-g 0" refers to the current generation, "-g all" refers to all generation files, "-g -n" refers to the generation file which is the nth generation counting backward from the current generation (as 0 generation).

When using relative generation number (RGN) to reference a GDS, note that the "relative generation number" means "relative position with the newest GDS whose generation number is 0".

For example, if GDG1 contains six GDS numbered as 1, 4, 6, 7, 9, and 10, respectively, the mapping of GN and RGN is listed as below.

GN	1	4	6	7	9	10
RGN	-5	-4	-3	-2	-1	0

In the following job, use `RGN=-1` to reference GDS whose GN equals 9 and use `RGN=-4` to reference GDS whose GN equals 4.

---

**Listing 3-22 Example of Referencing Existing Generation Files**

---

```
(STEP1)
m_FileAssign -d SHR,KEEP,KEEP -g -1 SYSUT1 "$DATA/GDG1"
m_FileAssign -d SHR,KEEP,KEEP -g -4 SYSUT2 "$DATA/GDG1"
```

---

If "DELETE" is specified in the `DISPOSITION` field of `m_FileAssign`, the corresponding GDS will be deleted after the current step finishes, resulting in a change of mapping between GN and RGN. The changed mapping will be visible in the next step.

For example, if `GDG1` contains six GDS numbered as 1, 4, 6, 7, 9, and 10, respectively, the mapping of GN and RGN is listed as below.

---

GN	1	4	6	7	9	10
RGN	-5	-4	-3	-2	-1	0

---

In the following job, use `RGN=-1` to reference GDS whose GN equals 9 and use `RGN=-4` to reference GDS whose GN equals 4.

You can run a job as below.

---

**Listing 3-23 Example of Referencing Existing Generation Files with DELETE Specified**

---

```
(STEP1)
m_FileAssign -d OLD,DELETE,DELETE -g -1 SYSUT1 "$DATA/GDG1"
m_FileAssign -d OLD,DELETE,DELETE -g -4 SYSUT2 "$DATA/GDG1"

(STEP2)
m_FileAssign -d OLD,DELETE,DELETE -g -1 SYSUT1 "$DATA/GDG1"
m_FileAssign -d OLD,DELETE,DELETE -g -2 SYSUT2 "$DATA/GDG1"
```



---

In the above example, after STEP1 finishes, the mapping of GN and RGN becomes the one as below.

GN	1	6	7	10
RGN	-3	-2	-1	0

In STEP2, the GDS pointed by SYSUT1 (the GDS whose GN is 7) and the GDS pointed by SYSUT2 (the GDS whose GN is 6) are deleted.

After STEP2 finishes, the mapping of GN and RGN becomes the one as below.

GN	1	10
RGN	-1	0

### Deleting Generation Files in a GDG

ART for Batch supports you to delete generation files, newly added or current existing, through the disposition of DD specified for `m_FileAssign`.

- Deleting Newly Added GDS (See [Listing 3-24](#) for an example)
- Deleting Existing GDS (See [Listing 3-25](#) for an example)

#### Listing 3-24 Deleting Newly Added GDS

---

```
(STEP1)
m_FileAssign -d NEW,DELETE,DELETE -g +1 SYSUT1 "$DATA/GDG1"
m_FileAssign -d NEW,DELETE,DELETE -g +5 SYSUT2 "$DATA/GDG1"
(STEP2)
m_FileAssign -d NEW,DELETE,DELETE -g +1 SYSUT1 "$DATA/GDG1"
```

## Using Batch Runtime

```
m_FileAssign -d NEW,DELETE,DELETE -g +5 SYSUT2 "$DATA/GDG1"
```

---

In the above example, eventually, no GDS is added to GDG1.

### Listing 3-25 Deleting Existing GDS

---

```
(STEP1)
m_FileAssign -d NEW,DELETE,DELETE -g -1 SYSUT1 "$DATA/GDG1"
m_FileAssign -d NEW,DELETE,DELETE -g -3 SYSUT2 "$DATA/GDG1"
(STEP2)
m_FileAssign -d NEW,DELETE,DELETE -g -1 SYSUT3 "$DATA/GDG1"
m_FileAssign -d NEW,DELETE,DELETE -g -3 SYSUT4 "$DATA/GDG1"
```

---

In the above example, GDG1 has six GDS numbered as 1, 4, 6, 7, 9, and 10, respectively. The GDS pointed by SYSUT1 (the GDS whose GN is 9), by SYSUT2 (the GDS whose GN is 6), by SYSUT3 (the GDS file whose GN is 7), and by SYSUT4 (the GDS file whose GN is 1) are deleted.

**Note:** Removing a GDG's all GDS does not remove the GDG itself, but just result in the fact that the GDG contains 0 GDS.

### Deleting a GDG

You can delete a GDG as a whole by calling `m_FileDelete` with the GDG base name, as shown in [Listing 3-26](#). In this way, all the GDG's GDS will be deleted accordingly. The operation of deleting GDG is committed immediately and cannot be rolled back.

### Listing 3-26 Deleting a GDG

---

```
m_FileDelete ${DATA}/PJ01DDD.BT.GDG
```

---

## Cataloging a GDG

Only GDG base can be cataloged; its GDS cannot be cataloged individually.

It is required to enable "file catalog" function in ART for Batch catalog a GDG. Additionally, in catalog mode, the parameter [-v volume] specified in `m_FileAssign` is ignored.

**Note:** A GDG will be cataloged once it is defined.

## Committing a GDG

All GDG having changes in the current step will be committed no matter if the current step successfully finishes.

Committing a GDG updates the information in GDG management system, such as Oracle DataBase or file (\*.gens), and commits the temporary generation files; however, committing a GDG does not change the mapping relationship between GN and RGN, meaning, in one step of a job, a RGN always references to the same GDS.

For example, GDG1 has six GDS numbered as 1, 4, 6, 7, 9, and 10, respectively.

### Listing 3-27 Example of Committing a GDG

---

```
(STEP1)
m_FileAssign -d NEW,KEEP,KEEP -g +1 SYSUT1 "$DATA/GDG1"
m_FileAssign -d NEW,KEEP,KEEP -g +2 SYSUT2 "$DATA/GDG1"
m_FileAssign -d NEW,KEEP,KEEP -g -1 SYSUT3 "$DATA/GDG1"
(STEP2)
m_FileAssign -d NEW,KEEP,KEEP -g -1 SYSUT4 "$DATA/GDG1"
```

---

In STEP1, the mapping of GN and RGN (both in job and in GDG management system) becomes the one as below. SYSUT3 references to the GDS whose GN is 9.

---

GN	1	4	6	7	9	10	11	12
RGN	-5	-4	-3	-2	-1	0	1	2

---

In STEP2, the mapping of GN and RGN in GDG management system becomes the one as below.

GN	1	4	6	7	9	10	11	12
RGN	-7	-6	-5	-4	-3	-2	-1	0

However, the mapping of GN and RGN in the current running job is not changed; in the below example, SYSUT4 stills references to the GDS whose GN is 9 rather than the GDS whose GN is 11.

GN	1	4	6	7	9	10	11	12
RGN	-5	-4	-3	-2	-1	0	1	2

## File-Based Management

### Configuration

MT\_GENERATION variable specifies the way of managing GDG files. To manage GDG in \*.gens files, you need to set the value to GENERATION\_FILE.

### Concurrency Control and Authorization

In file-based GDG management mechanism, one GDG file can only be accessed by one job at any time, that is, a single GDG cannot be accessed by multiple jobs simultaneously. To access a GDG file, the file lock must be acquired by the existing internal function `mi_FileConcurrentAccessReservation`. File-based GDG management mechanism uses a file \*.gens (\* represents the GDG base name) to control concurrency and authorization. User access checking depends on whether the \*.gens file can be accessed or not.

## DB-Based Management

For DB-based management, Oracle Database and DB2 database are supported.

**Note:** To enable this function, MT\_GENERATION must be set to GENERATION\_FILE\_DB, MT\_DB must be set to DB\_ORACLE or DB\_DB2LUW, and MT\_GDG\_DB\_ACCESS must be set to valid database connection string for accessing Oracle Database or DB2 database.

## Database Tables

[Table 3-5](#) shows the general management for each GDG managed by Batch Runtime. In this table, each row represents a GDG. All GDG files share a single `GDG_DETAIL` table.

