

Oracle Tuxedo Application Rehosting Workbench

Reference Guide

11g Release 1 (11.1.1.2)

July 2011

ORACLE®

Copyright © 2011, 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 software or related documentation 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 RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software 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 which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software 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:

1. Introduction

Concepts	1-1
----------------	-----

2. Codeset Conversion

Overview of Codeset Conversion	2-1
Purpose	2-1
Audience	2-2
z/OS – Displaying Graphical Characters	2-2
Reference Monitor	2-2
REFCOD80 File	2-2
Viewing Characters Under z/OS	2-3
UNIX/Linux: Displaying Graphical Characters	2-6
Reference Monitor	2-6
COBOL CONVERTMW.cpy	2-6
Validating and Adapting the Transcoding Copy File	2-18
Validation	2-18
Adapting the COBOL CONVERTMW.cpy Copy File	2-19
Finding the z/OS Character	2-19
Finding the UNIX Characters to Replace	2-20
Replacing the UNIX Character	2-21
Remarks Concerning the Example	2-21
Special Characters	2-21

Using the COBOL CONVERTMW.cpy File.	2-21
Error Messages.	2-22
See Also	2-22

3. Cataloger

Overview of the Cataloger	3-1
Inputs to the Cataloger Process.	3-1
Outputs from the Cataloger Process	3-2
The Cataloger Process.	3-2
Description of the Input Components	3-3
COBOL	3-3
References	3-3
Restrictions	3-3
Embedded CICS	3-4
References	3-4
SQL	3-4
References	3-4
JCL	3-4
References	3-4
General Information.	3-4
Sub-Files.	3-4
JCL Syntax	3-5
Restrictions	3-5
BMS screen definition.	3-6
CICS Configuration	3-7
Description of the Configuration Files.	3-7
System Description File	3-7
General Structure	3-7

Global Options	3-8
Special Options.	3-11
Directories	3-12
files Clause	3-14
logical-name clause	3-14
options-clause.	3-15
libraries-clause	3-15
sql-libraries-clause	3-16
Example of System Description File	3-16
JCL-Launcher Specification Files	3-19
Purpose	3-19
Syntax	3-19
Option List	3-20
Usage and Default Value	3-20
Description of the Output Files.	3-21
Catalog Reports	3-21
Format and Location	3-21
Field Definitions.	3-21
report- <code>\${SYSNAME}</code> -COBOL-Programs.	3-22
report- <code>\${SYSNAME}</code> -COBOL-Copy	3-23
report- <code>\${SYSNAME}</code> -JCL-Files	3-24
report- <code>\${SYSNAME}</code> -JCL-Sub-Files	3-24
report- <code>\${SYSNAME}</code> -JCL-Jobs	3-25
report- <code>\${SYSNAME}</code> -Screens	3-26
report- <code>\${SYSNAME}</code> -SQL-Tables	3-27
report- <code>\${SYSNAME}</code> -SQL-Views	3-27
report- <code>\${SYSNAME}</code> -Transactions.	3-28
report- <code>\${SYSNAME}</code> -Anomalies	3-29

Execution Logs	3-30
Description of Other Output Files.	3-30
POB Files for ASTs	3-30
CDM Files for COBOL Programs and Copy Files	3-31
The Cataloger Symtab and Other Miscellaneous Files	3-31
Detailed Processing	3-31
Processing Phases	3-31
Command-line Syntax.	3-33
The Oracle Tuxedo Application Rehosting Workbench Launcher	3-33
System-Wide Commands	3-34
The preparse-files Command	3-35
Component Search Operation	3-36
Compile-Time References.	3-36
Normal Sub-File Search.	3-37
Strict JCL-Sysin Search	3-38
Run-Time Reference	3-38
Unrestricted Search	3-39
Directed Search	3-39
Repetitive and Incremental Operation	3-40
Initial Processing: Repetitive Operation	3-40
Changes in the Asset: Incremental Operation.	3-41

4. DB2-to-Oracle Convertor

Overview of the DB2-to-Oracle Convertor	4-1
Purpose	4-1
Structure	4-2
See Also	4-2
Oracle Tuxedo Application Rehosting Workbench Schema.	4-2

Environment Variables	4-3
Description of the Input Components.	4-3
File Locations	4-3
Location of rdbms.sh	4-3
Location of db-param.cfg File	4-3
DB2 DDL Converted.	4-3
Conversion of DB2 Data Types	4-5
DB2 Column Property Conversion	4-5
Description of the Configuration Files	4-6
POB Files	4-6
DB2 DDL POB File	4-6
Symtab File	4-6
sql-system File	4-7
system.desc.	4-7
db-param.cfg.	4-7
Parameters and Syntaxes.	4-8
Date, Time Parameters	4-9
Index, Sort Parameters	4-9
File Modifying Generated Components.	4-10
Renaming File	4-11
rdbms-template.txt	4-12
rdbms_move_assignment.txt	4-13
Description of the Output Files.	4-15
File Locations	4-15
Location of Temporary Files	4-15
Locations of Log Files	4-15
Location of Generated Files	4-15
Generated Objects	4-16

Temporary Files	4-17
Datamap File	4-17
File Name	4-17
Syntax and Parameters	4-17
Mapper File	4-18
File Name	4-18
Generation Sample	4-19
Syntax and Parameters	4-20
Links to COBOL Copy	4-22
COBOL Description	4-22
Copy File Name	4-23
Copy File Syntax and Parameters	4-24
Unloading JCL	4-25
COBOL Transcoding Programs	4-28
Reloading Korn Shell Scripts	4-29
Transcoding Phase	4-30
Loading Phase	4-31
Check Phase	4-31
Target DDL	4-31
TABLE and COLUMNS	4-31
INDEX	4-32
CONSTRAINT	4-33
COMMENT	4-34
VIEW	4-34
SEQUENCE	4-35
SYNONYM	4-35
Identity Engineering	4-35
Ordered List of Tables File	4-37

COBOL Conversion Guide File	4-37
File Name	4-37
Generated Sample	4-37
SQL*LOADER Control Files	4-38
File Name	4-38
Generated Sample	4-39
DDL Translator Log File	4-40
Execution Reports	4-46
Detailed Processing	4-54
Command-line Syntax	4-54
rdbms.sh	4-54
Unitary Usage Sequence	4-56
Process Steps	4-56
Configuring the Environments and Installing the Components	4-56
Installing the Unloading Components Under z/OS	4-56
Installing the Reloading Components on the Target Platform	4-56
Installing the MWDB2ORA Package Component on the Target Platform	4-57
Unloading Data	4-58
Transferring the Data	4-59
Reloading the Data	4-59
Transcoding and Reloading Command	4-59
Checking the Transfers	4-60

5. File Convertor: Introduction

Overview of the File Convertor	5-2
Purpose	5-2
Structure	5-2

See Also	5-2
File Organizations Processed	5-2
z/OS File Organizations	5-3
File Conversion to File or to RDBMS Table	5-3
Oracle Tuxedo Application Rehosting Workbench Configuration Name.	5-3
File Descriptions and Managing Files With the Same Structure	5-4
COBOL Description	5-4
COBOL Description Format	5-4
COBOL Description and Related Discrimination Rules	5-5
List of the Input Components	5-6
Datamap File	5-6
Datamap Syntax and Parameters	5-7
Mapper File	5-8
Mapping File Clause	5-9
COBOL Description	5-12
POB Files	5-12
Symtab File	5-12
.....	5-12

6. File-to-File Converter

Overview of the File-to-File Converter	6-1
Purpose	6-1
Structure	6-2
See Also	6-2
File Organizations Processed	6-2
Keeping z/OS File Organization on the Target Platform	6-2
PDS File Organization	6-3
GDG File Organization	6-3

Oracle Tuxedo Application Rehosting Workbench Configuration Name	6-3
Environment Variables	6-3
Description of the Input Components.	6-4
File Locations	6-4
Location of file.sh.	6-4
Location of db-param.cfg File	6-4
Description of the Configuration Files	6-4
db-param.cfg.	6-4
Parameters and Syntaxes.	6-5
File Modifying Generated Components.	6-5
file-template.txt	6-6
file-move-assignation.pgm.	6-8
Datamap File	6-10
Mapper File	6-10
Discrimination Rules	6-11
COBOL Description.	6-14
Description of the Output Files.	6-14
File Locations	6-14
Location of Temporary Files	6-14
Location of Log Files	6-14
Location of Generated Files	6-14
Generated Objects	6-15
Unloading JCL	6-15
Unloading JCL for QSAM and VSAM files.	6-16
Unloading JCL for Generation Data Group.	6-18
COBOL Transcoding Programs	6-22
Migration of z/OS Files to UNIX/Linux Files	6-22
Reloading Korn Shell Scripts.	6-23

