Oracle® Retail Batch Script Architecture

Implementation Guide

Release 14.0

December 2013

ORACLE®

Oracle Retail Batch Script Architecture Implementation Guide, Release 14.0

Primary Author: Melissa Artley

Applications. For purposes of this section, “alteration” refers to all alterations, translations, upgrades, enhancements, customizations or modifications of all or any portion of the VAR Applications including all reconfigurations, reassembly or reverse assembly, re-engineering or reverse engineering and recompilations or reverse compilations of the VAR Applications or any derivatives of the VAR Applications. You acknowledge that it shall be a breach of the agreement to utilize the relationship, and/or confidential information of the VAR Applications for purposes of competitive discovery.

The VAR Applications contain trade secrets of Oracle and Oracle's licensors and Customer shall not attempt, cause, or permit the alteration, decompilation, reverse engineering, disassembly or other reduction of the VAR Applications to a human perceivable form. Oracle reserves the right to replace, with functional equivalent software, any of the VAR Applications in future releases of the applicable program.

Contents

List of Tables

Send Us Your Comments

Oracle Retail Batch Script Architecture Implementation Guide, 14.0

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document.

Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

> **Note:** Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the Online Documentation available on the Oracle Technology Network Web site. It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: retail-doc_us@oracle.com.

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at www.oracle.com.

Preface

The *Oracle Retail Batch Script Architecture Implementation Guide* is intended for integrators and implementation staff, as well as the retailer's IT personnel.

Audience

This Implementation Guide is intended for the Batch Script Architecture application integrators and implementation staff, as well as the retailer's IT personnel. This guide is also intended for business analysts who are looking for information about processes and interfaces to validate the support for business scenarios within Batch Script Architecture and other systems across the enterprise.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following applications that include information about the Oracle Retail Batch Script Architecture (BSA) Release 14.0:

- Oracle Retail Advanced Inventory Planning
- Oracle Retail Analytic Parameter Calculator for Regular Price Optimization
- Oracle Retail Assortment Planning
- Oracle Retail Category Management
- Oracle Retail Demand Forecasting
- Oracle Retail Item Planning
- Oracle Retail Item Planning Configured for COE
- Oracle Retail Merchandise Financial Planning
- Oracle Retail Regular Price Optimization

- Oracle Retail Replenishment Optimization
- Oracle Retail Size Profile Optimization

Customer Support

To contact Oracle Customer Support, access My Oracle Support at the following URL:

https://support.oracle.com/

When contacting Customer Support, please provide the following:

- Product version and program/module name
- Functional and technical description of the problem (include business impact)
- Detailed step-by-step instructions to recreate
- Exact error message received
- Screen shots of each step you take

Review Patch Documentation

When you install the application for the first time, you install either a base release (for example, 14.0) or a later patch release (for example, 14.0.1). If you are installing the base release, additional patch, and bundled hot fix releases, read the documentation for all releases that have occurred since the base release before you begin installation. Documentation for patch and bundled hot fix releases can contain critical information related to the base release, as well as information about code changes since the base release.

Oracle Retail Documentation on the Oracle Technology Network

Documentation is packaged with each Oracle Retail product release. Oracle Retail product documentation is also available on the following Web site:

http://www.oracle.com/technology/documentation/oracle_retail.html

(Data Model documents are not available through Oracle Technology Network. These documents are packaged with released code, or you can obtain them through My Oracle Support.)

Documentation should be available on this Web site within a month after a product release.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1
Introduction

The Batch Script Architecture (BSA) is designed to provide a robust, enterprise-ready architecture for parallel process control, restart control, log consolidation, and dependency checks. Applications that take advantage of the architecture include:

- Oracle Retail Advanced Inventory Planning
- Oracle Retail Analytic Parameter Calculator for Regular Price Optimization
- Oracle Retail Assortment Planning
- Oracle Retail Category Management
- Oracle Retail Demand Forecasting
- Oracle Retail Item Planning
- Oracle Retail Item Planning Configured for COE
- Oracle Retail Merchandise Financial Planning
- Oracle Retail Regular Price Optimization
- Oracle Retail Replenishment Optimization
- Oracle Retail Size Profile Optimization

Functionally, the batch operability is enhanced by making it easier to diagnose and correct problems. Errors and exceptions are isolated (contained) within the applications' domains in which they occur. Processing is designed to easily restart and proceed with execution.