**Table 3-5 GDG\_DEFINE**

Name	Type	Description
<code>GDG_BASE_NAME</code>	<code>VARCHAR(1024)</code>	Full path name of GDG.  It cannot contain only a relative path relative to a single repository. The length of <code>GDG_BASE_NAME</code> is limited to 1024, i.e. the minimum of <code>PATH_MAX</code> on different UNIX platforms.
<code>GDG_MAX_GEN</code>	<code>INT</code>	Maximum number of generation files.  It contains the upper limit of generations specified by <code>-s</code> option. <code>-s</code> option can be set in the range of 1-9999.
<code>GDG_CUR_GEN</code>	<code>INT</code>	GDG current generation number
Primary Key: <code>GDG_BASE_NAME</code>		

[Table 3-6](#) shows the detailed information of all the GDG generation files. In this table, each row represents a generation file of a GDG.

**Table 3-6 GDG\_DETAIL**

Name	Type	Description
<code>GDG_BASE_NAME</code>	<code>VARCHAR(1024)</code>	Full path of the GDG principal name.
<code>GDG_REL_NUM</code>	<code>INT</code>	Relative generation number of a generation file.
<code>GDG_ABS_NUM</code>	<code>INT</code>	Absolute generation number of a generation file.
<code>GDG_JOB_ID</code>	<code>VARCHAR(8)</code>	The ID of the job that creates the file.
<code>GDG_JOB_NAME</code>	<code>VARCHAR(32)</code>	The name of the job that creates the file.

**Table 3-6 GDG\_DETAIL**

Name	Type	Description
GDG_STEP_NAME	VARCHAR( 32 )	The name of the step that creates the file.
GDG_CRE_TIME	TIMESTAMP	The timestamp when the file is created.
Primary Key: GDG_BASE_NAME+ GDG_ABS_NUM		

GDG\_FILE\_NAME (the physical generation file name) is not stored in table GDG\_DETAIL since it can be constructed from GDG\_BASE\_NAME in GDG\_DEFINE and GDG\_ABS\_NUM in GDG\_DETAIL.

**Note:** To back up GDG information, you need to back up two database tables: GDG\_DEFINE and GDG\_DETAIL.

## Generation File Naming Rule

Table 3-7 shows the rule of generation file name:

**Table 3-7 Generation File Naming Rule**

Condition	File Name	Description
GDG_REL_NUM > 0	\${GDG_BASE_NAME}.Gen.\${GDG_ABS_NUM}.tmp	Uncommitted
GDG_REL_NUM <= 0	\${GDG_BASE_NAME}.Gen.\${GDG_ABS_NUM}	Committed

## Configuration Variables

### MT\_GENERATION

This variable specifies the way of managing GDG files. To manage GDG files in database, you need to set the value to GENERATION\_FILE\_DB and configure MT\_GDG\_DB\_ACCESS appropriately.

### MT\_GDG\_DB\_ACCESS

This variable is used along with MT\_GENERATION when it is set to GENERATION\_FILE\_DB, and must be set with the valid database login account. For accessing Oracle DB, it should be specified in the format of `userid/password@sid`, for example, `scott/tiger@orcl`.

## External Shell Scripts

You can use the two external shell scripts to create and drop the new database table automatically.

## **CreateTableGDG.sh**

### Description

Creates table GDG\_DEFINE and GDG\_DETAIL in database

### Usage

```
CreateTableGDG.sh <DB_LOGIN_PARAMETER>
```

### Sample

```
CreateTableGDG.sh scott/tiger@orcl
```

## **DropTableGDG.sh**

### Description

Drops table GDG\_DEFINE and GDG\_DETAIL from database.

### Usage

```
DropTableGDG.sh <DB_LOGIN_ PARAMETER>
```

### Sample

```
DropTableGDG.sh scott/tiger@orcl
```

## **Concurrency Control and Authorization**

DB-based GDG management mechanism maintains the same concurrency control behavior as File-based GDG management mechanism, but has a different \*.ACS (\* represents the GDG base name) file format. In DB-based GDG management mechanism, you don't need to lock the tables mentioned in [Database Tables](#) as any job that accesses the rows corresponding to a GDG must firstly acquire the file lock of the GDG. That is to say, there is no need to perform concurrency control in the database access level. You cannot access database if you don't have access permission (read or write) to the corresponding \*.ACS file. If you need to modify a GDG file, you must have write permissions to the generation files and the directory holding the generation files, and MT\_GDG\_DB\_ACCESS must be configured correctly to have appropriate permissions to the tables mentioned in [Database Tables](#).

You can only copy DB-based GDG management description entirely and replace the file name.

## **Exception Handling**

There are four kinds of information in DB-based GDG management mechanism:

- GDG\_DEFINE
- \*.ACS file
- GDG\_DETAIL
- Physical file on disk

These information should be kept consistently for a GDG file. Batch Runtime checks the consistency from GDG\_DEFINE to Physical files when a GDG file is accessed the first time in a job. If exceptions happen and result in inconsistency among these information, Batch Runtime terminates the current job and reports error.

This behavior is different from the existing file-based mechanism, which does not check the consistency but only reports exceptions encountered in the process.

## Support for Data Control Block (DCB)

Both file-based GDG and DB-based GDG support Data Control Block (DCB).

### Defining .dcb File

.dcb file can have two values: "-t <file type>" and "-r <record length>".

#### -t <file type>

-t <file type> must be LSEQ or SEQ in m\_FileAssign to create the first generation file. If you don't specify any file type in job ksh file, LSEQ will be used by default.

#### -r <record length>

For SEQ file, the value is mandatory and must be a number or "number1-number2".

For LSEQ file, the value is optional. Once set, this value must be a number.

### Creating .dcb file

Create .dcb file for GDG data set when the first generation file is created by m\_FileAssign -g +1.

**Notes:** If a GDG is created by m\_GenDefine rather than m\_FileAssign, .dcb file will not exist until the first generation file is created by m\_FileAssign -g +1.

Once .dcb file is created, its contents will not be changed by any other m\_FileAssign statement afterwards, unless such m\_FileAssign creates the first generation file again.



## Deleting .dcb file

If a GDG is deleted by `m_FileDelete`, the corresponding `.dcb` file will be deleted automatically. However, if all generation files in one GDG are deleted while the GDG itself exists, the corresponding `.dcb` file will not be deleted.

## Using an In-Stream File

To define and use a file whose data is written directly inside the Korn shell script, use the `m_FileAssign` function with the `-i` parameter. By default the string `_end` is the “end” delimiter of the in-stream flow as shown in [Listing 3-28](#).

### Listing 3-28 In-stream Data Example

---

```
(STEP1)
    m_FileAssign -i INFIL
data record 1
data record 2
...
_end
```

---

## Using a Set of Concatenated Files

To use a set of files as a concatenated input (which in z/OS JCL was coded as a DD card, where only the first one contains a label), use the `m_FileAssign` function with the `-C` parameter as shown in [Listing 3-29](#).

### Listing 3-29 Using a Concatenated Set of Files Example

---

```
(STEPDD02)
    m_FileAssign -d SHR INF ${DATA}/PJ01DDD.BT.QSAM.KBDDI002
    m_FileAssign -d SHR -C ${DATA}/PJ01DDD.BT.QSAM.KBDDI001
```

```
m_ProgramExec BDDAB001
```

---

## Using an External “sysin”

To use an “external sysin” file which contains commands to be executed, use the `m_UtilityExec` function.

```
m_FileAssign -d OLD SYSIN ${SYSIN}/SYSIN/MUEX07  
m_UtilityExec
```

## Deleting a File

Files (including generation files) can be deleted using the `m_FileDelete` function:

```
m_FileDelete ${DATA}/PJ01DDD.BT.QSAM.KBST0045
```

## RDB Files

In a migration project from z/OS to UNIX/Linux, some permanent data files may be converted to relational tables. See the File-to-Oracle chapter of the Oracle Tuxedo Application Runtime Workbench.

When a file is converted to a relational table, this change has an impact on the components that use it. Specifically, when such a file is used in a z/OS JCL, the converted Korn shell script corresponding to that JCL should be able to handle operations that involve this file.

In order to keep the translated Korn shell script as standard as possible, this change is not handled in the translation process. Instead, all the management of this type of file is performed at execution time within Batch Runtime.

In other words, if in the z/OS JCL there was a file copy operation involving the converted file, this is translated to a standard copy operation for files in Batch Runtime, in other words an `m_FileLoad` operation).

The management of a file converted to a table is made possible through an RDB file. An RDB file is a file that has the same name as the file that is converted to a table but with an additional suffix: `.rdb`.

Each time a file-related function is executed by Batch Runtime, it checks whether the files were converted to table (through testing the presence of a corresponding `.rdb` file). If one of the files

concerned have been converted to a table, then the function operates the required intermediate operations (such as: unloading and reloading the table to a file) before performing the final action.

All of this management is transparent to the end-user.