Reloading Korn Shell Scripts for Migrating z/OS QSAM/VSAM Files to UNIX/Linux Files	6-23
Reloading Korn Shell Scripts for Migrating z/OS Generation Data Set to UNIX/Linux Files	6-24
Transcoding and Loading Phases	6-25
Check Phase	6-25
Access Functions and Utility Programs	6-26
Access Functions	6-26
Access Function Call Arguments	6-27
Call Arguments Used	6-29
OPEN	6-29
CLOSE	6-30
CLOSE-LOCK	6-30
DELETE	6-30
READ	6-30
REWRITE	6-31
START	6-32
WRITE	6-33
Copy Files to Be Implemented	6-34
Execution Reports	6-34
Detailed Processing	6-40
Command-Line Syntax	6-41
file.sh	6-41
Unitary Usage Sequence	6-42
Process Steps	6-42
Configuring the Environments and Installing the Components	6-42
Installing the Unloading Components Under z/OS	6-42
Installing the Reloading Components on the Target Platform	6-42

Compiling COBOL Transcoding Programs	6-43
Unloading Data.	6-43
Transferring the Data	6-44
Reloading the Data	6-44
Transcoding and Reloading Command for Files	6-44
Transcoding and reloading command for Generation Data Group files .	6-45
Synopsis	6-45
Options	6-45
Checking the Transfers.	6-45

7. File-to-Oracle Converter

Overview of the File-to-Oracle Converter	7-1
Purpose	7-1
Structure.	7-2
See Also.	7-2
File Organizations Processed.	7-2
Migrating to Oracle Table on the Target Platform	7-2
Oracle Tuxedo Application Rehosting Workbench Configuration Name.	7-3
VSAM Files Becoming Oracle Table	7-3
Specific Migration Rules Applied	7-3
Rules Applied to Picture Clauses.	7-3
Environment Variables	7-4
Description of the Input Components.	7-4
File Locations	7-4
Location of file.sh.	7-4
Location of db-param.cfg File	7-4
Description of the Configuration Files	7-5
db-param.cfg.	7-5

Parameters and Syntaxes	7-6
File Modifying Generated Components	7-6
file-template.txt	7-7
file-move-assignation.pgm.	7-9
Datamap File	7-11
Mapper File	7-11
Mapping Strategy Clauses	7-14
Mapping Strategy Clause Syntax and Parameters	7-14
Mapping Strategy Examples	7-15
Discard Subfield Example	7-15
Redefines With Default Option Example	7-15
REDEFINES With OPAQUE FIELD Option Example	7-18
REDEFINES With DETAIL TABLE Option Example	7-20
Discrimination Rules	7-23
COBOL Description	7-26
Description of the Output Files	7-26
File Locations	7-26
Location of Temporary Files	7-26
Location of Log Files	7-26
Location of Generated Files	7-26
Generated Objects	7-27
Unloading JCL	7-27
COBOL Transcoding Programs	7-30
Migration of z/OS Files to Oracle Tables	7-30
Reloading Korn Shell Scripts	7-31
Reloading Korn Shell Scripts for Migrating z/OS Files to Oracle Tables	7-31
Creating Oracle DDL Phase	7-32
Transcoding and Loading Phases	7-32

Check Phase	7-32
Target DDL	7-33
Access Functions and Utility Programs	7-34
Access Functions	7-34
Access Function Call Arguments	7-35
Call Arguments Used	7-38
OPEN	7-38
CLOSE	7-38
CLOSE-LOCK	7-38
DELETE	7-38
READ	7-39
REWRITE	7-39
START	7-40
WRITE	7-41
Copy Files to Be Implemented	7-42
Korn Shell Utilities	7-42
Oracle Tuxedo Application Runtime for CICS Configuration Files	7-43
COBOL and JCL Conversion Guide Files	7-43
.rdb Files	7-44
Parameters and Syntax	7-44
Example of .rdb File	7-46
Execution Reports	7-46
Detailed Processing	7-54
Command-Line Syntax	7-54
file.sh	7-54
Unitary Usage Sequence	7-56
Process Steps	7-56
Configuring the Environments and Installing the Components	7-56

Installing the Unloading Components Under z/OS	7-56
Installing the Reloading Components on the Target Platform	7-56
Compiling COBOL Transcoding Programs	7-57
Unloading Data	7-57
Transferring the Data	7-58
Reloading the Data.	7-58
Transcoding and Reloading Command for Tables	7-58
Checking the Transfers	7-58

8. File-to-Db2/luw (udb) Converter

Overview of the File-to-Db2/luw (udb) Converter	8-1
Purpose	8-1
Structure	8-2
See Also	8-2
File Organizations Process	8-2
Migrating to Db2/luw (udb) Table on the Target Platform	8-2
Oracle Tuxedo Application Rehosting Workbench Configuration Name.	8-3
VSAM Files Becoming Db2/luw (udb) Table	8-3
Specific Migration Rules Applied	8-3
Rules Applied to Picture Clauses.	8-3
Environment Variables	8-4
Description of the Input Components	8-4
File Locations	8-4
Location of file.sh	8-4
Location of db-param.cfg File	8-4
Description of the Configuration Files	8-5
db-param.cfg	8-5
Parameters and Syntaxes	8-6

File Modifying Generated Components	8-6
file-template-db2luw.txt	8-7
file-move-assignation-db2luw.pgm	8-9
Datamap File	8-11
Mapper File	8-11
Mapping Strategy Clauses	8-14
Mapping Strategy Clause Syntax and Parameters	8-14
Mapping Strategy Examples.	8-15
Discard Subfield Example.	8-15
Redefines With Default Option Example	8-15
REDEFINES With OPAQUE FIELD Option Example	8-18
REDEFINES With DETAIL TABLE Option Example	8-20
Discrimination Rules	8-23
COBOL Description.	8-26
Description of the Output Files.	8-26
File Locations	8-26
Location of Temporary Files	8-26
Location of Log Files	8-26
Location of Generated Files	8-26
Generated Objects	8-27
Unloading JCL	8-27
COBOL Transcoding Programs	8-30
Migration of z/OS Files to Db2/luw (udb) Tables	8-30
Reloading Korn Shell Scripts.	8-31
Reloading Korn Shell Scripts for Migrating z/OS Files to Db2/luw (udb)	
Tables.	8-31
Creating Oracle DDL Phase	8-32
Transcoding and Loading Phases	8-32

Check Phase	8-32
Target DDL	8-33
Access Functions and Utility Programs	8-34
Access Functions	8-34
Access Function Call Arguments	8-35
Call Arguments Used	8-38
OPEN	8-38
CLOSE	8-38
CLOSE-LOCK	8-38
DELETE	8-38
READ	8-39
REWRITE	8-39
START	8-40
WRITE	8-41
Copy Files to Be Implemented	8-42
Korn Shell Utilities	8-42
Oracle Tuxedo Application Runtime for CICS Configuration Files	8-43
COBOL and JCL Conversion Guide Files	8-43
.rdb Files	8-44
Parameters and Syntax	8-44
Example of .rdb File	8-46
Execution Reports	8-46
Detailed Processing	8-53
Command-Line Syntax	8-54
file.sh	8-54
Unitary Usage Sequence	8-55
Process Steps	8-55
Configuring the Environments and Installing the Components	8-55

Installing the Unloading Components Under z/OS	8-56
Installing the Reloading Components on the Target Platform	8-56
Compiling COBOL Transcoding Programs	8-57
Unloading Data	8-57
Transferring the Data	8-57
Reloading the Data	8-57
Transcoding and Reloading Command for Tables	8-57
Checking the Transfers	8-58

9. JCL Translator

Overview	9-1
JCL Translator Definitions	9-1
General Description and Operation	9-2
General Information	9-2
Behavior Coverage	9-4
Description of Input Components	9-4
Description of the Configuration Files	9-4
The System Description File	9-4
The JCL-Translation Configuration File	9-6
Description of Output Files	9-8
Translated KSH Scripts and Sub-Files	9-8
KSH Version	9-8
File Structure, Naming Scheme and Sub-File Handling	9-8
Handling of JCL and KSH Variables	9-9
Script Structure	9-9
Script Layout	9-9
Execution Logs	9-10
Detailed Operation	9-12

General Information	9-12
Command-line Syntax.	9-13
The Refine Launcher Interface	9-13
The jclz-unix Command	9-13
Repetitive and Incremental Operation	9-14
Initial Processing: Repetitive Operation	9-14
Changes in the Asset: Incremental Operation	9-15
Concurrent Operation.	9-15
Frequently Asked Questions	9-15
When do I translate anew some JCL?.	9-15
How do I force the (re)translation of a JCL?	9-16
I deleted a JCL. Why is the corresponding KSH still present?	9-16
I run the translator but it produces no translation	9-16
The procedures are not included in the JCLs, and hence in the KSH.	9-16
Where do I find the translated procedures?	9-16
Why are some FSNs lost during translation?	9-17

10. COBOL Converter

Overview of the COBOL Converter	10-1
Scope.	10-1
Inputs.	10-2
Outputs	10-2
Conversion Phases	10-2
Restrictions and Limitations.	10-3
Use of COMP-5 Type on Linux Platforms	10-3
Use of COMP-5 Type and the TRUNC Compiler Option	10-4
EBCDIC-to-ASCII Conversion Issues	10-5
Literal Constants: Characters or Numbers?	10-6

