Oracle Tuxedo Application Runtime for Batch

Users Guide

11*g* Release 1 (11.1.1.3.0)

December 2011

ORACLE®

Contents

Best Practices

Using Tuxedo Job Enqueueing Service (TuxJES)

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 user 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 detailed information about the Batch Runtime, specifically on how to code the different functions, see the Oracle Tuxedo Application Runtime Reference Guide

CHAPTER **2**

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 reproduce 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

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 messageand 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. See The Rehosting Workbench Installation Guide.

Table 3-1 lists the environment variables 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 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 BatchRT.conf configuration file).
MT_COBOL	Depending on the used COBOL, must contain: - "COBOL_MF" for MicroFocus - "COBOL_IT" for CobolIT (see the BatchRT.conf configuration file)

Table 3-2 Oracle Tuxedo Application Runtime for Batch environment variables

Variable	Usage
MT_CTL_FILES	Directory where the control file (CTL) used by the function m_DBTableLoad (sqlldr with ORACLE, load and export with UDB).
MT_DB	Depending on the target data base, must contain : - "DB_ORACLE" for ORACLE 11g - "DB_DB2LUW" for UDB 9.1 (see the BatchRT.conf configuration file)
MT_DB_LOGIN	Database connection user.
MT_FROM_ADDRES S_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_KSH	Path of the used "ksh" (pdksh or ksh88 only)
MT_LOG	Logs directory (without TuxJes).
MT_ROOT	Directory where the Batch Runtime application has been installed. (see the BatchRT.conf 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).
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 BatchRT.conf configuration file)
MT_SYSOUT	Sysout directory (without TuxJes).
MT_TMP	Directory for temporary internal files (see the BatchRT.conf configuration file).

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 3-3. 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 3-3 Script Structure

Script	Description
#!/bin/ksh#	
m_JobBegin -j *JOBNAME* **-s START -v 2.00**	m_JobBegin is mandatory and must contain at least the following options: • -j: internal job name • -s: name of the first label to begin execution (usually should be START) • -v: Minimum version number of Batch Runtime required for this script (upward compatible).
while true ;do	The "while true; do" loop provides a mechanism to simulate the movement from one step to the next.
m_PhaseBegin	m_PhaseBegin enables parameters to be initialized at the beginning of a step.

Table 3-3 Script Structure

Script	Description
`case ${CURRENT_LABEL} in`	The case statement enables a branching to the current step.
`(START)`	The start label (used in the -s option of m_JobBegin)
`JUMP_LABEL=STEP1`	JUMP_LABEL is mandatory in all steps and gives the name of the next step.
`;;`	;; ends a step and are mandatory.
(STEP1)	A functional step begins with (LABEL); where LABEL is the name of the step.
m_* *m_**	A typical step continues with a series of calls to Batch Runtime functions.
JUMP_LABEL=*STEP2*	There is always a branching to the next step (JUMP_LABEL=)
`;;`	And always the ;; at the end of each step.
(*PENULTIMATESTEP*)	
m_* m_*	The last functional step has the same format as the others, except…

`JUMP_LABEL=END_JOB`
 `;;`
 `(END_JOB)`
	For the label, which must point to END_JOB. The _ is necessary, because the character is forbidden on z/OS.

`break`
 `;;`
 `(*)`
	This step enables the processing loop to be broken.

`m_RcSet ${MT_RC_ABORT:-S999} "Unknown label : ${CURRENT_LABEL}"`
 `break`
 `;;`
 `esac`
	This is a catch-all step that picks-up branching to unknown steps.

Table 3-3 Script Structure

Script	Description
`m_PhaseEnddone`	m_PhaseEnd manages the end of a step including file management depending on disposition and return codes.
`m_JobEnd`	m_JobEnd manages the end of a job including clearing-up temporary files and returning completion code to job caller.

Script Example

shows a Korn shell script example.

Listing 3-1   Korn shell Script Example

```ksh
#!/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=ENDJOB

        ;;

(ABORT)

        break

        ;;

(ENDJOB)

        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 ${DATA}/PJ01DDD.BT.QSAM.KBSTO0$[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

      ;;
```

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 3-4 is an example.

Listing 3-4   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.KBIEO005

     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 3-5

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 3-5   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 3-6

Listing 3-6   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 3-7 and Listing 3-8 are examples.