## Using an RDBMS Connection

When executing an application program that needs to connect to the RDBMS, the `-b` option must be used when calling the `m_ProgramExec` function.

Connection and disconnection (as well as the commit and rollback operations) are handled implicitly by Batch Runtime and can be defined using the following two methods:

- Set the environment variable `MT_DB_LOGIN` before booting the TuxJES system.

**Note:** In this case, all executing jobs use this variable.

- Set its value in the TuxJES Security Configuration file for different users.

The `MT_DB_LOGIN` value must use the following form: `dbuser/dbpasswd[@ssid]` or `"/`.

**Note:** `"/` should be used when the RDBMS is configured to allow the use of UNIX authentication and not RDBMS authentication, for the database connexion user.

Please check with the database administrator whether `"/` should be used or not.

The `-b` option must also be used if the main program executed does not directly use the RDBMS but one of its subsequent sub-programs does as shown in [Listing 3-30](#).

### Listing 3-30 RDBMS Connection Example

---

```
(STEPDD02)
  m_FileAssign -d MOD OUTF ${DATA}/PJ01DDD.BT.QSAM.REPO01
  m_ProgramExec -b DBREP001
```

---

The `m_ProgramExec` function may submit three types of executable files (Cobol executable, command language script, or C executable). It launches the `runb` program. We have provided the `runb` for `$ARTDIR/Batch_RT/ejr_mf_ora` (on Linux) and `ejr_ora` (other platforms). If you use neither Microfocus COBOL compiler nor Oracle Database, go to `$ARTDIR/Batch_RT/ejr` and run `"make.sh"` to generate your required `runb`.

## Using Batch Runtime

The `runb` program, runtime compiled with database libraries, runs the `runbatch` program.

The `runbatch` program, is in charge to :

- do the connection to the database (if necessary)
- run the user program
- do the commit or rollback (if necessary)
- do the disconnection from the database (if necessary)

## Submitting a Job Using INTRDR Facility

The INTRDR facility allows you to submit the contents of a sysout to TuxJES (see the TuxJES documentation). If TuxJES is not present, a command “nohup EJR” is used.

Example:

```
m_FileAssign -d SHR SYSUT1 ${DATA}/MTWART.JCL.INFO
m_OutputAssign -w INTRDR SYSUT2
m_FileRepro -i SYSUT1 -o SYSUT2
```

The contents of the file `${DATA}/MTWART.JCL.INFO` (ddname `SYSUT1`) is copied into the file which ddname is `SYSUT2` and using the option “-w INTRDR” is submitted.

Note that the output file must contain valid ksh syntax.

**Note:** If the batch job script generated at runtime is in JCL language, it can't be submitted by INTRDR.

## Submitting a Job With EJR

When using Batch Runtime, TuxJES can be used to launch jobs (see the TuxJES documentation), but a job can also be executed directly using the EJR spawner.

Before performing this type of execution, ensure that the entire context is correctly set. This includes environment variables and directories required by Batch Runtime.

Example of launching a job with EJR:

```
# EJR DEFVCUST.ksh
```

For a complete description of the EJR spawner, please refer to the Oracle Tuxedo Application Runtime for Batch Reference Guide.

## User-Defined Entry/Exit

Batch Runtime allows you to add custom pre- or post- actions for public APIs. For each `m_*` (\* represents any function name) function, you can provide `m_*_Begin` and `m_*_End` function and put them in `ejr/USER_EXIT` directory. They are invoked automatically when a job execution entering or leaving an `m_*` API.

Whether an `m_*` API calls its user-defined entry/exit function depends on the existence of `m_*_Begin` and `m_*_End` under `ejr/USER_EXIT`.

A pair of general user entry/exit APIs, `mi_UserEntry` and `mi_UserExit`, are called at the entry and exit point of each external API. The argument to these APIs consists of the function name in which they are called, and the original argument list of that function. You don't need to modify these two APIs, but just need to provide your custom entry/exit for `m_*` external APIs.

`mi_UserEntry` and `mi_UserExit` are placed under `ejr/COMMON`.

**Note:** In user entry/exit function, users are not allowed to use any function provided by ART for Batch; however, in user's script, a return statement returns value to the caller and ART for Batch checks if calling user entry/exit function works successfully through the return code. Return code 0 continues the job; non-zero value terminates the job.

You are suggested not to call `exit` in user entry/exit function. Because In the framework, `exit` is aliased an internal function, `mif_ExitTrap`, which is invoked ultimately if `exit` in user entry/exit function is called. If `exit 0` is called, the framework does nothing and job is continue, if `exit not_0` is called, a global variable is set and may terminate the current job.

## Configuration

You should include only one function, e.g. `m_*_Begin` or `m_*_End`, in a single file with the same name as the function, and then put all such files under `ejr/USER_EXIT`.

You are not allowed to provide custom entry/exit functions for any `mi_*` prefix function provided by Batch Runtime.

## Batch Runtime Logging

This section contains the following topics:

- [General Introduction](#)
- [Log Header](#)
- [File Information Logging](#)

## General Introduction

### Log Message Format

Each log message defined in `CONF/Messages.conf` is composed of six fields, as listed in [Table 3-8](#):

**Table 3-8 Log Message Format**

Field	Content
1	Message identifier
2	Functions that can display the message (generic name using *)
3	Level of display. Default value: 4
4	Destination of display (u,e,o). <ul style="list-style-type: none"> <li>• U: User output</li> <li>• E: Error Output (stderr)</li> <li>• O: Standard output (stdout)</li> </ul>
5	Header flag (0,1,b). Default value: 0 <ul style="list-style-type: none"> <li>• 0: No header will be displayed</li> <li>• 1: A hard-coded header format will be displayed</li> <li>• b: Specific for exceptions messages Fatal/Error/Warning</li> </ul>
6	The message to be displayed with possible dynamic values

The levels of these messages are set to 4 by default.

You can specify the message level of Batch Runtime to control whether to print these three messages in job log.

## Log Message Level

[Table 3-9](#) lists the Log message levels provided by Batch Runtime:

**Table 3-9 Log Message Level**

Level	Message
1	FATAL only
2	Previous level and errors
3	Previous level and information
4	Previous level and file information log

**Table 3-9 Log Message Level**

Level	Message
5	Previous level and high level functions
6	Previous level and technical functions
7	Same as level 3 and high level functions which correspond to the <code>-d regexp</code> option
8	Same as 7 and technical level functions which correspond to the <code>-d regexp</code> option
9	Reserved

## Log Level Control

The default level of displaying messages in job log is 3. You can also choose one of the following ways to change the level:

- Use `-v` option of EJR
- Use the environment variable `MT_DISPLAY_LEVEL`

The display level set by EJR can override the level set by `MT_DISPLAY_LEVEL`.

## Log File Structure

For each launched job, Batch Runtime produces a log file containing information for each step that was executed. This log file has the following structure as shown in [Listing 3-31](#).

**Listing 3-31 Log File Example**


---

```
JOB Jobname BEGIN AT 20091212/22/09 120445
BEGIN PHASE Phase1
Log produced for Phase1
.....
.....
.....
```



```

END PHASE Phase1 (RC=Xnnnn, JOBRC=Xnnnn)

BEGIN PHASE Phase2

Log produced for Phase2

.....

.....

.....

END PHASE Phase2 (RC=Xnnnn, JOBRC=Xnnnn)

.....

.....

BEGIN PHASE END_JOB

.....

END PHASE END_JOB (RC=Xnnnn, JOBRC=Xnnnn)

JOB ENDED WITH CODE (C0000})

Or

JOB ENDED ABNORMALLY WITH CODE (S990})

```

---

When not using TuxJes, the log file is created under the  `${MT_LOG}`  directory with the following name:  `<Job name>_<TimeStamp>_<Job id>.log`

For more information, see [Using Tuxedo Job Enqueueing Service \(TuxJES\)](#).

## Log Header

Batch Runtime logging functionality provides an informative log header in front of each log line, in the following format:

```
YYYYmmdd:HH:MM:SS:TuxSiteID:JobID:JobName:JobStepName
```

You can configure the format of log header, but should not impact any configuration and behavior of existing specific message header: type 0, 1 and b.

[Table 3-10](#) shows the variables you can use for specifying the general log header:

**Table 3-10 variables for Specifying General Log Header**

Variable	Description
MTI_SITE_ID	If the job is submitted from TuxJES, it is the logical machine ID configured for the machine by TuxJES, otherwise it's empty.
MTI_JOB_ID	If the job is submitted from TuxJES, it is the job ID assigned by JES.
MTI_JOB_NAME	Name of the job assigned by <code>m_JobBegin</code> in the job script.
MTI_STEP_NAME	Name of the current executing job step.
MTI_SCRIPT_NAME	Name of the job script.
MTI_PROC_NAME	Name of the proc when the code included from a PROC by <code>m_ProcInclude</code> is executing; empty otherwise.