Use of Floating-Point Variables.	10-6
REWRITE Operations on LINE SEQUENTIAL Files.	10-8
Pointer Manipulation.	10-8
Pointer Size Changes: Beware of Redefinitions.	10-8
Linkage-Section Arguments with NULL Address.	10-9
Representation of the NULL Pointer Value	10-10
Description of the Input Components, Prerequisites	10-10
Description of the Configuration Files	10-11
System Description File.	10-11
Main Conversion Configuration File.	10-11
General Syntax	10-11
target-dir Clause	10-12
Sql-rules Clause	10-12
keep-same-file-names, target-program-extension and target-copy-extension Clauses	10-12
Verbosity-Level Clause	10-13
deferred-copy-reconcil Clause	10-13
force-translation Clause	10-14
rename-copy-map-file Clause	10-14
rename-call-map-file Clause	10-14
post-translation-file Clause.	10-14
on-size-error-call Clause	10-15
hexa-map-file Clause	10-15
conv-ctrl-file Clause and alt-key-file Clause	10-15
RDBMS-conversion-file Clause.	10-16
keywords-file Clause	10-16
accept-date and accept-day Clauses	10-16
sql-stored-procedures-file Clause.	10-17

remove-sql-qualifier Clause	10-17
activate-cics-rules Clause	10-17
pure-seq-map-file Clause	10-18
dont-print-what-string Clause	10-18
remove-empty-copies Clause	10-19
sql-return-codes-file Clause	10-19
copy-renaming Configuration File	10-19
Call-Renaming Configuration File	10-20
Post-Translation Configuration File	10-21
Hexadecimal Conversion Configuration File	10-22
How to Generate the hexa-map File	10-22
Error Messages	10-23
File-to-RDBMS Configuration Files	10-23
RDBMS-conversion Configuration Files	10-23
keywords File	10-24
stored-procedure File	10-24
purely-sequential Configuration File	10-24
sql-return-codes Configuration File	10-25
Description of Output Files	10-26
Converted Programs and Copy Files	10-26
Naming Scheme	10-26
Transformation Comments	10-26
Modified Code	10-27
Added Code	10-27
Deleted Code	10-28
Moved Code	10-28
Other Comment Rules	10-28
Layout	10-28

Miscellaneous Issues	10-29
Compiler Options	10-29
MicroFocus.	10-29
Mandatory Options	10-29
Installation-dependent Options.	10-34
1.1.1.3 Options Depending on Customer Choice.	10-34
1.1.1.4 Options Influencing Compile-Time Operation	10-36
Mandatory Options	10-37
Installation-dependent options	10-40
Options depending on customer choice	10-41
Options influencing compile-time operation	10-43
COBOL-IT	10-45
Detailed Processing.	10-45
Overview	10-45
Command-Line Syntax	10-47
Refine Launcher Interface	10-47
cobol-convert Command	10-47
Repetitive and Incremental Operation.	10-49
Initial Processing: Repetitive Operation	10-49
Changes in the Asset: Incremental Operation	10-49
Common Information	A-1
COBOL Reloading Programs Reserved Words List	A-1

Introduction

Oracle Tuxedo Application Rehosting Workbench is part of a packaged and comprehensive solution that enables its users:

- To perform a replatforming project with minimum risk and cost;
- To run the replatformed applications in a standardized UNIX/Linux, Oracle Tuxedo, Oracle Database environment.

Oracle Tuxedo Application Rehosting Workbench is used only during the replatforming project itself, whereas Oracle Tuxedo Application Runtime for Batch is used throughout the whole life of the migrated system. Oracle Tuxedo Application Rehosting Workbench is composed of several tools, among which the cataloger, the data-migration tools, the COBOL converter, the JCL translator.

Concepts

The following terms are used to describe the Rehosting Workbench tools, it is important to understand these concepts before using the rest of the documentation. The Oracle Tuxedo Application Rehosting Workbench is used for migrating components and their source files, and also the data files or databases from one platform to another. The migration process and the different platforms are described in more detail in the Oracle Tuxedo Application Runtime Process Guide. The main concepts are clarified below.

Platform

Execution platform or simply platform: a combination of hardware and software components used to execute an application.

Source platform

The platform on which the original software application executes. The hardware platform is an IBM mainframe and the software components include z/OS, IBM COBOL, JCL, DB2 and CICS.

Target platform

The platform on which the final, migrated software application executes. Oracle Tuxedo Application Runtime provides for several different target platforms but they are all based on Unix/Linux and include Tuxedo, the Oracle DBMS and, of course, the Oracle Tuxedo Application Runtime. Depending on the project methodology and organization, the target platform may be subdivided into the test platform and the production platform.

Migration platform

The platform on which the migration tools (the Rehosting Workbench) execute, including the cataloger. This platform is based on Linux running on an Intel-compatible hardware platform.

Source file

A file containing all or part of the source text of a component. There are two kinds of source files:

Main source file

The source file containing all of the source text of a component, or the "top-level" file submitted to the compiler or launcher, possibly containing directives to include sub-files. Examples include COBOL program files, JCL job files, etc.

Included source file or sub-file

A source file containing part of the source text for one or more components, to be included in a main source file. Examples are COBOL copybooks (copy files), JCL PROC files, JCL SYSIN files, etc.

Component

An element of the software system to be migrated or its definition. The Oracle Tuxedo Application Rehosting Workbench cataloger, and the Rehosting Workbench in general, only deals with components defined by source files, such as COBOL programs, SQL tables or JCL jobs. By extension, source files are also considered as components, and hence we distinguish:

Parsable components

Components which can be analyzed in isolation, which have a "meaning" and a role by themselves. These generally correspond to the main source files.

Non-parsable components

Components which have a meaning only when manipulated by other components (e.g. data files) or when included in other components (included source files or sub-files, as defined above).

Abstract Syntax Tree (AST)

The result of parsing (syntactic analysis) and linking (semantic analysis) the source file(s) for a parsable component. This structure captures all the information in the source file and exposes the syntactic and semantic relationships (structure) between the various constructs in this file. This structured form is much more suitable for sophisticated analysis and transformations tools than the initial textual form; this is why it is at the heart of the Rehosting Workbench.

Codeset Conversion

This chapter contains the following topics:

- [Overview of Codeset Conversion](#)
- [z/OS – Displaying Graphical Characters](#)
- [UNIX/Linux: Displaying Graphical Characters](#)
- [Validating and Adapting the Transcoding Copy File](#)
- [Using the COBOL CONVERTMW.cpy File](#)
- [Error Messages](#)

Overview of Codeset Conversion

Purpose

The aim of this document is to describe the configuration required to convert the z/OS EBCDIC CodeSet/CodePage to the UNIX/Linux ASCII CodeSet.

The COBOL copy file `CONVERTMW.cpy` stores the correspondence between source z/OS hexadecimal values and target UNIX/Linux hexadecimal values. The copy file is used by all of the COBOL reloading files generated by the Rehosting Workbench data conversion tools when transcoding z/OS characters to UNIX characters.

Audience

This chapter is intended to be used for migrating z/OS files and DB2 tables to UNIX/Linux files and Oracle tables or Db2/luw (udb). A good understanding of the z/OS platform, the z/OS CodeSet and Codepage as well as the Oracle CharacterSet and UNIX CodePage is required.

z/OS – Displaying Graphical Characters

In order to transcode data assets, you need to determine the character mapping grid in the source z/OS environment.

Reference Monitor

You need to use a monitor that is configured with all of the graphical characteristics linked to the application in order to display all of the characters included in the data used by the z/OS application to be migrated.

The monitor will be used to display the contents of the following file. This step is important because it is the characters displayed by this monitor that enables the completion of the EBCDIC to ASCII conversion phase.

REFCOD80 File

Retrieve the REFCOD80 file from <refinedir>/<release>/convert-data/codeset-tool and transfer it in BINARY format to a z/OS PDS with a RecordLength = 80 parameter. This type of PDS is commonly used to stock JCL and COBOL components.

The REFCOD80 file contains a list of all the characters in the EBCDIC alphabet. Each line in the file has the following format:

```
DEC MVS:<dec>, HEXA MVS:<hex>, CAR=/<car>/
```

Where:

<dec>

Is an EBCDIC decimal value between 000 and 255.

<hex>

Is an EBCDIC hexadecimal value between 00 and FF.

<car>

Is a graphical character as displayed on the reference monitor.

Listing 2-1 REFCOD80 File Example

```
DEC MVS:000, HEXA MVS:00, CAR=/ /  
[...]  
DEC MVS:192, HEXA MVS:C0, CAR={ /  
DEC MVS:193, HEXA MVS:C1, CAR=/A/  
DEC MVS:194, HEXA MVS:C2, CAR=/B/  
DEC MVS:195, HEXA MVS:C3, CAR=/C/  
DEC MVS:196, HEXA MVS:C4, CAR=/D/  
[...]
```

Notes: Depending on your monitor, the graphical characters displayed may be different than those shown in the example;

The transfer in `binary` mode is mandatory because the file is stored in `z/OS` format on the `UNIX/Linux` platform, and the contents should not be altered;

Viewing Characters Under `z/OS`

A `VIEW` under `TSO` is sufficient to look at the file.