Parallel tasks are managed, logged, and checked for errors, to the same standard as non-parallel tasks.

BSA provides the following features:

- Error handling and logging architecture and their implementation
- Script parallel process management (logging, error control, and waiting) framework

The BSA is a common component architecture that is used by many of the Oracle Retail application batch scripts for the reusing of code. The BSA has a structured approach of execution.

In addition to modularization and complete separation of the generic functionality, additional functionality has been added, including the management of access credentials for database connections, and SQL query, move and copy-with confirmation, and HTML- and XML-based logging.

Contents of This Guide

This implementation guide addresses the following topics:

- Chapter 1, "Introduction"— Overview of BSA and the applications that use BSA architecture.
- Chapter 2, "Getting Started"— Explanation of BSA deployment.
- Chapter 3, "Batch Architecture Functionality"— Information on BSA functionality that includes containment and management of shell environment settings, logging, error handling, and parallel control.

2

Getting Started

Specialized application environment setup is best done by setting up a common script that is sourced by all the specialized application environment scripts. The common script should initialize the specialized environment settings, and then source bsa_common.sh to complete the environment setup. Each specialized environment script should normally cache any input parameters (preferably saved to a local, descriptively-named script variable), and then source the common application script.

Deploying the Batch Script Architecture

This section describes deployment of the BSA.

Preparation and BSA Installation

Get the file bsa.zip from the BSA package. Oracle recommends installing under $RPAS_HOME. The instructions below refer to this folder as {install-folder} and assumes that the file bsa.zip is placed in folder/tmp.

Install as follows:

cd {install-folder}

unzip -q "/tmp/bsa.zip" ## Creates subfolder {install-folder}/bsa.

chmod 755 bsa/*

In the user's `.profile` (or similar setup script) add the "{install-folder}/bsa" to PATH. For example, if the {install folder} location used above were /u00/rpas13x, then you could add this line to your `.profile`:

PATH="/u00/rpas13x/bsa:$PATH"

3

Batch Architecture Functionality

The BSA provides structure and helper functionality for containment and management of shell environment settings, logging, error handling, and parallel control. The following scripts are included:

Script	Description
bsa_common.sh	A convenience script that sources bsa_env.sh, bsa_logger.sh, bsa_verify.sh, and bsa_para.sh for simplified packaging.
bsa_env.sh	Environment settings (paths, and so forth) are optional redefinitions that can be preset in the environment or in a local environment script to specialize for each installation. Also includes routines to programmatically create, verify, and adjust the environment settings.
bsa_logger.sh	Functionality to perform automatic and manual information and error logging.
bsa_para.sh	Functionality to perform parallel process control, error containment, and logging.
bsa_verify.sh	Utility functions to perform verifications of scripts and binary executions.
bsa_cred.sh	Utility functions to manage access credentials (Connect string for Oracle database).
bsa_sql.sh	Utility functions to fetch values from, and runs stored procedures/functions on, the Oracle database.

Logger Functionality Overview

The bsa_logger.sh script contains utility API functions to perform basic information and error logging. At its core are the `_log_message` and `_call APIs` that perform explicit information and error logging, and the wrapping of the standard out and standard error streams to the log. The script also enables simultaneous logging to the terminal.

Automatic Functions

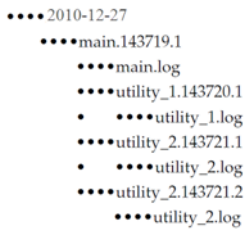
In addition to explicit functionality, the logger.sh header script performs the following automatic functions:

Log path setup: A log directory, ${BSA_LOG_HOME}/<date>, is created. Logs are created in subdirectories and in a folder hierarchy that exactly parallels the script call stack. Log files are named <script base name>.log. Later, a summary log may be implemented. Each log directory name is constructed as follows:

```
<base script name>.<timestamp>.<sequence number>
```

The sequence number is important, as some scripts are called repeatedly in a sequence or in parallel by other scripts. Each distinct call gets its own log. For example, if, on December 27, 2010, the main.sh script calls `utility_1.sh` once and `utility_2.sh` three times, the following log directories and files result:

Figure 3–1 Log Path Setup Example



Script Entry Logging

The _log_message API is automatically called to log the script entry point to the script's log. Refer to the sections, Logger Private API and Logger Public API for additional information.