## Configuration

`MT_LOG_HEADER` is a new configuration variable added in `CONF/BatchRT.conf`, for example:

```
MT_LOG_HEADER='$(date +%Y%m%d:%H%M%S') : ${MTI_SITE_ID} : ${MTI_JOB_NAME} : ${MTI_JOB_ID} : ${MTI_JOB_STEP} : '
```

If the value of `MT_LOG_HEADER` is not a null string, its contents are evaluated as a shell statement to get its real value to be printed as the log header, otherwise this feature is disabled.

**Note:** The string that configured to `MT_LOG_HEADER` is treated as a shell statement in the source code, and is interpreted by "eval" command to generate the corresponding string used as log header:

Syntax inside: `eval mt_MessageHeader="\ ${MT_LOG_HEADER} \"`

To configure this variable, you need to comply with the following rules:

- `MT_LOG_HEADER` must be a valid shell statement for "eval", and must be quoted by single quotation marks.
- All the variables used in `MT_LOG_HEADER` must be quoted by "\${}". For example: `${MTI_JOB_STEP}`
- All the command line used in `MT_LOG_HEADER` must be quoted by "\$()". For example: `$(date +%Y%m%d:%H%M%S')`

You can modify the above examples according to your format needs using only the variables listed in [Table 3-10](#).

This configuration variable is commented by default, you need to uncomment it to enable this feature.

## File Information Logging

Logging system can logs the detailed file information in job log, as well as the information when a file is assigned to a DD and when it is released.

File assignment information is logged in the following functions:

`m_FileAssign`

File release information is logged in the following functions:

`m_PhaseEnd`

File information is logged in the following functions:

- `m_FileBuild`
- `m_FileClrData`
- `m_FileConcatenate`
- `m_FileCopy`
- `m_FileDelete`
- `m_FileEmpty`
- `m_FileExist`
- `m_FileLoad`
- `m_FileRename`
- `m_FilePrint`
- `m_FileRepro`

## Configuration

### Messages.conf

The following message identifiers are defined in `CONF/Messages.conf` to support using of `mi_DisplayFormat` to write file assignment and file information log.

- `FileAssign;m_FileAssign;4;ueo;0;%s`

- `FileRelease;m_PhaseEnd;4;ueo;0;%s`
- `FileInfo;m_File*;4;ueo;0;%s`

**Notes:**

`CONF/Messages.conf` is not configurable. Do not edit this file.

The string "%s" at the end of each identifier represents it will be written to log file. You can configure its value using the following variables defined in `CONF/Batch.conf`. For more information, see [Table 3-12](#).

- `MT_LOG_FILE_ASSIGN` (for `FileAssign`)
- `MT_LOG_FILE_RELEASE` (for `FileRelease`)
- `MT_LOG_FILE_INFO` (for `FileInfo`)

**BatchRT.conf**

Three configuration variables should be defined in `CONF/BatchRT.conf` to determine the detailed file information format. With the placeholders listed in [Table 3-11](#), you can configure file log information more flexibly.

**Table 3-11 Placeholders**

Placeholder	Description	Value and Sample
<code>&lt;%DDNAME%&gt;</code>	DD Name for the file being operated	<code>SYSOUT1</code>
<code>&lt;%FULLPATH%&gt;</code>	Full path for the file being operated	<code>/local/simpjob/work/TEST001.Gen.000000001</code>
<code>&lt;%FILEDISP%&gt;</code>	DISP for the file being operated	<code>SHR or NEW</code>

**Table 3-12 Configuration Variables in CONF/BatchRT.conf**

Name	Value and Sample	Available Placeholder
MT_LOG_FILE_ASSIGN	FileAssign: DDNAME=( <%DDNAME%> ); FILEINFO=( \$(ls -l --time-style='+ %Y/%m/%d %H:%M:%S' --no-group <%FULLPATH%> )'; FILEDISP=( <%FILEDISP %> )	<%DDNAME%> <%FULLPATH%> <%FILEDISP%>
MT_LOG_FILE_RELEASE	FileRelease: DDNAME=( <%DDNAME%> ); FILEINFO=( \$(ls -l --time-style='+ %Y/%m/%d %H:%M:%S' --no-group <%FULLPATH%> )'; FILEDISP=( <%FILEDISP %> )	<%DDNAME%> <%FULLPATH%> <%FILEDISP%>
MT_LOG_FILE_RELEASE	FILEINFO=( \$(ls -l --time-style='+ %Y/%m/%d %H:%M:%S' --no-group <%FULLPATH%> ) )	<%FULLPATH%>

**Note:** "operation" is hard-coded into source code, such as FileCopy source, FileCopy Destination, and FileDelete etc.

To configure strings to these MT\_LOG\_FILE\_\* variables, replace the placeholders with corresponding values (just string replacement). The result is treated as a shell statement, and is interpreted by "eval" command to generate the corresponding string writing to log:

Syntax inside: eval mt\_FileInfo="\\${MT\_LOG\_FILE\_INFO}\\"

To configure these variables, you need to comply with the following rules:

- After placeholders are replaced, MT\_LOG\_FILE\_\* must be a valid shell statement for "eval", and must be quoted by single quotation marks.
- Only the placeholders listed in Table 3-11 can be used in MT\_LOG\_FILE\_\*.
- All the command line used in MT\_LOG\_HEADER must be quoted by "\$()". For example:  
\$(ls -l --time-style='+ %Y/%m/%d %H:%M:%S' --no-group <%FULLPATH%> )

If the level of FileInfo message is equal to or less than the message level specified for Batch Runtime and MT\_LOG\_FILE\_\* is set to a null string, FileInfo message will not be displayed in job log. If MT\_LOG\_FILE\_\* is set to an incorrect command to make file information invisible,

FileInfo message will not be displayed in job log as well, but the job execution will not be impacted.

**Note:** You can customize these variables according to your format needs, but make sure the command is valid, otherwise the file information will not be logged.

## Using Batch Runtime With a Job Scheduler

Entry points are provided in some functions (`m_JobBegin`, `m_JobEnd`, `m_PhaseBegin`, `m_PhaseEnd`) in order to insert specific actions to be made in relation with the selected Job Scheduler.

## Executing an SQL Request

A SQL request may be executed using the function `m_ExecSQL`.

Depending on the target database, the function executes a “sqlplus” command with ORACLE database, or a “db2 -tsx” command with UDB.

Note that the environment variable `MT_DB_LOGIN` must be set (database connection user login).

The `SYSIN` file must contain the SQL requests and the user has to verify the contents regarding the database target.

## Simple Application on COBOL-IT / BDB

Batch COBOL programs compiled by COBOL-IT can access the indexed ISAM files which are converted from Mainframe VSAM files through the ART Workbench. VSAM files can be stored in BDB through COBOL-IT.

To enable this function in Batch runtime, do the followings during runtime:

- Compile COBOL programs by COBOL-IT compiler with specifying `bdb:yes`.
- Set `DB_HOME` correctly because it's required by BDB.
- Set the following environment variables before ART for Batch launches a job.
  - `export COB_EXTFH_INDEXED=BDBEXTFH`
  - `export COB_EXTFH_LIB=/path_to_Cobol-IT/lib/libbdbextfh.so #For example, export COB_EXTFH_LIB=/opt/cobol-it-64/lib/libbdbextfh.so`

# Dynamic JCL Job Execution

This section contains the following topics:

- [General Introduction](#)
- [Requirements](#)
- [Configurations](#)
- [Using JES Client to Manage JCL Jobs](#)

## General Introduction

Oracle Tuxedo ART Batch Runtime supports users to manage native JCL jobs with real-time workbench conversion without any pre-conversion. For more information, please refer to [JCL Conversion](#).

## Requirements

It's required to install Oracle Tuxedo ART Workbench and make it executable.

Two additional requirements should be fulfilled as below if Oracle Tuxedo ART Workbench is deployed on the remote machine (`host1`) while `ARTJESCONV` server is deployed on another machine (`host2`).

- NFS must be configured to cover all the folders and files shared by both ART Workbench and JES, and `$JESROOT` must be configured on NFS.
- A trusted SSH connection must be configured between `host1` and `host2`, that is, the user (`user2`) who boots up `ARTJESCONV` is allowed to log into `host1` without passwords. By doing this, `ARTJESCONV` can invoke ART Workbench installed on `host1` directly without interaction.

**Note:** If multiple `ARTJESCONV` servers on more than one machine are configured in JES domain, the trusted SSH connection should be configured on each machine equipped with `ARTJESCONV`.