The following pictures are screen captures of the complete `REFCOD80` file using a test monitor.

Figure 2-1 REFCOD80 File From Test Monitor: Part 1

```

***** Top of Data **
000001 DEC MVS:000, HEXA MVS:00, DRR=/ 000039 DEC MVS:038, HEXA MVS:26, DRR=/
000002 DEC MVS:001, HEXA MVS:01, DRR=/ 000040 DEC MVS:039, HEXA MVS:27, DRR=/
000003 DEC MVS:002, HEXA MVS:02, DRR=/ 000041 DEC MVS:040, HEXA MVS:28, DRR=/
000004 DEC MVS:003, HEXA MVS:03, DRR=/ 000042 DEC MVS:041, HEXA MVS:29, DRR=/
000005 DEC MVS:004, HEXA MVS:04, DRR=/ 000043 DEC MVS:042, HEXA MVS:2A, DRR=/
000006 DEC MVS:005, HEXA MVS:05, DRR=/ 000044 DEC MVS:043, HEXA MVS:2B, DRR=/
000007 DEC MVS:006, HEXA MVS:06, DRR=/ 000045 DEC MVS:044, HEXA MVS:2C, DRR=/
000008 DEC MVS:007, HEXA MVS:07, DRR=/ 000046 DEC MVS:045, HEXA MVS:2D, DRR=/
000009 DEC MVS:008, HEXA MVS:08, DRR=/ 000047 DEC MVS:046, HEXA MVS:2E, DRR=/
000010 DEC MVS:009, HEXA MVS:09, DRR=/ 000048 DEC MVS:047, HEXA MVS:2F, DRR=/
000011 DEC MVS:010, HEXA MVS:0A, DRR=/ 000049 DEC MVS:048, HEXA MVS:30, DRR=/
000012 DEC MVS:011, HEXA MVS:0B, DRR=/ 000050 DEC MVS:049, HEXA MVS:31, DRR=/
000013 DEC MVS:012, HEXA MVS:0C, DRR=/ 000051 DEC MVS:050, HEXA MVS:32, DRR=/
000014 DEC MVS:013, HEXA MVS:0D, DRR=/ 000052 DEC MVS:051, HEXA MVS:33, DRR=/
000015 DEC MVS:014, HEXA MVS:0E, DRR=/ 000053 DEC MVS:052, HEXA MVS:34, DRR=/
000016 DEC MVS:015, HEXA MVS:0F, DRR=/ 000054 DEC MVS:053, HEXA MVS:35, DRR=/
000017 DEC MVS:016, HEXA MVS:10, DRR=/ 000055 DEC MVS:054, HEXA MVS:36, DRR=/
000018 DEC MVS:017, HEXA MVS:11, DRR=/ 000056 DEC MVS:055, HEXA MVS:37, DRR=/
000019 DEC MVS:018, HEXA MVS:12, DRR=/ 000057 DEC MVS:056, HEXA MVS:38, DRR=/
000020 DEC MVS:019, HEXA MVS:13, DRR=/ 000058 DEC MVS:057, HEXA MVS:39, DRR=/
000021 DEC MVS:020, HEXA MVS:14, DRR=/ 000059 DEC MVS:058, HEXA MVS:3A, DRR=/
000022 DEC MVS:021, HEXA MVS:15, DRR=/ 000060 DEC MVS:059, HEXA MVS:3B, DRR=/
000023 DEC MVS:022, HEXA MVS:16, DRR=/ 000061 DEC MVS:060, HEXA MVS:3C, DRR=/
000024 DEC MVS:023, HEXA MVS:17, DRR=/ 000062 DEC MVS:061, HEXA MVS:3D, DRR=/
000025 DEC MVS:024, HEXA MVS:18, DRR=/ 000063 DEC MVS:062, HEXA MVS:3E, DRR=/
000026 DEC MVS:025, HEXA MVS:19, DRR=/ 000064 DEC MVS:063, HEXA MVS:3F, DRR=/
000027 DEC MVS:026, HEXA MVS:1A, DRR=/ 000065 DEC MVS:064, HEXA MVS:40, DRR=/
000028 DEC MVS:027, HEXA MVS:1B, DRR=/ 000066 DEC MVS:065, HEXA MVS:41, DRR=/
000029 DEC MVS:028, HEXA MVS:1C, DRR=/ 000067 DEC MVS:066, HEXA MVS:42, DRR=/
000030 DEC MVS:029, HEXA MVS:1D, DRR=/ 000068 DEC MVS:067, HEXA MVS:43, DRR=/
000031 DEC MVS:030, HEXA MVS:1E, DRR=/ 000069 DEC MVS:068, HEXA MVS:44, DRR=/
000032 DEC MVS:031, HEXA MVS:1F, DRR=/ 000070 DEC MVS:069, HEXA MVS:45, DRR=/
000033 DEC MVS:032, HEXA MVS:20, DRR=/ 000071 DEC MVS:070, HEXA MVS:46, DRR=/
000034 DEC MVS:033, HEXA MVS:21, DRR=/ 000072 DEC MVS:071, HEXA MVS:47, DRR=/
000035 DEC MVS:034, HEXA MVS:22, DRR=/ 000073 DEC MVS:072, HEXA MVS:48, DRR=/
000036 DEC MVS:035, HEXA MVS:23, DRR=/ 000074 DEC MVS:073, HEXA MVS:49, DRR=/
000037 DEC MVS:036, HEXA MVS:24, DRR=/ 000075 DEC MVS:074, HEXA MVS:4A, DRR=/
000038 DEC MVS:037, HEXA MVS:25, DRR=/ 000076 DEC MVS:075, HEXA MVS:4B, DRR=/
000039 DEC MVS:038, HEXA MVS:26, DRR=/ 000077 DEC MVS:076, HEXA MVS:4C, DRR=/

```

Figure 2-2 REFCOD80 file from test monitor: Part 2

```

000078 DEC MVS:077, HEXA MVS:4D, DRR=/ 000117 DEC MVS:116, HEXA MVS:74, DRR=/
000079 DEC MVS:078, HEXA MVS:4E, DRR=/ 000118 DEC MVS:117, HEXA MVS:75, DRR=/
000080 DEC MVS:079, HEXA MVS:4F, DRR=/ 000119 DEC MVS:118, HEXA MVS:76, DRR=/
000081 DEC MVS:080, HEXA MVS:50, DRR=/ 000120 DEC MVS:119, HEXA MVS:77, DRR=/
000082 DEC MVS:081, HEXA MVS:51, DRR=/ 000121 DEC MVS:120, HEXA MVS:78, DRR=/
000083 DEC MVS:082, HEXA MVS:52, DRR=/ 000122 DEC MVS:121, HEXA MVS:79, DRR=/
000084 DEC MVS:083, HEXA MVS:53, DRR=/ 000123 DEC MVS:122, HEXA MVS:7A, DRR=/
000085 DEC MVS:084, HEXA MVS:54, DRR=/ 000124 DEC MVS:123, HEXA MVS:7B, DRR=/
000086 DEC MVS:085, HEXA MVS:55, DRR=/ 000125 DEC MVS:124, HEXA MVS:7C, DRR=/
000087 DEC MVS:086, HEXA MVS:56, DRR=/ 000126 DEC MVS:125, HEXA MVS:7D, DRR=/
000088 DEC MVS:087, HEXA MVS:57, DRR=/ 000127 DEC MVS:126, HEXA MVS:7E, DRR=/
000089 DEC MVS:088, HEXA MVS:58, DRR=/ 000128 DEC MVS:127, HEXA MVS:7F, DRR=/
000090 DEC MVS:089, HEXA MVS:59, DRR=/ 000129 DEC MVS:128, HEXA MVS:80, DRR=/
000091 DEC MVS:090, HEXA MVS:5A, DRR=/ 000130 DEC MVS:129, HEXA MVS:81, DRR=/
000092 DEC MVS:091, HEXA MVS:5B, DRR=/ 000131 DEC MVS:130, HEXA MVS:82, DRR=/
000093 DEC MVS:092, HEXA MVS:5C, DRR=/ 000132 DEC MVS:131, HEXA MVS:83, DRR=/
000094 DEC MVS:093, HEXA MVS:5D, DRR=/ 000133 DEC MVS:132, HEXA MVS:84, DRR=/
000095 DEC MVS:094, HEXA MVS:5E, DRR=/ 000134 DEC MVS:133, HEXA MVS:85, DRR=/
000096 DEC MVS:095, HEXA MVS:5F, DRR=/ 000135 DEC MVS:134, HEXA MVS:86, DRR=/
000097 DEC MVS:096, HEXA MVS:60, DRR=/ 000136 DEC MVS:135, HEXA MVS:87, DRR=/
000098 DEC MVS:097, HEXA MVS:61, DRR=/ 000137 DEC MVS:136, HEXA MVS:88, DRR=/
000099 DEC MVS:098, HEXA MVS:62, DRR=/ 000138 DEC MVS:137, HEXA MVS:89, DRR=/
000100 DEC MVS:099, HEXA MVS:63, DRR=/ 000139 DEC MVS:138, HEXA MVS:8A, DRR=/
000101 DEC MVS:100, HEXA MVS:64, DRR=/ 000140 DEC MVS:139, HEXA MVS:8B, DRR=/
000102 DEC MVS:101, HEXA MVS:65, DRR=/ 000141 DEC MVS:140, HEXA MVS:8C, DRR=/
000103 DEC MVS:102, HEXA MVS:66, DRR=/ 000142 DEC MVS:141, HEXA MVS:8D, DRR=/
000104 DEC MVS:103, HEXA MVS:67, DRR=/ 000143 DEC MVS:142, HEXA MVS:8E, DRR=/
000105 DEC MVS:104, HEXA MVS:68, DRR=/ 000144 DEC MVS:143, HEXA MVS:8F, DRR=/
000106 DEC MVS:105, HEXA MVS:69, DRR=/ 000145 DEC MVS:144, HEXA MVS:90, DRR=/
000107 DEC MVS:106, HEXA MVS:6A, DRR=/ 000146 DEC MVS:145, HEXA MVS:91, DRR=/
000108 DEC MVS:107, HEXA MVS:6B, DRR=/ 000147 DEC MVS:146, HEXA MVS:92, DRR=/
000109 DEC MVS:108, HEXA MVS:6C, DRR=/ 000148 DEC MVS:147, HEXA MVS:93, DRR=/
000110 DEC MVS:109, HEXA MVS:6D, DRR=/ 000149 DEC MVS:148, HEXA MVS:94, DRR=/
000111 DEC MVS:110, HEXA MVS:6E, DRR=/ 000150 DEC MVS:149, HEXA MVS:95, DRR=/
000112 DEC MVS:111, HEXA MVS:6F, DRR=/ 000151 DEC MVS:150, HEXA MVS:96, DRR=/
000113 DEC MVS:112, HEXA MVS:70, DRR=/ 000152 DEC MVS:151, HEXA MVS:97, DRR=/
000114 DEC MVS:113, HEXA MVS:71, DRR=/ 000153 DEC MVS:152, HEXA MVS:98, DRR=/
000115 DEC MVS:114, HEXA MVS:72, DRR=/ 000154 DEC MVS:153, HEXA MVS:99, DRR=/
000116 DEC MVS:115, HEXA MVS:73, DRR=/ 000155 DEC MVS:154, HEXA MVS:9A, DRR=/