Script Exit Logging

Upon exit, the `_log_message` API is called to log the script exit to the script's log. Both normal and Ctrl+C and other hard exit-induced exits are logged and distinguished, and a descriptive error code number is included in the log message.

Read from the application's environment shell script (for example, `aip_env_rpas.sh`, `environment.sh`), and create, if absent, the environment variable, BSA_LOG_LEVEL. This is the minimum level for which log content is generated.

Valid values for this environment variable (BSA_LOG_LEVEL) are:

- PROFILE
- DEBUG
- INFORMATION
- WARNING
- ERROR
- NONE

If this variable is not already set by the environment, `logger.sh` defaults the value to INFORMATION.

Read from the application's environment shell script and create, if absent, the environment variable, BSA_SCREEN_LEVEL. This is the minimum level for which log content is copied to the terminal.

Valid values for this environment variable (BSA_SCREEN_LEVEL) are:

- PROFILE
- DEBUG
- INFORMATION
- WARNING

- ERROR
- NONE

If this variable is not already set by the environment, `logger.sh` defaults the value to NONE.

Logger Private API

Automatic functionality is implemented by using the private API, as well as the public API. Table 3–1 describes the Logger Private API Scripts.

Table 3–1 Logger Private API Scripts

Script	Description
__internal_log_message	Logs the message, complete with the caller's line number
__internal_log_entry	Calls __internal_log_message to log the entry to the script's log
__internal_log_normal_exit	Called implicitly by the firing of the shell EXIT trap; calls __ internal_log_message to log the exit code and message to the script's log
__internal_log_forced_exit	Called implicitly by the firing of the shell INT and CLOSE traps; calls __internal_log_message to log the exit code and message to the script's log

Logger Public API

Automatic functionality is implemented by using the private API, as well as the public API. The following table describes the Logger Public API Scripts and Environment Variables:

Table 3–2 Logger Public API Scripts and Environment Variables

Script	Description
BSA_LOG_LEVEL environment variable	The BSA_LOG_LEVEL environment variable, found in the application's environment script (for example, environment.ksh), can be set to the desired level of logging to the log files. Valid values are PROFILE, DEBUG, INFORMATION, WARNING, ERROR and NONE. If this variable is not already set by the environment, logger.sh defaults the value to INFORMATION.
BSA_SCREEN_LEVEL environment variable	The BSA_SCREEN_LEVEL environment variable, found in the application's environment script (for example, environment.ksh), can be set to the desired level of logging to the terminal. Valid values are PROFILE, DEBUG, INFORMATION, WARNING, ERROR and NONE. If this variable is not already set by the environment, logger.sh defaults the value to NONE.
_log_message <level> <message>	Implemented as an alias that captures the script line number and calls __internal_log_message. The level can be PROFILE, DEBUG, INFORMATION, WARNING, ERROR or NONE. It writes the time stamp, level, script name, line number, and message to the script's log and/or terminal, depending on the level as compared with the BSA_LOG_LEVEL and BSA_ SCREEN_LEVEL. This renders obsolete most uses of echo and print to the log file.
_call <script or binary call with params>	Wraps any script or binary call so that the standard error and standard out are redirected to the script's log file. This renders obsolete most uses of > and >>.

Send Log to Screen

BSA allows for logging of output to the screen, simultaneous with logging to log files. This is controlled by the environment variable BSA_SCREEN_LEVEL.

Valid values for this environment variable (BSA_SCREEN_LEVEL) are:

- PROFILE
- DEBUG
- INFORMATION
- WARNING
- ERROR
- NONE

If this variable is not already set by the environment, logger.sh defaults the value to INFORMATION. Logging to file is still controlled by the existing variable BSA_LOG_LEVEL. The two levels for LOG and SCREEN are independent.