If Workbench is deployed on local machine, it is optional to set the `host`.

For example, you must do the followings to add `user2@host2` to `user1@host1` if Oracle Tuxedo ART Workbench and `ARTJESCONV` are deployed in different machines.

1. Login `host2` with user name `user2`.

2. Run "cd \$HOME/.ssh" on host2.
3. Run "ssh-keygen -t rsa" to generate id\_rsa and id\_rsa.pub.
4. Login host1 with user name user1.
5. Run "cd \$HOME/.ssh" on host1.
6. Add the content of host2:\$HOME/.ssh/id\_rsa.pub file to authorized\_keys.

## Configurations

### Working Folder Configurations for JCL Conversion

The template working folder for JCL conversion is \$JESROOT/jcl\_conv\_dir, which will be created automatically if it does not exist at startup. \$JESDIR/Batch\_RT/jcl\_conv\_dir contents are automatically copied to such working folder when Batch Runtime starts. When a JCL job is submitted, JES copies this template folder to folder \$JESROOT/<JOBID>/JCL, and puts the JCL job file to folder \$JESROOT/<JOBID>/JCL/source/JCL/, where Workbench works.

Users need to copy all the INCL, PROC, and SYSIN to the template working folder for each JCL job. When converting and executing a JCL job,

\$JESROOT/<JOBID>/JCL/target/PROC:\$JESROOT/<JOBID>/JCL/target /INCL is added to the head of the environment variable PROCLIB, and

\$JESROOT/<JOBID>/JCL/target/Master-SYSIN is set to the environment variable SYSIN.

The Workbench configuration file in the working folder for JCL conversion is param/config-trad-JCL.desc. Users should customize it; otherwise, default values will be used. For more information, please refer to [The JCL-Translation Configuration File](#).

### EJR Configurations

MT\_REFINEDIR and MT\_REFINEDISTRIB are required to be configured. For more information, please refer to [Table 3-3](#).

### The Queue for JCL Conversion

The queue, CONV\_JCL, is added to the queue space JES2QSPACE to support JCL conversion. For more information, please refer to [Table 8 TuxJES Queues](#).



## Using JES Client to Manage JCL Jobs

### Submitting a JCL Job

Option `-l` is used to submit a JCL job with the following usage.

- `artjesadmin -l JCLScriptName` (in the shell command line)
- `submitjob -l JCLScriptName` (in the `artjesadmin` console)

### Printing Jobs

`[-t JCL|KSH]` is used as a filter with the following usage.

- Print all jobs: `printjob`
- Print JCL jobs: `printjob -t JCL`
- Print KSH jobs: `printjob -t KSH`

The column, `job type`, is added to the results with one of the following values.

- `JCL` for JCL jobs
- `KSH` for KSH jobs

Before the conversion phase completes, the JCL job name and class are null, and the priority is displayed as 0.

### Holding/Releasing/Canceling/Purging a JCL job

The usage is the same as KSH jobs.

### JCL Conversion Log

The JCL conversion log is `$JESROOT/<JOBID>/LOG/<JOBID>.jcllog`.

## Network Job Entry (NJE) Support

This section contains the following topics.

- [General Introduction](#)
- [Configurations](#)
- [NJE Job Sample](#)

## General Introduction

With NJE support, users can implement the following functionalities in Batch Runtime exactly as they do in JCL jobs.

- /\* ROUTE XEQ
- /\* XEQ
- /\* XMIT

By `m_SetJobExecLocation` API of Batch Runtime, users can develop KSH jobs with NJE support. For example,

- Specify the server group, on which the job will be executed.
- In a job, transmit an in-stream job to another server group and make it run on that server group.

## Configurations

### Job Execution Server Group

When specifying the server group name, which is specified as job execution group in API `m_JobSetExecLocation`, please ensure the followings.

- The specified server group must exist in `ubbconfig` file of JES domain.
- At least one `ARTJESINITIATOR` server must be deployed in that server group.

### ON/OFF Setting of NJE Support

There is a corresponding setting item in JES configuration file.

**Table 3-13** Configurations in `<APPDIR>/jesconfig`

Name	Value	Default Value
NJESUPPORT	ON: Enable NJE support OFF: Disable NJE support	OFF

If NJE support is disabled in `jesconfig`, the statement `m_SetJobExecLocation <SvrGrpName>` is ignored by TuxJES and then the job may be executed by any `ARTJESINITIATOR` in any server group.

## Environment Variable MT\_TMP in MP Mode

In MP mode, MT\_TMP needs to be configured on NFS, and all the nodes in tuxedo domain should have the same value of MT\_TMP and share it.

MT\_TMP can be configured in file \$MT\_ROOT/CONF/BatchRT.conf, or to export it as environment value before tlisten is started in each node.

## Queue EXECGRP

If NJESUPPORT is enabled in jesconfig, a new queue named EXECGRP must be created in the existing queue space JES2QSPACE. If EXECGRP is not created, no jobs can be processed by JES.

## NJE Job Sample

### Listing 3-32 Sample of Specifying Job Execution Server Group (XEQ)

---

```
m_JobBegin -j SAMPLEJCL -s START -v 2.0 -c R
m_JobSetExecLocation "ATLANTA"

while true ;
do

    m_PhaseBegin

    case ${CURRENT_LABEL} in

    (START)

    # XEQ ATLANTA

        JUMP_LABEL=STEP01

        ;;

    (STEP01)

        m_OutputAssign -c "*" SYSPRINT

        m_FileAssign -i SYSIN

        m_FileDelete ${DATA}/GBOM.J.PRD.ABOMJAW1.ABEND02

        m_RcSet 0

    _end
```

## Using Batch Runtime

```
m_UtilityExec
JUMP_LABEL=END_JOB
;;
(END_JOB)
break
;;
(*)
m_RcSet ${MT_RC_ABORT:-S999} "Unknown label : ${CURRENT_LABEL}"
break
;;
esac
m_PhaseEnd
done
m_JobEnd
```

---

In the above sample, the job can be submitted on any JES node, but only be executed by the ARTJESINITIATOR which belongs to JES's tuxedo server group ATLANTA.

### Listing 3-33 Sample of Transmitting and Submitting a Job to Another Server Group (XMIT)

---

```
m_JobBegin -j JOBA -s START -v 2.0
while true;
do
    m_PhaseBegin
    case ${CURRENT_LABEL} in
(START)
        m_FileAssign -i -D \_DML_XMIT_TEST1 SYSIN
m_JobBegin -j TEST1 -s START -v 2.0 -c B
```

```

m_JobSetExecLocation "ATLANTA"
while true ;
do
    m_PhaseBegin
    case ${CURRENT_LABEL} in
(START)
        JUMP_LABEL=STEP01
        ;;
(STEP01)
        m_OutputAssign -c "*" SYSPRINT
        m_FileAssign -i SYSIN
        m_FileDelete ${DATA}/GBOM.J.PRD.ABOMJAW1.ABEND02
        m_RcSet 0
_end

        m_UtilityExec
        JUMP_LABEL=END_JOB
        ;;
(END_JOB)
        break
        ;;
(*)
        m_RcSet ${MT_RC_ABORT:-S999} "Unknown label : ${CURRENT_LABEL}"
        break
        ;;
esac
m_PhaseEnd

```

## Using Batch Runtime

```
done
m_JobEnd
_DML_XMIT_TEST1
    m_ProgramExec artjesadmin -i ${DD_SYSIN}
    JUMP_LABEL=END_JOB
    ;;
(END_JOB)
    break
    ;;
(*)
    m_RcSet ${MT_RC_ABORT:-S999} "Unknown label : {CURRENT_LABEL}"
    break
    ;;
esac
m_PhaseEnd
done
m_JobEnd
```

---

In the above sample, job `TEST1` will be submitted by the current job and executed by the `ARTJESINITIATOR` which belongs to JES's Tuxedo server group `ATLANTA`.

## File Catalog Support

This section contains the following topics.

- [General Introduction](#)
- [Database Table](#)
- [Configuration Variables](#)
- [External Shell Scripts](#)

- [External Dependency](#)

## General Introduction

With file catalog support in Batch Runtime, users can access dataset under volumes. A volume is a dataset carrier and exists as a folder; each dataset should belong to a volume.

File catalog contains the mapping from each dataset to each volume. When referencing an existing and cataloged file on Mainframe, file catalog will be requested to find out the volume in which the file is located, and then the file will be accessed.

If file catalog functionality is disabled, the behavior in Batch Runtime remains the same as it is without such functionality.

## Database Table

This table shows the general management for file catalog functionality by Batch Runtime. In this table, each row represents one file-to-volume mapping.