```

Figure 2-3 REFCOD80 file from test monitor: Part 3

000156	DEC MVS:155, HEXA MVS:B6, CAR=/ /	000195	DEC MVS:194, HEXA MVS:C2, CAR=/B/
000157	DEC MVS:156, HEXA MVS:B7, CAR=/ /	000196	DEC MVS:195, HEXA MVS:C3, CAR=/C/
000158	DEC MVS:157, HEXA MVS:B8, CAR=/Z/	000197	DEC MVS:196, HEXA MVS:C4, CAR=/D/
000159	DEC MVS:158, HEXA MVS:B9, CAR=/R/	000198	DEC MVS:197, HEXA MVS:C5, CAR=/E/
000160	DEC MVS:159, HEXA MVS:BA, CAR=/ /	000199	DEC MVS:198, HEXA MVS:C6, CAR=/F/
000161	DEC MVS:160, HEXA MVS:BB, CAR=/U/	000200	DEC MVS:199, HEXA MVS:C7, CAR=/G/
000162	DEC MVS:161, HEXA MVS:BC, CAR=/ /	000201	DEC MVS:200, HEXA MVS:C8, CAR=/H/
000163	DEC MVS:162, HEXA MVS:BD, CAR=/S/	000202	DEC MVS:201, HEXA MVS:C9, CAR=/I/
000164	DEC MVS:163, HEXA MVS:BE, CAR=/ /	000203	DEC MVS:202, HEXA MVS:CA, CAR=/ /
000165	DEC MVS:164, HEXA MVS:BF, CAR=/U/	000204	DEC MVS:203, HEXA MVS:CB, CAR=/S/
000166	DEC MVS:165, HEXA MVS:C0, CAR=/ /	000205	DEC MVS:204, HEXA MVS:CC, CAR=/S/
000167	DEC MVS:166, HEXA MVS:C1, CAR=/U/	000206	DEC MVS:205, HEXA MVS:CD, CAR=/S/
000168	DEC MVS:167, HEXA MVS:C2, CAR=/ /	000207	DEC MVS:206, HEXA MVS:CE, CAR=/S/
000169	DEC MVS:168, HEXA MVS:C3, CAR=/U/	000208	DEC MVS:207, HEXA MVS:CF, CAR=/S/
000170	DEC MVS:169, HEXA MVS:C4, CAR=/Z/	000209	DEC MVS:208, HEXA MVS:DA, CAR=/ /
000171	DEC MVS:170, HEXA MVS:C5, CAR=/ /	000210	DEC MVS:209, HEXA MVS:DB, CAR=/ /
000172	DEC MVS:171, HEXA MVS:C6, CAR=/ /	000211	DEC MVS:210, HEXA MVS:DC, CAR=/N/
000173	DEC MVS:172, HEXA MVS:C7, CAR=/B/	000212	DEC MVS:211, HEXA MVS:DD, CAR=/L/
000174	DEC MVS:173, HEXA MVS:C8, CAR=/ /	000213	DEC MVS:212, HEXA MVS:DE, CAR=/N/
000175	DEC MVS:174, HEXA MVS:C9, CAR=/B/	000214	DEC MVS:213, HEXA MVS:DE, CAR=/N/
000176	DEC MVS:175, HEXA MVS:CA, CAR=/B/	000215	DEC MVS:214, HEXA MVS:DE, CAR=/D/
000177	DEC MVS:176, HEXA MVS:CB, CAR=/ /	000216	DEC MVS:215, HEXA MVS:DE, CAR=/P/
000178	DEC MVS:177, HEXA MVS:CC, CAR=/Z/	000217	DEC MVS:216, HEXA MVS:DE, CAR=/U/
000179	DEC MVS:178, HEXA MVS:CD, CAR=/N/	000218	DEC MVS:217, HEXA MVS:DE, CAR=/R/
000180	DEC MVS:179, HEXA MVS:CE, CAR=/ /	000219	DEC MVS:218, HEXA MVS:DE, CAR=/V/
000181	DEC MVS:180, HEXA MVS:CE, CAR=/B/	000220	DEC MVS:219, HEXA MVS:DE, CAR=/U/
000182	DEC MVS:181, HEXA MVS:CE, CAR=/B/	000221	DEC MVS:220, HEXA MVS:DE, CAR=/U/
000183	DEC MVS:182, HEXA MVS:CE, CAR=/U/	000222	DEC MVS:221, HEXA MVS:DE, CAR=/U/
000184	DEC MVS:183, HEXA MVS:CE, CAR=/B/	000223	DEC MVS:222, HEXA MVS:DE, CAR=/U/
000185	DEC MVS:184, HEXA MVS:CE, CAR=/U/	000224	DEC MVS:223, HEXA MVS:DE, CAR=/U/
000186	DEC MVS:185, HEXA MVS:CE, CAR=/U/	000225	DEC MVS:224, HEXA MVS:DE, CAR=/U/
000187	DEC MVS:186, HEXA MVS:CE, CAR=/ /	000226	DEC MVS:225, HEXA MVS:DE, CAR=/ /
000188	DEC MVS:187, HEXA MVS:CE, CAR=/ /	000227	DEC MVS:226, HEXA MVS:DE, CAR=/S/
000189	DEC MVS:188, HEXA MVS:CE, CAR=/ /	000228	DEC MVS:227, HEXA MVS:DE, CAR=/T/
000190	DEC MVS:189, HEXA MVS:CE, CAR=/S/	000229	DEC MVS:228, HEXA MVS:DE, CAR=/U/
000191	DEC MVS:190, HEXA MVS:CE, CAR=/Z/	000230	DEC MVS:229, HEXA MVS:DE, CAR=/N/
000192	DEC MVS:191, HEXA MVS:CE, CAR=/ /	000231	DEC MVS:230, HEXA MVS:DE, CAR=/U/
000193	DEC MVS:192, HEXA MVS:CE, CAR=/ /	000232	DEC MVS:231, HEXA MVS:DE, CAR=/X/
000194	DEC MVS:193, HEXA MVS:CE, CAR=/R/	000233	DEC MVS:232, HEXA MVS:DE, CAR=/V/