Verification of Successful Completion

The verify functions provide single-line error handling after calling any script or binary. The three API functions, _verify_script, _verify_binary, and _verify_log ensure the success or report the failure and exit the batch, based on the run of a script, binary, or the identification of an error string in the script's log.

Verify the Public API

Note: In order for _verify_script to work correctly, all the scripts should exit only with predefined exit codes. Under no circumstances should exit be called without a code, nor should any return be called.

Table 3–3 Verify Script Descriptions

Script	Description
_verify_script	This is implemented as an alias that captures the script line number and calls __internal_verify_script. This function must be called immediately after the script to be verified. It checks the return value from the script; if nonzero, it posts an error code specific message to the script's log with level=ERROR, and then exits from the script with the same code. This, along with _ verify_binary, renders obsolete most processing functions of $?.
_verify_binary <custom error message>	This is implemented as an alias that captures the script line number and calls __internal_verify_binary. This function must be called immediately after the binary to be verified. It checks the return value from the binary; if nonzero, it posts the passed custom error message to the script's log with level=ERROR, and then exits from the script with a standard error code (not the binary exit code). The standard error code is returned so that the calling script can automatically report the error by use of _ verify_script. This, along with _verify_script, renders obsolete most processing functions of $?.

Table 3–3 (Cont.) Verify Script Descriptions

Script	Description
_verify_log	This is implemented as an alias that captures the script line number and calls __internal_verify_log. This function performs grep-based checking of the script's log for an easily configured list of strings, including error, exception, and failure. If a configured string is found, a log entry is generated with level=ERROR, indicating the error found; and then a non-zero exit is performed to immediately halt the script. This renders obsolete most functions in the explicit grepping of log files.

The codes defined for use as script exits are as follows:

__CODE_SUCCESS
__CODE_NONZERO_EXIT
__CODE_FORCED_EXIT
__CODE_FILE_NOT_FOUND
__CODE_NONEXISTENT_LOG_FILE
__CODE_NONEXISTENT_ERROR_FILE
__CODE_TOO_FEW_ARGS
__CODE_TOO_MANY_ARGS
__CODE_UNSPECIFIED_ERROR
__CODE_GENERAL_SCRIPT_ERROR
__CODE_UNDEFINED_ERROR_LEVEL
__CODE_MACE_ERROR_LEVEL
__CODE_INVALID_PATH
__CODE_INVALID_DOMAIN_PATH
__CODE_PARALLEL_ERROR
__CODE_UNSUPPORTED_FUNCTIONALITY
__CODE_UNSUPPORTED_DATA_SET_KEY
__CODE_FIRST_LOG_ERROR
__CODE_LOGGED_ERROR
__CODE_LOGGED_EXCEPTION
__CODE_LOGGED_FATAL
__CODE_LOGGED_ABORT
__CODE_LAST_LOG_ERROR

Overview of the Parallelization Functionality

A framework for the parallelization of script and binary calls is introduced to replace the use of the naked and wait commands. The framework has these three components:

Function	Description
BSA_MAX_ PARALLEL environment variable	The BSA_MAX_PARALLEL environment variable, found in the application's environment script (for example, environment.ksh), should hold a positive integer value. This value is interpreted by the _para_spawn function as the maximum number of processes that can be started in parallel, from any spawning process. This value is not the maximum total number of running processes, as spawned processes may further spawn more processes. If this variable is not already set by the environment, bsa_env.sh defaults the value to 4.

Function	Description
_para_spawn function	The _para_spawn function is a replacement for invoking a script or binary with a trailing '&'. _para_spawn <command> starts command in parallel, as soon as the number of processes spawned by the spawning process falls below BSA_MAX_PARALLEL. Scripts spawned with _para_spawn, log messages just like the non-parallel-called scripts. _para_spawn must be used only in conjunction with _para_wait. For more information, see the section, Para API.
_para_wait function	The _para_wait function is a replacement for the wait command, and it should always be used to wait for processes spawned with _para_spawn. _para_wait handles the detailed reporting of spawned process failures and returns; either __CODE_SUCCESS (0) if all the parallel processes succeed, or __CODE_PARALLEL_ERROR if any spawned process fails. _para_wait must be used only in conjunction with _para_spawn. For more information, see the section, Para API.