Listing 3-7   Defining Procedure Example

```
m_ProcBegin   PROCE

        JUMP_LABEL=STEPE

        ;;

(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 3-8   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 3-9 shows how to replace the assignment of the logical file SYSUT1 using the m_FileOverride function.

Listing 3-9   m_FileOverride Function Example

```
 m_ProcBegin  PROCE

      JUMP_LABEL=STEPE

      ;;

(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   m_FileOverride Procedure Call:

```
(COPIERE)

        m_ProcInclude PROCE 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 3-11. These functions can be nested up to 15 times.

Listing 3-11 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 3-12.

Listing 3-12   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_CondEnd` 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.

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 a 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 3-13 Example of File Assignment

```
(STEPCP01)

        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 3-14 Example of Using a File by a COBOL Program

```
(STEPCBL1)

        m_FileAssign -d OLD INFIL ${DATA}/PJ01DDD.BT.QSAM.KBIFI091

        m_FileAssign -d MOD OUTFIL ${DATA}/PJ01DDD.BT.QSAM.KBIFO091

        m_ProgramExec BIFAB090

…
```

Using a Generation File (GDG)

In order to reproduce the notion of generation files, present on the z/OS mainframe, but which is not a UNIX standard, Batch Runtime provides a set of functions to handle this type of file.

Defining a Generation File

A GDG (generation data group) file is defined through the `m_GenDefine` function. The only parameter to be specified is the maximum number of versions to keep on the disk:

```
m_GenDefine -s 31 ${DATA}/PJ01DDD.BT.GDG
```

Note: the function m_GenDefine is not mandatory to define a GDG file. As shown in Listing 3-15, the GDG file will be created with a limit of 9999.

Listing 3-15   Example of using a generation file:

```
(STEP03)

      m_FileAssign -d SHR SYSUT1 ${DATA}/PJ01DDD.BT.FILE1

      m_FileAssign -d NEW,CATLG -g +1 SYSUT2 ${DATA}/PJ01DDD.BT.GDG

      m_FileRepro -i SYSUT1 -o SYSUT2
```

Using a Generation File

The `m_FileAssign` function has a special parameter (`-g`) that serves to specify that the file being assigned is a generation file and to set the desired version of the file as shown in Listing 3-16.

Listing 3-16   Example of Using a Generation File:

```
(STEPDD05)

      m_FileAssign -d SHR -g +1 INFIL ${DATA}/PJ01DDD.BT.GDG

      m_FileAssign -d MOD OUTFIL ${DATA}/PJ01DDD.BT.QSAM.KBDDO002

      m_ProgramExec BDDAB001
```

Adressing a Generation Group

The `m_FileAssign` function may be used to address all the generations of a GDG.

As shown in Listing 3-17, the SORT function envolves all generations of the GDG file named AP.GDG.

Listing 3-17   GDG File Example

```
(PT0010)

       m_FileAssign -d SHR SORTIN ${DATA}/APP.GDG

       m_FileAssign -d NEW,CATLG,DELETE SORTOUT ${DATA}/AP.GDG.SORT

       m_FileAssign -i SYSIN

SORT FIELDS=COPY

_end

       m_FileSort -s SYSIN -i SORTIN -o SORTOUT
```

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 3-18.

Listing 3-18   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 3-19.

Listing 3-19 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 NEW 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.KBSTO045
```

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 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 3-20   RDBMS Connection Example

```
(STEPDD02)

   m_FileAssign -d MOD OUTF ${DATA}/PJ01DDD.BT.QSAM.REPO001

   m_ProgramExec -b DBREP001
```

The `m_ProgramExec` function may submit three types of executable files (Cobol executable, command language script, or C executable). It launchs 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.

The `runb` program, runtime compiled with database librairies, 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 ouput file must contain valid ksh syntax.

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.

LOG File Structure

For each launched job, Batch Runtime produces a log file containing information for each step (phase) that was executed. This log file has the following structure: as shown in Listing 3-21.

Listing 3-21   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).

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

Based on BDB with XA protocal, the Batch COBOL programs compiled by COBOL IT can access the indexed ISAM files which are converted from Mainframe VSAM files through the ART Workbench. To enable this application in Batch runtime, you need to unset `COB_ENABLE_XA` before running Batch COBOL program.

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 jobswe 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 using the 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, ARTJESCNOV, 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, 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.

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.

EVENTPOST=S,C,E,P,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.

>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 genapprofile 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 2 shows a segment example of user mapping file for the TuxJES system.

Listing 2   User Mapping File Example For the TuxJES System

```
tuxedouser1 OSuser1
```

```
tuxedouser2 OSuser2
```

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`

Displaying Job Information

You can display the information of a job or a series of jobs using the `artjesadmin` 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

 -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

- 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.

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 S,C,E,P,A] on|off

S: job submission event

C: job conversion complete event

E: job execution finish event

P: job purge 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