**Table 3-14 Batch Runtime Catalog**

Name	Type	Description
FILENAME	VARCHAR ( 256 )	The file name. It cannot contain any slash.
VOLUME	VARCHAR ( 256 )	The volume name. It cannot contain any slash.
VOLUME_ATTR	CHAR ( 1 )	Reserved.

Primary Key: PK\_ART\_BATCH\_CATALOG

## Configuration Variables

Three configuration variables are required to be added in `BatchRT.conf` or be set as environment variables.

### MT\_USE\_FILE\_CATALOG

If it is set to yes (`MT_USE_FILE_CATALOG=yes`), the file catalog functionality is enabled; otherwise, the functionality is disabled.

### **MT\_VOLUME\_DEFAULT**

If no volumes are specified when a new dataset is created, Batch Runtime uses the volume defined by `MT_VOLUME_DEFAULT`. `MT_VOLUME_DEFAULT` contains only one volume. For example, `MT_VOLUME_DEFAULT=volume1`.

### **MT\_DB\_LOGIN**

This variable contains database access information. Since the file catalog is stored in database, Batch Runtime accesses it through `MT_DB_LOGIN`. For Oracle, its value is `username/password@sid`, such as `scott/tiger@gdg001`. For Db2, its value is `your-database USER your-username USING your-password`, such as `db2linux USER db2svr USING db2svr`.

## **External Shell Scripts**

You can use `CreateTableCatalog[Oracle|Db2].sh` or `DropTableCatalog[Oracle|Db2].sh` to create or drop the new database table.

### **CreateTableCatalog[Oracle|Db2].sh**

#### **Description**

Creates table `ART_BATCH_CATALOG` in database.

#### **Usage**

```
CreateTableCatalog[Oracle|Db2].sh <DB_LOGIN_PARAMETER>
```

#### **Sample**

```
CreateTableCatalogOracle.sh scott/tiger@orcl
```

### **DropTableCatalog[Oracle|Db2].sh**

#### **Description**

Drops table `ART_BATCH_CATALOG` from database.

#### **Usage**

```
DropTableCatalog[Oracle|Db2].sh <DB_LOGIN_PARAMETER>
```



## Sample

```
DropTableCatalogOracle.sh scott/tiger@orcl
```

## External Dependency

To use file catalog functionality in Batch Runtime, File Converter and JCL Converter in ART Workbench should enable catalog functionality. For more information, please refer to [Oracle Tuxedo Application Rehosting Workbench User Guide](#).

## Using Batch Runtime

# Best Practices

## Adapting z/OS Capabilities on a UNIX/Linux Environment

Due to the fact that the Batch Runtime is generally used to execute Korn shell scripts issued from the migration of a z/OS JCL asset, several specific features are provided in order to reproduce some capabilities of z/OS.

The usage of some of these functions may not have a lot of sense in the target platform when modifying migrated jobs or writing new ones.

In this chapter, we present some of these features along with other best practices that we recommend.

### Defining Paths for Procedures, Includes and Programs

In z/OS JCLs, the following cards are used to define the libraries where procedures, includes and programs are stored:

- JOBLIB, STEPLIB for programs.
- JCLLIB for procedures and steps.

Oracle Tuxedo Application Runtime for Batch offers the functions `m_JobLibSet`, `m_StepLibSet` and `m_JclLibSet` as a replacement to these statements.

Even if these functions provide the same functionality, for modified and new jobs we encourage you to adopt the UNIX common rule which is to directly set the environment variables where the programs, procedures and includes are searched for.

The main variables to set are:

- **PATH** : environment variable that specifies where to find executable programs.
- **COBPATH** : environment variable that specifies where to find object Cobol programs.
- **PROCLIB** : environment variable that specifies where to find procedures and includes.

## **Prohibiting the Use of UNIX Commands**

In order to trap every possible error or abnormal end, it is better to avoid using basic UNIX commands (for example: `cp / ls / ...` ).

We recommend that you use only the functions provided by the Batch Runtime.

## **Avoiding the Use of File Overriding**

In order to keep jobs simple and understandable, we recommend you avoid the using of file overriding mechanism in new or modified jobs.

# Using Tuxedo Job Enqueueing Service (TuxJES)

This chapter contains the following topics:

- [Overview](#)
- [Configuring a TuxJES System](#)
- [Using TuxJES](#)

## Overview

The batch job system is an important mainframe business application model. The Tuxedo Job Enqueueing Service (TuxJES) emulation application provides smooth mainframe application migration to open systems. TuxJES implements a subset of the mainframe JES2 functions (for example, submit a job, display a job, hold a job, release a job, and cancel a job).

TuxJES addresses the following batch job phases:

- Input
- Conversion
- Processing
- Purge

## Requirements

TuxJES is an Oracle Tuxedo application; Oracle Tuxedo is required in order to run TuxJES.

A shared file system (for example, NFS) is required in order to deploy TuxJES in distributed environment.

## TuxJES Components

TuxJES includes the following key components:

- `genjesprofile`  
Generates the security profile for Oracle Tuxedo applications
  - `artjesadmin`  
TuxJES command interface. It is an Oracle Tuxedo client
  - `ARTJESADM`  
TuxJES administration server. It is an Oracle Tuxedo server.
  - `ARTJESCONV`  
TuxJES conversion server. It is an Oracle Tuxedo server.
  - `ARTJESINITIATOR`  
TuxJES Job Initiator. It is an Oracle Tuxedo server.
  - `ARTJESPURGE`  
TuxJES purge server. It is an Oracle Tuxedo server.
- For more information, see the Oracle Tuxedo Application Runtime for Batch Reference Guide.

## Configuring a TuxJES System

### Setting up TuxJES as an Oracle Tuxedo Application

TuxJES is an Oracle Tuxedo application. Most of the TuxJES components are Oracle Tuxedo client or Oracle Tuxedo servers. You must first configure TuxJES as an Oracle Tuxedo application. The environment variable `JESDIR` must be configured correctly which points to the directory where TuxJES installed.

### Oracle Tuxedo Configuration File

[Listing 1](#) shows is an Oracle Tuxedo configuration file (UBBCONFIG) example segment for a TuxJES system.

**Listing 1 Oracle Tuxedo UBBCONFIG File Example for the TuxJES System**

---

```

*GROUPS

QG
    LMID=L1 GRPNO=2 TMSNAME=TMS_QM TMSCOUNT=2
    OPENINFO="TUXEDO/QM:/jes2queue/QUE:JES2QSPACE"

ARTG
    LMID=L1 GRPNO=4

EVTG
    LMID=L1 GRPNO=8

*SERVERS
DEFAULT:
    CLOPT="-A"

TMUSREVT    SRVGRP=EVTG SRVID=1 CLOPT="-A"

TMQUEUE
    SRVGRP = QG  SRVID = 1
    RESTART = Y CONV = N MAXGEN=10
    CLOPT = "-s JES2QSPACE:TMQUEUE -- -t 5 "

ARTJESADM    SRVGRP =ARTG  SRVID = 1 MIN=1 MAX=1
    CLOPT = "-A -- -i jesconfig"

ARTJESCONV    SRVGRP =ARTG  SRVID = 20 MIN=1 MAX=1
    CLOPT = "-A --"

ARTJESINITIATOR    SRVGRP =ARTG  SRVID = 30
    CLOPT = "-A -- -c ABCDEFG"

ARTJESPURGE    SRVGRP =ARTG  SRVID = 100
    CLOPT = "-A --"

```

---

The following TuxJES servers should be included in the Oracle Tuxedo configuration file (UBBCONFIG):

- ARTJESADM
- ARTJESCONV
- ARTJESINITIATOR
- ARTJESPURGE

**Note:** Multiple instances of ARTJESADM, ARTJESCONV, ARTJESINITIATOR and ARTJESPURGE can be configured.

For the TuxJES administration server ARTJESADM, a TuxJES configuration file should be specified using the `-i` option. In the Oracle Tuxedo configuration file (UBBCONFIG), ARTJESADM should be configured in front of ARTJESCONV, ARTJESINITIATOR, or ARTJESPURGE servers.

For more information, see the Oracle Tuxedo Application Runtime for Batch Reference Guide.

TuxJES uses the Oracle Tuxedo /Q component, therefore an Oracle Tuxedo group with an Oracle Tuxedo messaging server TMQUEUE with TMS\_QM configured is required in the UBBCONFIG file. The name of the /Q queue space should be configured as JES2QSPACE.

TuxJES uses the Oracle Tuxedo Event component, therefore an Oracle Tuxedo user event server, TMUSREVT is required in the UBBCONFIG file.