Figure 2-4 REFCOD80 file from test monitor: Part 4

000234	DEC MVS:233, HEXA MVS:E9, CAR=/Z/
000235	DEC MVS:234, HEXA MVS:EA, CAR=/Z/
000236	DEC MVS:235, HEXA MVS:EB, CAR=/O/
000237	DEC MVS:236, HEXA MVS:EC, CAR=/O/
000238	DEC MVS:237, HEXA MVS:ED, CAR=/O/
000239	DEC MVS:238, HEXA MVS:EE, CAR=/O/
000240	DEC MVS:239, HEXA MVS:EF, CAR=/O/
000241	DEC MVS:240, HEXA MVS:FO, CAR=/O/
000242	DEC MVS:241, HEXA MVS:F1, CAR=/1/
000243	DEC MVS:242, HEXA MVS:F2, CAR=/2/
000244	DEC MVS:243, HEXA MVS:F3, CAR=/3/
000245	DEC MVS:244, HEXA MVS:F4, CAR=/4/
000246	DEC MVS:245, HEXA MVS:F5, CAR=/5/
000247	DEC MVS:246, HEXA MVS:F6, CAR=/6/
000248	DEC MVS:247, HEXA MVS:F7, CAR=/7/
000249	DEC MVS:248, HEXA MVS:F8, CAR=/8/
000250	DEC MVS:249, HEXA MVS:F9, CAR=/9/
000251	DEC MVS:250, HEXA MVS:FA, CAR=/3/
000252	DEC MVS:251, HEXA MVS:FB, CAR=/O/
000253	DEC MVS:252, HEXA MVS:FC, CAR=/U/
000254	DEC MVS:253, HEXA MVS:FD, CAR=/U/
000255	DEC MVS:254, HEXA MVS:FE, CAR=/U/
000256	DEC MVS:255, HEXA MVS:FF, CAR=/ /

***** Bottom of Data

Notes: The first screen captures do not show any graphic characters under CAR: / /, because these characters do not have a graphical representation under z/OS.

The test monitor used displays 38 lines per page, other monitors may differ.

UNIX/Linux: Displaying Graphical Characters

Reference Monitor

As under z/OS it is important that the monitor on the target platform as well as all of the graphical environment parameters are equivalent to those configured for the data migration:

- CodePage and Character Set of the database
- Configuration of the LC_* viewed using the “local” command under UNIX.
- etc.

The characters displayed on this screen must be in line with the character set that has been chosen.

COBOL CONVERTMW.cpy

The COBOL CONVERTMW.cpy file is located in <refinedir>/<release>/convert-data/codeset-tool. The copy file is used by reloading scripts of the Rehosting Workbench data tools to convert character strings from EBCDIC to ASCII along with the DB2 table data and the VSAM/SAM files.

Listing 2-2 COBOL CONVERTMW.cpy Copy File Supplied Code

```
*
* TEMPLATE:
*
* Version: <project_name>
* Date:    <date_input>
* Source:  <source_name>
* Source Information:
* <insert_here_any_information>
*
*
01 TRANSCODE-INFO PIC X(70) VALUES "<project_name> <date_input>".
01 TRANSCODE-LENGTH PIC 9(4) VALUE 256.
01 TRANSCODE-SOURCE.
*
*
02 FILLER PIC X(32) VALUE X"000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f".
02 FILLER PIC X(32) VALUE X"202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f".
02 FILLER PIC X(32) VALUE X"404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f".
```

```

02 FILLER PIC X(32) VALUE X"606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f".
02 FILLER PIC X(32) VALUE X"808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f".
02 FILLER PIC X(32) VALUE X"a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbefb".
02 FILLER PIC X(32) VALUE X"c0c1c2c3c4c5c6c7c8c9cacbcccdcecfcd0d1d2d3d4d5d6d7d8d9dadbdcddefff".
02 FILLER PIC X(32) VALUE X"e0e1e2e3e4e5e6e7e8e9eaebeceedeefeff0f1f2f3f4f5f6f7f8f9fafbfcfdfefff".

01 TRANSCODE-CIBLE.
*
02 FILLER PIC X(32) VALUE X"000102039c09867f978d8e0b0c0d0e0f101112139d0a08871819928f1c1d1e1f".
02 FILLER PIC X(32) VALUE X"808182838485171b88898a8b8c050607909116939495960498999a9b14159e1a".
02 FILLER PIC X(32) VALUE X"20a0e2e4e0e1e3e5e7f15b2e3c282b2126e9eaebe8edeefecdf5d242a293b5e".
02 FILLER PIC X(32) VALUE X"2d2fc2c4c0c1c3c5c7d1a62c255f3e3ff8c9cacbc8cdcefc603a2340273d22".
02 FILLER PIC X(32) VALUE X"d8616263646566676869abbbf0fdfeb1b06a6b6c6d6e6f707172aaba6b8c6a4".
02 FILLER PIC X(32) VALUE X"b57e737475767778797a1bfd0dddeaea2a3a5b7a9a7b6bcbdbeac7cafa8b4d7".
02 FILLER PIC X(32) VALUE X"7b414243444546474849adf4f6f2f3f5f7d4a4b4c4d4e4f505152b9fbfcf7fa9f".
02 FILLER PIC X(32) VALUE X"5cf7535455565758595ab2d4d6d2d3d530313233343536373839b3dbdcd9daff".

```

The transcoding copy file is composed of three parts:

- A description and COBOL comment part and the definition of two COBOL variables: TRANSCODE-INFO containing information about the project and TRANSCODE-LENGTH containing a constant that must not be modified.
- The TRANSCODE-SOURCE variable represents the 256 hexadecimal values of the source platform.
- The TRANSCODE-CIBLE COBOL variable contains 256 hexadecimal values corresponding to the ASCII equivalents of the EBCDIC hexadecimal values stored in TRANSCODE-SOURCE.

Note: Each FILLER variable contains 16 hexadecimal character pairs; that is 32 characters coded from 0 to 9 and from A to F.

Viewing characters under UNIX

To view the result of the configuration of the COBOL CONVERTMW.cpy copy file, execute the show_convertmw.sh script stored in

<refinedir>/<release>/convert-data/codeset-tool using the location and name of the copy file as a parameter:

```

<refinedir>/<release>/convert-data/codeset-tool/show_convertmw.sh
<refinedir>/<release>/convert-data/codeset-tool/CONVERTMW.cpy

```

Note: This script requires the initialization of the TMPPROJECT UNIX variable. For example:

```
export TMPPROJECT=$HOME/tmp
```

The script produces a list of all of the characters in the ASCII alphabet sorted in exactly the same manner as the REFCOD80 file viewed under z/OS. Each line displayed has the following format:

```
DEC MVS:<dec>, HEXA MVS:<hex>, DEC UNIX:<decU>, HEXA UNIX:<hexU>,  
CAR=/<carU>/
```

Where:

<dec>

Is an EBCDIC decimal value between 000 and 255.

<hex>

Is an EBCDIC hexadecimal value between 00 and FF.

Is a graphical character as displayed on the reference monitor.

<decU>

Is an ASCII equivalent decimal value between 000 and 255.

<hexU>

Is an ASCII equivalent hexadecimal value between 00 and FF.

<carU>

Is an ASCII graphical character of the UNIX/Linux platform.

Listing 2-3 UNIX Character Example

```
DEC MVS:000, HEXA MVS:00, DEC UNIX:000, HEXA UNIX:00, CAR=//  
[...]  
DEC MVS:192, HEXA MVS:C0, DEC UNIX:123, HEXA UNIX:7b, CAR=/{/  
DEC MVS:193, HEXA MVS:C1, DEC UNIX:065, HEXA UNIX:41, CAR=/A/  
DEC MVS:194, HEXA MVS:C2, DEC UNIX:066, HEXA UNIX:42, CAR=/B/  
DEC MVS:195, HEXA MVS:C3, DEC UNIX:067, HEXA UNIX:43, CAR=/C/  
DEC MVS:196, HEXA MVS:C4, DEC UNIX:068, HEXA UNIX:44, CAR=/D/  
[...]
```

Notes: Some characters displayed under UNIX/Linux can provoke display problems. On a z/OS platform, these characters are probably not represented graphically by the TSO VIEWER.

If you view the results on an XTERM configured in 7-bit mode, you will only see some of the graphical characters.