Para API

Table 3–4 describes the Para API functions:

Table 3–4 Para API Functions

Function	Description
_para_spawn <script or binary call with params>	This function wraps the calling of a script or binary, in parallel. This renders obsolete the use of standard parallel calling "&". Scripts or binaries called using this function are started only if the number of processes that have already been spawned from the same calling script and still running do not exceed the BSA_MAX_PARALLEL environment variable. If the number of already spawned and still running jobs equals BSA_MAX_PARALLEL, then, this function waits until at least one of the jobs completes, before spawning the requested job. _para_spawn must be used in conjunction with _para_wait. _para_spawn should not be called with the _call function, although the use of _call within a script spawned by _para_spawn is acceptable.
_para_wait	This function wraps the wait function, and should be used to wait for jobs spawned using _para_spawn. It returns 0 if all the jobs spawned with _para_spawn return 0, or nonzero (actually, the resolved value of __CODE_PARALLEL _ERROR), otherwise.

Tar Command Customization

BSA includes functionality for uncompressing and unpacking a file that has been packed with UNIX's tar and compress utilities. The tar command used by BSA is customizable by editing bsa_env.sh and replacing "TAR=tar" with the name or full path of any alternate version of tar required to circumvent standard UNIX tar file size limits.

SQL Query Wrapping

The API functions, _sqlplus and _sqlplus_fetch, are used to simplify the database access through batch scripts. Both functions use the BSA credential file to obtain the

schema access. The credential file application can be sent to the functions. Both functions use DEFAULT_BSA_SQL_CRED_APP for credential lookup if an application is not passed.

The _sqlplus function can be sent a string of PL/SQL commands, including calls to stored procedures and functions. The function propagates connection errors, syntax errors, unhandled database errors, and handled PL/SQL logic errors. The _sqlplus function commits automatically.

The _sqlplus_fetch function can be sent a SQL query. It propagates connection errors, syntax errors, and unhandled database errors.

BSA Setup

Prior to the sourcing of the BSA common architecture scripts, a batch script or its caller must specialize the BSA environment settings as necessary (for example, to set BSA_LOG_HOME to a local, writable directory). The BSA architecture is designed to set the environment settings only if they are not preset by the specialized environment. In other words, set up the application environment, and then source the BSA architecture scripts, which fine tune and default any necessary settings that do not already exist.

Specialized application environment setup is best done by setting up a common script that is sourced by all the specialized application environment scripts. The common script should initialize the specialized environment settings, and then source bsa_common.sh to complete the environment setup. Each specialized environment script should normally cache any input parameters (preferably saved to a local, descriptively-named script variable), and then source the common application script.

The following variables in Table 3–5 should be customized. Some of these have been described more extensively in the preceding documentation.

Table 3–5 BSA Variables

BSA Variable	Default Value	Description
BSA_ARCHIVE_DIR	$TMP	The directory that will contain the copies of archive files created with tar and compress command and imported from data source outside the application.
BSA_CONFIG_DIR	$TMP	The directory containing configuration files used for unpacking of data files.
BSA_LOG_HOME	$TMP	The directory containing the log files generated by the BSA logging functionality (for example, _call).
BSA_LOG_LEVEL	INFORMATION	The desired level of message logging. Used by both logging to screen and to log files. Valid values are PROFILE, DEBUG, INFORMATION, WARNING, ERROR and NONE.
BSA_LOG_TYPE	1	The desired logging type. 1=Text Only. 2=XML Only. 3=Text & XML.
BSA_MAX_PARALLEL	4	The maximum number of processes that can be started in parallel, from any spawning process.
BSA_SCREEN_LEVEL	NONE	The desired level of logging to the terminal. Valid values are PROFILE, DEBUG, INFORMATION, WARNING, ERROR, and NONE.
BSA_TEMP_DIR	$TMP	Directory used to store temporary files, for example: for sorting temporary space, for one-off files created during batch processes.