A TuxJES system can be either an Oracle Tuxedo SHM application which runs on a single machine, or an Oracle Tuxedo MP application which runs on multiple machines.

For more information on how to set up Oracle Tuxedo application, see Oracle Tuxedo related documentation.

### **Block Time in UBBCONFIG for TuxJES**

You can specify the number of timeout periods for blocking messages, transactions, and other system activities by setting the `SCANUNIT` and `BLOCKTIME` parameter. The value you assign must be a positive multiple of 5.



**Table 1 Characteristics of the SCANUNIT and BLOCKTIME Parameters**

Parameter	Characteristics
SCANUNIT	Controls the granularity of checking intervals and timeouts. SCANUNIT must be a multiple of 5 and between 0 and 60 seconds. Example: SCANUNIT 20 The default is 10.
BLOCKTIME	BLOCKTIME controls how much time can a message block before it times out. SCANUNIT * BLOCKTIME must not exceed 32767. The default time of SCANUNIT * BLOCKTIME is approximately 60 seconds.

**Listing 2 Example Settings**

```

*RESOURCES
IPCKEY      113333
DOMAINID    jesdomain
MASTER      SITE1
MODEL       SHM
MAXACCESSERS 200
MAXSERVERS  50
NOTIFY      SIGNAL
SCANUNIT    20
BLOCKTIME   50

```

In this example, sanity scans are performed in every 20 seconds and request block for no more than  $20 * 50 = 1000$  seconds.

## Oracle Tuxedo /Q Queue Space and Queue Creation

A /Q queue space with name JES2QSPACE must be created for a TuxJES system. And some /Q queues should be created within this queue space. TuxJES provides a sample shell script

(`jesqinit`) to create the queue space (`JES2QSPACE`) and the queues. For more information, see the Oracle Tuxedo Application Runtime Batch Reference Guide.

## File System Configuration

TuxJES uses a file system to communicate with Batch Execution Engine. A directory is created on the file system for the communication between TuxJES and Batch Execution Engine. The name of the directory should be specified in the TuxJES configuration file. This directory should reside at a shared file system (for example, NFS) if you want to deploy the TuxJES system on multiple machines.

## TuxJES Configuration File

A configuration file can be specified for the TuxJES administration server `ARTJESADM`. The following parameters can be configured in the configuration file:

### `JESROOT`

The root directory to store job information. It is a mandatory attribute. If this directory does not exist, `ARTJESADM` creates it automatically.

### `DEFAULTJOBCLASS`

The default job class if the job class is not set in `JCL`. It is an optional attribute. The default job class is `A` if this attribute is not set.

### `DEFAULTJOBPRIORITY`

The default job priority if the job priority is not set in `JCL`. It is an optional attribute. The default job priority is `0` if this attribute is not set.

### `DUPL_JOB=NODELAY`

If it is not set, only one job can be in execution status for a job name. `NODELAY` will remove the dependency check. The default value is delay execution.

### `NJESUPPORT=ON`

If it is not set, NJE support will be disabled and thus jobs cannot be run on the specified server group by Batch Runtime API `m_JobSetExecLocation`. The default value is `OFF`.

### `EVENTPOST=S,C,E,P,L,A`

Specifies whether events are posted for a job at particular stages.

`S`: Job submission event.

`C`: Job conversion complete event.

`E`: Job execution complete event.

`P`: Job purge event.

L: Job cancel completed event.

A: all supported events

If `EVENTPOST` is not specified, no events are posted. The data buffer with event post is FML32 type and the fields are defined in `tuxjes/include/jesflds.h`.

#### JOBREPOSITORY

The path of the job repository where jobs are stored. The script file path inputted in job submitting may be a relative path in `JOBREPOSITORY` if it is set.

#### PRIVILEGE\_MODE

Specifies whether and how to enable the user substitution (See [TuxJES User Substitution](#)). The values are:

`NONE`: Default value. Indicates jobs are executed by the OS user who starts JES system. This is compatible with all previous implementations on JES system.

`USER_IDENTICAL`: Indicates jobs are executed by the Oracle Tuxedo user with which JES client joins JES system. Make sure that each Oracle Tuxedo user corresponds to an existing OS user before you choose this value.

`USER_MAPPING`: When this value is specified, the JES system looks up the TuxJES user mapping file and finds out the OS user corresponding to the Oracle Tuxedo user with which JES client joins JES system, and then appoints this OS user as the job executor.

#### USER\_MAPPING\_FILE

The full path where TuxJES user mapping file is stored. It is used along with `PRIVILEGE_MODE` when its value is `USER_MAPPING`.

## TuxJES Security Configuration

TuxJES leverages the Oracle Tuxedo security mechanism to implement authentication. If authentication is enabled, a security profile should be generated using the `genaprofile` utility and it should be used as a `artjesadmin` parameter to access the TuxJES system. The user used in the profile will be the job owner. A job only can be administrated by its owner, such as cancel, purge, hold and release. A job can be viewed by everybody. If a job is without owner, it can be manipulated by everyone.

Even if an Oracle Tuxedo application does not have security configured, the `genjesprofile` utility still can be used to enforce job owner permission checking and store the database connection `MT_DB_LOGIN`.

## TuxJES User Mapping File

User mapping file is loaded and takes effect when `PRIVILEGE_MODE` value is specified to `MAPPING_CREDENTIAL`. It defines the mapping relationship between Oracle Tuxedo users and OS users. Every line in the mapping file is in the format as below:

```
tuxedousername OSusername
```

It is recommended that the owner of user mapping file is root and the file permission is `"-rw-----"`.

[Listing 3](#) shows a segment example of user mapping file for the TuxJES system.

### Listing 3 User Mapping File Example For the TuxJES System

---

```
tuxedouser1 OSuser1  
tuxedouser2 OSuser2
```

---

## Setting Up TuxJES in MP Mode

TuxJES now can be easily configured within MP mode. For the purpose of running job, however, the configuration in both EJR and TuxJES need to be adjusted so that jobs can be run in parallel on different machines. This section clarifies the configuration mandatory for configuring Batch Runtime in MP mode.

Being shared by all the servers on different machines in a TuxJES domain, the data of jobs should be located on a shared storage (NFS) and can be accessible by all machines in the domain. In addition, the NFS should be mounted with the same mount point on all machines. Finally, `JESROOT` should be configured correctly on each node to point to the shared JES Root Directory. During runtime, all the TuxJES servers on any machine would write data to or get data from such shared `JESROOT`.

For the details of configuring EJR in MP mode, see [“Configuring Batch Runtime in MP Mode”](#) under [“Using Batch Runtime”](#).

## Using TuxJES

After the TuxJES system starts, you can use the `artjesadmin` utility to submit a job, hold a job, release a job, cancel a job, purge a job, display the job information, or subscribe event for job status change.

### Submitting a Job

You can submit a job using the `artjesadmin` subcommand `submitjob`:

#### `submitjob(smj) -i scriptfile`

The `scriptfile` parameter is the job script to be submitted. The job script is generated by Oracle Tuxedo ART Workbench from a JCL. It can be an absolute path format, a relative path in the current working directory, or a relative path in `JOBREPOSITORY` if it is set. Its length is limited to 1023.

`artjesadmin` also supports direct job submission using the following format:

```
artjesadmin -i scriptfile
```

#### `submitjob(smj) -I JCLFile`

The `JCLFile` parameter is the JCL job to be submitted. It can be an absolute path format, a relative path in the current working directory, or a relative path in `JOBREPOSITORY` if it is set. Its length is limited to 1023.

`artjesadmin` also supports direct job submission using the following format:

```
artjesadmin -I JCLFile
```

### Submitting a Job in Synchronous Way

You can submit a job in synchronous way by using `artjesadmin` with the following format:

```
artjesadmin [-f [security_profile]] [-o ejr_option] [-s shell_option] [-y  
[-t timeout(s)]] -i/-I scriptfile
```

**Note:** To submit a job in synchronous way, in TuxJES Configuration File, it's required to set `EVENTPOST=A`; in `UBBCONFIG` file, it's required to set `NOTIFY` to `DIPIN` and set server `TMUSREVT`.

#### `-y` and `-t`

##### Descriptions

Option `-y` and `-t` are added to submit a job in the synchronous way. [Table 2](#) shows some details.

**Table 2 Option -y and -t Descriptions**

Option	Value Range	Descriptions	Notes
-y	N/A	Enables synchronous mode to wait for job end.	N/A
-t	1 ~ (2 <sup>31</sup> -1)	Specifies the timeout value.	Optional. If -t is omitted, <code>artjesadmin</code> will wait infinitely. While timeout occurs, <code>artjesadmin</code> command line will exit but the job will run by JES continuously without impact.