Listing 2-4 Complete UNIX Character List

```
DEC MVS:000, HEXA MVS:00, DEC UNIX:000, HEXA UNIX:00, CAR=/ /
DEC MVS:001, HEXA MVS:01, DEC UNIX:001, HEXA UNIX:01, CAR=/ /
DEC MVS:002, HEXA MVS:02, DEC UNIX:002, HEXA UNIX:02, CAR=/ /
DEC MVS:003, HEXA MVS:03, DEC UNIX:003, HEXA UNIX:03, CAR=/ /
DEC MVS:004, HEXA MVS:04, DEC UNIX:156, HEXA UNIX:9c, CAR=/œ/
DEC MVS:005, HEXA MVS:05, DEC UNIX:009, HEXA UNIX:09, CAR=/ /
DEC MVS:006, HEXA MVS:06, DEC UNIX:134, HEXA UNIX:86, CAR=/†/
DEC MVS:007, HEXA MVS:07, DEC UNIX:127, HEXA UNIX:7f, CAR=? /
DEC MVS:008, HEXA MVS:08, DEC UNIX:151, HEXA UNIX:97, CAR=- /
DEC MVS:009, HEXA MVS:09, DEC UNIX:141, HEXA UNIX:8d, CAR=? /
DEC MVS:010, HEXA MVS:0A, DEC UNIX:142, HEXA UNIX:8e, CAR=Ž /
DEC MVS:011, HEXA MVS:0B, DEC UNIX:011, HEXA UNIX:0b, CAR=/ /
DEC MVS:012, HEXA MVS:0C, DEC UNIX:012, HEXA UNIX:0c, CAR=/ /
DEC MVS:013, HEXA MVS:0D, DEC UNIX:013, HEXA UNIX:0d, CAR=/ /
DEC MVS:014, HEXA MVS:0E, DEC UNIX:014, HEXA UNIX:0e, CAR=/ /
DEC MVS:015, HEXA MVS:0F, DEC UNIX:015, HEXA UNIX:0f, CAR=/ /
DEC MVS:016, HEXA MVS:10, DEC UNIX:016, HEXA UNIX:10, CAR=/ /
DEC MVS:017, HEXA MVS:11, DEC UNIX:017, HEXA UNIX:11, CAR=/ /
DEC MVS:018, HEXA MVS:12, DEC UNIX:018, HEXA UNIX:12, CAR=/ /
DEC MVS:019, HEXA MVS:13, DEC UNIX:019, HEXA UNIX:13, CAR=/ /
DEC MVS:020, HEXA MVS:14, DEC UNIX:157, HEXA UNIX:9d, CAR=? /
DEC MVS:021, HEXA MVS:15, DEC UNIX:010, HEXA UNIX:0a, CAR=/ /
DEC MVS:022, HEXA MVS:16, DEC UNIX:008, HEXA UNIX:08, CAR=/ /
```

```

DEC MVS:023, HEXA MVS:17, DEC UNIX:135, HEXA UNIX:87, CAR= / ‡ /
DEC MVS:024, HEXA MVS:18, DEC UNIX:024, HEXA UNIX:18, CAR= / /
DEC MVS:025, HEXA MVS:19, DEC UNIX:025, HEXA UNIX:19, CAR= / /
DEC MVS:026, HEXA MVS:1A, DEC UNIX:146, HEXA UNIX:92, CAR= / ' /
DEC MVS:027, HEXA MVS:1B, DEC UNIX:143, HEXA UNIX:8f, CAR= / ? /
DEC MVS:028, HEXA MVS:1C, DEC UNIX:028, HEXA UNIX:1c, CAR= / /
DEC MVS:029, HEXA MVS:1D, DEC UNIX:029, HEXA UNIX:1d, CAR= / /
DEC MVS:030, HEXA MVS:1E, DEC UNIX:030, HEXA UNIX:1e, CAR= / /
DEC MVS:031, HEXA MVS:1F, DEC UNIX:031, HEXA UNIX:1f, CAR= / /
DEC MVS:032, HEXA MVS:20, DEC UNIX:128, HEXA UNIX:80, CAR= / € /
DEC MVS:033, HEXA MVS:21, DEC UNIX:129, HEXA UNIX:81, CAR= / ? /
DEC MVS:034, HEXA MVS:22, DEC UNIX:130, HEXA UNIX:82, CAR= / , /
DEC MVS:035, HEXA MVS:23, DEC UNIX:131, HEXA UNIX:83, CAR= / f /
DEC MVS:036, HEXA MVS:24, DEC UNIX:132, HEXA UNIX:84, CAR= / " /
DEC MVS:037, HEXA MVS:25, DEC UNIX:133, HEXA UNIX:85, CAR= / ... /
DEC MVS:038, HEXA MVS:26, DEC UNIX:023, HEXA UNIX:17, CAR= / /
DEC MVS:039, HEXA MVS:27, DEC UNIX:027, HEXA UNIX:1b, CAR= / /
DEC MVS:040, HEXA MVS:28, DEC UNIX:136, HEXA UNIX:88, CAR= / ^ /
DEC MVS:041, HEXA MVS:29, DEC UNIX:137, HEXA UNIX:89, CAR= / % /
DEC MVS:042, HEXA MVS:2A, DEC UNIX:138, HEXA UNIX:8a, CAR= / Š /
DEC MVS:043, HEXA MVS:2B, DEC UNIX:139, HEXA UNIX:8b, CAR= / < /
DEC MVS:044, HEXA MVS:2C, DEC UNIX:140, HEXA UNIX:8c, CAR= / Œ /
DEC MVS:045, HEXA MVS:2D, DEC UNIX:005, HEXA UNIX:05, CAR= / /
DEC MVS:046, HEXA MVS:2E, DEC UNIX:006, HEXA UNIX:06, CAR= / /
DEC MVS:047, HEXA MVS:2F, DEC UNIX:007, HEXA UNIX:07, CAR= / /
DEC MVS:048, HEXA MVS:30, DEC UNIX:144, HEXA UNIX:90, CAR= / ? /
DEC MVS:049, HEXA MVS:31, DEC UNIX:145, HEXA UNIX:91, CAR= / ' /

```

DEC MVS:050, HEXA MVS:32, DEC UNIX:022, HEXA UNIX:16, CAR=/ /
 DEC MVS:051, HEXA MVS:33, DEC UNIX:147, HEXA UNIX:93, CAR="/ /
 DEC MVS:052, HEXA MVS:34, DEC UNIX:148, HEXA UNIX:94, CAR="/ /
 DEC MVS:053, HEXA MVS:35, DEC UNIX:149, HEXA UNIX:95, CAR=/o/
 DEC MVS:054, HEXA MVS:36, DEC UNIX:150, HEXA UNIX:96, CAR=- /
 DEC MVS:055, HEXA MVS:37, DEC UNIX:004, HEXA UNIX:04, CAR=/ /
 DEC MVS:056, HEXA MVS:38, DEC UNIX:152, HEXA UNIX:98, CAR=/~/
 DEC MVS:057, HEXA MVS:39, DEC UNIX:153, HEXA UNIX:99, CAR=/™/
 DEC MVS:058, HEXA MVS:3A, DEC UNIX:154, HEXA UNIX:9a, CAR=/š/
 DEC MVS:059, HEXA MVS:3B, DEC UNIX:155, HEXA UNIX:9b, CAR=/> /
 DEC MVS:060, HEXA MVS:3C, DEC UNIX:020, HEXA UNIX:14, CAR=/ /
 DEC MVS:061, HEXA MVS:3D, DEC UNIX:021, HEXA UNIX:15, CAR=/ /
 DEC MVS:062, HEXA MVS:3E, DEC UNIX:158, HEXA UNIX:9e, CAR=/ž/
 DEC MVS:063, HEXA MVS:3F, DEC UNIX:026, HEXA UNIX:1a, CAR=/ /
 DEC MVS:064, HEXA MVS:40, DEC UNIX:032, HEXA UNIX:20, CAR=/ /
 DEC MVS:065, HEXA MVS:41, DEC UNIX:160, HEXA UNIX:a0, CAR=/ /
 DEC MVS:066, HEXA MVS:42, DEC UNIX:226, HEXA UNIX:e2, CAR=/â/
 DEC MVS:067, HEXA MVS:43, DEC UNIX:228, HEXA UNIX:e4, CAR=/ä/
 DEC MVS:068, HEXA MVS:44, DEC UNIX:224, HEXA UNIX:e0, CAR=/à/
 DEC MVS:069, HEXA MVS:45, DEC UNIX:225, HEXA UNIX:e1, CAR=/á/
 DEC MVS:070, HEXA MVS:46, DEC UNIX:227, HEXA UNIX:e3, CAR=/ã/
 DEC MVS:071, HEXA MVS:47, DEC UNIX:229, HEXA UNIX:e5, CAR=/å/
 DEC MVS:072, HEXA MVS:48, DEC UNIX:231, HEXA UNIX:e7, CAR=/ç/
 DEC MVS:073, HEXA MVS:49, DEC UNIX:241, HEXA UNIX:f1, CAR=/ñ/
 DEC MVS:074, HEXA MVS:4A, DEC UNIX:091, HEXA UNIX:5b, CAR=/[/
 DEC MVS:075, HEXA MVS:4B, DEC UNIX:046, HEXA UNIX:2e, CAR=/./
 DEC MVS:076, HEXA MVS:4C, DEC UNIX:060, HEXA UNIX:3c, CAR=/</