**Exit Code**

[Table 3](#) lists the exit codes for `artjesadmin` if `-y` is specified.

**Table 3 Exit Code**

Exit Code	Descriptions	Notes
0	Job is finished successfully.	N/A
1	Command execution fails.	The code will be returned by either invalid timeout value or a command line syntax error.
2	Job conversion fails.	Job is submitted successfully, but job conversion fails.
3	Job execution fails.	Job is submitted successfully, but job execution fails.
4	Job is canceled.	Job is submitted successfully, but is canceled before reaching final status.
20	Timeout occurs.	Job has not been finished within the time specified by <code>-t</code> option.

**Standard Output**

Information shown on [Table 4](#) will be printed to stdout in the following format.

```
<JOBID> , <JOBNAME> , <JOBSTATUS> , <JOB RETURN CODE>
```

**Table 4 Standard Output**

<b>Output Content</b>	<b>Descriptions</b>	<b>Sample</b>
<JOBID>	Job ID	00005097
<JOBNAME>	Job name	JOBA
<JOBSTATUS>	Job final status (only available if job is finished before the timeout occurs)	DONE
<JOB RETURN CODE>	Job return code from EJR (only available if a job is finished before the timeout occurs)	C000

**Listing 4 Sample: Job is Executed Successfully**

```
00000002,JOBA,DONE,C0000
```

**Listing 5 Sample: Job Fails**

```
00000002,JOBA,FAILED,U0568
```

**Listing 6 Sample: Timeout Occurs**

```
00000002,JOBA,Already Timeout!
```

## Displaying Job Information

You can display the information of a job or a series of jobs using the `art jesadmin` subcommand `printjob`:

```
printjob(ptj) -n jobname | -j jobid | -c job_class [-a [-v]
-n jobname: Display jobs with given job name
-j jobid: Display a particular job information
```

## Using Tuxedo Job Enqueueing Service (TuxJES)

- c `job_class`: Display a particular class jobs information
- a: Display all jobs
- v: Verbose mode

The output of the `printjob` subcommand includes:

- **JOBNAME**: The job Name
- **JobID**: The Job ID generated by TuxJES system
- **Owner**: The submission user of the job
- **Prty**: Priority of the job
- **C**: Job Class
- **Status**: Job Status
  - EXECUTING: a job is running
  - CONVING: a job waiting for conversion
  - WAITING: a job waiting for execution
  - DONE: a job finished successfully
  - FAIL: a job finished but failed
  - HOLD\_WAITING: a job is in hold state after conversion
  - HOLD\_CONVING: a job is in hold state without conversion
  - INDOUBT: a job is in doubt state due to its initiator restarted
  - CANCELED: a job is canceled
- **Submit time**: The submit time of the job
- **Step**: The current running job step. It is only applicable to running jobs.
- **Type Run**: The TYPRUN definition of the job.
- **Machine**: Only for running/done/failed jobs. It is the machine name that the job is/was running on.
- **CPU usage**: The user CPU usage and system CPU usage for the job execution.
- **Execution status**: Job execution status.
- **Result**: Job operation result, "OK" or error message.



**Note:** If there are too many jobs in JES2 system, printing all jobs' status in console may lead to time out; to avoid this situation, users need to configure long enough block time in ubbconfig of JES.

For more information about how to set block time, please refer to [Block Time in UBBCONFIG for TuxJES](#).

## Getting Job Status in Synchronous Way

You can get job status in synchronous way by using `artjesadmin` with the following format:

```
artjesadmin [-f [security_profile]] -p -j jobid
```

### -p and -j

#### Descriptions

Option `-p` and `-j` are added to get job status without interaction in `artjesadmin` console.

#### Exit Code

[Table 5](#) lists the exit codes for `artjesadmin` if `-p` is specified.

**Table 5 Exit Code**

Exit Code	Descriptions	Notes
0	Job is finished normally. Job status = DONE	A job is finished successfully.
1	Command execution fails.	The failure is caused by an internal error, a network error, or a syntax error.
3	Job status = FAIL	JOB execution fails.
4	Job status = CANCEL	A job is canceled.
5	Job status = CONVING	A job is waiting for conversion.
6	Job status = EXECUTING	A job is running.
7	Job status = HOLD_CONVING	A job is in hold state without conversion.
8	Job status = HOLD_WAITING	A job is in hold state after conversion.
9	Job status = WAITING	A job is waiting for execution.

**Table 5 Exit Code**

Exit Code	Descriptions	Notes
10	Job status = DISCARD	This status will occur if <code>topenqueue ( )</code> fails.
11	Job status = INDOUBT	When a job is running, if JES server ARTJESINITIATOR is shutdown and then restarted, the job status will be INDOUBT.
22	Job doesn't exist.	N/A

### Standard Output

Information shown on [Table 6](#) will be printed to stdout in the following format.

<JOBID> , <JOBNAME> , <JOBSTATUS> , <JOB RETURN CODE>

**Table 6 Standard Output**

Output Content	Descriptions	Sample
<JOBID>	Job ID	00005097
<JOBNAME>	Job name	JOBA
<JOBSTATUS>	Job current status	DONE
<JOB RETURN CODE>	Job return code from EJRC (only available if a job has finished)	C000

### Listing 7 Sample: Job has been Finished Normally

```
00000002,JOBA,DONE,C0000
```

### Listing 8 Sample: Job is Finished but Fails

```
00000002,JOBA,FAILED,U0568
```

**Listing 9 Sample: Job is Running**

---

```
00000002,JOBA,EXECUTING
```

---

## Holding a Job

You can hold a job or a series of jobs which are in `CONVING` or `WAITING` status using the `artjesadmin` subcommand `holdjob`:

```
holdjob(hj) -n job name | -j jobid | -c job_class | -a
-n jobname: hold jobs with given job name
-j jobid: hold a particular job
-c job_class: hold a particular class jobs
-a: hold all jobs
```

## Releasing a Job

You can release a job or a series of jobs which are in `HOLD_WAITING` or `HOLD_CONVING` status using the `artjesadmin` subcommand `releasejob`:

```
releasejob(rlj) -n job name | -j jobid | -c job_class | -a
-n jobname: release jobs with given job name
-j jobid: release a particular job
-c job_class: release a particular class jobs
-a: release all jobs
```

## Canceling a Job

You can cancel a job or a series of jobs using the `artjesadmin` subcommand `canceljob`:

```
canceljob(cj) -n job name | -j jobid | -c job_class | -a
-n jobname: cancel jobs with given job name
-j jobid: cancel a particular job
-c job_class: cancel a particular class jobs
-a: cancel all jobs
```

## Purging a Job

You can purge a job or a series of jobs using the `artjesadmin` subcommand `purgejob`:

```
purgejob(pgj) -n job name | -j jobid | -a
-n jobname: purge jobs with given job name
-j jobid: purge a particular job
-a: purge all jobs
```

Completed jobs in the `DONE` or `FAIL` status are moved to the purge queue. For other jobs, `purgejob` has same effect as `canceljob`. The `purgejob` command does not purge the job directly. The `ARTJESPURGE` server deletes the job from the TuxJES system.

## Displaying/Changing ARTJESINITIATOR Configuration

You can display the number of maximum concurrent executing jobs of an `ARTJESINITIATOR` server using the `artjesadmin` subcommand `printconcurrent`:

```
printconcurrent(pco) -g groupname -i serverid
-g groupname: the Tuxedo group name of the ARTJESINITIATOR server
-i serverid: the Tuxedo server id of the ARTJESINITIATOR server
```

You can change the number of maximum concurrent executing jobs of an `ARTJESINITIATOR` server using the `artjesadmin` subcommand `changeconcurrent`:

```
changeconcurrent(chco) -g groupname -i serverid -n concurrent_num
-g groupname: the Tuxedo group name of the ARTJESINITIATOR server
-i serverid: the Tuxedo server id of the ARTJESINITIATOR server
-n concurrent_num: the number of maximum concurrent executing jobs
```

## Event Subscribing/Unsubscribing

You can subscribe or unsubscribe job status change event using the `artjesadmin` subcommand `event`:

```
event (et) [-t C,E,P,L,A] on|off
C: job conversion complete event
E: job execution finish event
P: job purge event
L: job cancel completed event
```

`A`: all supported events. If the event is set to "on", `A` is the default.

`on | off`: The submission is on or off. the "on" setting can be used with the `-t` option.

After subscribing to an event, you are notified on the `artjesadmin` console when the corresponding event is generated.

## See Also

- [Oracle Tuxedo Application Runtime for Batch Reference Guide](#)

## Using Tuxedo Job Enqueueing Service (TuxJES)