DEC MVS:077, HEXA MVS:4D, DEC UNIX:040, HEXA UNIX:28, CAR=/(/
 DEC MVS:078, HEXA MVS:4E, DEC UNIX:043, HEXA UNIX:2b, CAR=/(+ /
 DEC MVS:079, HEXA MVS:4F, DEC UNIX:033, HEXA UNIX:21, CAR=/(! /
 DEC MVS:080, HEXA MVS:50, DEC UNIX:038, HEXA UNIX:26, CAR=/(& /
 DEC MVS:081, HEXA MVS:51, DEC UNIX:233, HEXA UNIX:e9, CAR=/(é /
 DEC MVS:082, HEXA MVS:52, DEC UNIX:234, HEXA UNIX:ea, CAR=/(ê /
 DEC MVS:083, HEXA MVS:53, DEC UNIX:235, HEXA UNIX:eb, CAR=/(ë /
 DEC MVS:084, HEXA MVS:54, DEC UNIX:232, HEXA UNIX:e8, CAR=/(è /
 DEC MVS:085, HEXA MVS:55, DEC UNIX:237, HEXA UNIX:ed, CAR=/(í /
 DEC MVS:086, HEXA MVS:56, DEC UNIX:238, HEXA UNIX:ee, CAR=/(î /
 DEC MVS:087, HEXA MVS:57, DEC UNIX:239, HEXA UNIX:ef, CAR=/(ï /
 DEC MVS:088, HEXA MVS:58, DEC UNIX:236, HEXA UNIX:ec, CAR=/(ì /
 DEC MVS:089, HEXA MVS:59, DEC UNIX:223, HEXA UNIX:df, CAR=/(ß /
 DEC MVS:090, HEXA MVS:5A, DEC UNIX:093, HEXA UNIX:5d, CAR=/(] /
 DEC MVS:091, HEXA MVS:5B, DEC UNIX:036, HEXA UNIX:24, CAR=/(\$ /
 DEC MVS:092, HEXA MVS:5C, DEC UNIX:042, HEXA UNIX:2a, CAR=/(* /
 DEC MVS:093, HEXA MVS:5D, DEC UNIX:041, HEXA UNIX:29, CAR=/() /
 DEC MVS:094, HEXA MVS:5E, DEC UNIX:059, HEXA UNIX:3b, CAR=/(; /
 DEC MVS:095, HEXA MVS:5F, DEC UNIX:094, HEXA UNIX:5e, CAR=/(^ /
 DEC MVS:096, HEXA MVS:60, DEC UNIX:045, HEXA UNIX:2d, CAR=/(- /
 DEC MVS:097, HEXA MVS:61, DEC UNIX:047, HEXA UNIX:2f, CAR=/(/ /
 DEC MVS:098, HEXA MVS:62, DEC UNIX:194, HEXA UNIX:c2, CAR=/(Â /
 DEC MVS:099, HEXA MVS:63, DEC UNIX:196, HEXA UNIX:c4, CAR=/(Ä /
 DEC MVS:100, HEXA MVS:64, DEC UNIX:192, HEXA UNIX:c0, CAR=/(À /
 DEC MVS:101, HEXA MVS:65, DEC UNIX:193, HEXA UNIX:c1, CAR=/(Á /
 DEC MVS:102, HEXA MVS:66, DEC UNIX:195, HEXA UNIX:c3, CAR=/(Ã /
 DEC MVS:103, HEXA MVS:67, DEC UNIX:197, HEXA UNIX:c5, CAR=/(Å /

DEC MVS:104, HEXA MVS:68, DEC UNIX:199, HEXA UNIX:c7, CAR=/Ç/
 DEC MVS:105, HEXA MVS:69, DEC UNIX:209, HEXA UNIX:d1, CAR=/Ñ/
 DEC MVS:106, HEXA MVS:6A, DEC UNIX:166, HEXA UNIX:a6, CAR=/|/
 DEC MVS:107, HEXA MVS:6B, DEC UNIX:044, HEXA UNIX:2c, CAR=/,//
 DEC MVS:108, HEXA MVS:6C, DEC UNIX:037, HEXA UNIX:25, CAR=/%//
 DEC MVS:109, HEXA MVS:6D, DEC UNIX:095, HEXA UNIX:5f, CAR=/_/
 DEC MVS:110, HEXA MVS:6E, DEC UNIX:062, HEXA UNIX:3e, CAR=/>/
 DEC MVS:111, HEXA MVS:6F, DEC UNIX:063, HEXA UNIX:3f, CAR=/?/
 DEC MVS:112, HEXA MVS:70, DEC UNIX:248, HEXA UNIX:f8, CAR=/ø/
 DEC MVS:113, HEXA MVS:71, DEC UNIX:201, HEXA UNIX:c9, CAR=/É/
 DEC MVS:114, HEXA MVS:72, DEC UNIX:202, HEXA UNIX:ca, CAR=/Ê/
 DEC MVS:115, HEXA MVS:73, DEC UNIX:203, HEXA UNIX:cb, CAR=/Ë/
 DEC MVS:116, HEXA MVS:74, DEC UNIX:200, HEXA UNIX:c8, CAR=/È/
 DEC MVS:117, HEXA MVS:75, DEC UNIX:205, HEXA UNIX:cd, CAR=/Í/
 DEC MVS:118, HEXA MVS:76, DEC UNIX:206, HEXA UNIX:ce, CAR=/Î/
 DEC MVS:119, HEXA MVS:77, DEC UNIX:207, HEXA UNIX:cf, CAR=/Ï/
 DEC MVS:120, HEXA MVS:78, DEC UNIX:204, HEXA UNIX:cc, CAR=/Ì/
 DEC MVS:121, HEXA MVS:79, DEC UNIX:096, HEXA UNIX:60, CAR=/`/
 DEC MVS:122, HEXA MVS:7A, DEC UNIX:058, HEXA UNIX:3a, CAR=/:/
 DEC MVS:123, HEXA MVS:7B, DEC UNIX:035, HEXA UNIX:23, CAR=/#/
 DEC MVS:124, HEXA MVS:7C, DEC UNIX:064, HEXA UNIX:40, CAR=/@/
 DEC MVS:125, HEXA MVS:7D, DEC UNIX:039, HEXA UNIX:27, CAR=/'/
 DEC MVS:126, HEXA MVS:7E, DEC UNIX:061, HEXA UNIX:3d, CAR=/=/
 DEC MVS:127, HEXA MVS:7F, DEC UNIX:034, HEXA UNIX:22, CAR=/"/
 DEC MVS:128, HEXA MVS:80, DEC UNIX:216, HEXA UNIX:d8, CAR=/Ø/
 DEC MVS:129, HEXA MVS:81, DEC UNIX:097, HEXA UNIX:61, CAR=/a/
 DEC MVS:130, HEXA MVS:82, DEC UNIX:098, HEXA UNIX:62, CAR=/b/

DEC MVS:131, HEXA MVS:83, DEC UNIX:099, HEXA UNIX:63, CAR=/c/
 DEC MVS:132, HEXA MVS:84, DEC UNIX:100, HEXA UNIX:64, CAR=/d/
 DEC MVS:133, HEXA MVS:85, DEC UNIX:101, HEXA UNIX:65, CAR=/e/
 DEC MVS:134, HEXA MVS:86, DEC UNIX:102, HEXA UNIX:66, CAR=/f/
 DEC MVS:135, HEXA MVS:87, DEC UNIX:103, HEXA UNIX:67, CAR=/g/
 DEC MVS:136, HEXA MVS:88, DEC UNIX:104, HEXA UNIX:68, CAR=/h/
 DEC MVS:137, HEXA MVS:89, DEC UNIX:105, HEXA UNIX:69, CAR=/i/
 DEC MVS:138, HEXA MVS:8A, DEC UNIX:171, HEXA UNIX:ab, CAR="//
 DEC MVS:139, HEXA MVS:8B, DEC UNIX:187, HEXA UNIX:bb, CAR="//
 DEC MVS:140, HEXA MVS:8C, DEC UNIX:240, HEXA UNIX:f0, CAR=/ð/
 DEC MVS:141, HEXA MVS:8D, DEC UNIX:253, HEXA UNIX:fd, CAR=/ŷ/
 DEC MVS:142, HEXA MVS:8E, DEC UNIX:254, HEXA UNIX:fe, CAR=/þ/
 DEC MVS:143, HEXA MVS:8F, DEC UNIX:177, HEXA UNIX:b1, CAR=/±/
 DEC MVS:144, HEXA MVS:90, DEC UNIX:176, HEXA UNIX:b0, CAR=/°/
 DEC MVS:145, HEXA MVS:91, DEC UNIX:106, HEXA UNIX:6a, CAR=/j/
 DEC MVS:146, HEXA MVS:92, DEC UNIX:107, HEXA UNIX:6b, CAR=/k/
 DEC MVS:147, HEXA MVS:93, DEC UNIX:108, HEXA UNIX:6c, CAR=/l/
 DEC MVS:148, HEXA MVS:94, DEC UNIX:109, HEXA UNIX:6d, CAR=/m/
 DEC MVS:149, HEXA MVS:95, DEC UNIX:110, HEXA UNIX:6e, CAR=/n/
 DEC MVS:150, HEXA MVS:96, DEC UNIX:111, HEXA UNIX:6f, CAR=/o/
 DEC MVS:151, HEXA MVS:97, DEC UNIX:112, HEXA UNIX:70, CAR=/p/
 DEC MVS:152, HEXA MVS:98, DEC UNIX:113, HEXA UNIX:71, CAR=/q/
 DEC MVS:153, HEXA MVS:99, DEC UNIX:114, HEXA UNIX:72, CAR=/r/
 DEC MVS:154, HEXA MVS:9A, DEC UNIX:170, HEXA UNIX:aa, CAR=/^/
 DEC MVS:155, HEXA MVS:9B, DEC UNIX:186, HEXA UNIX:ba, CAR=/°/
 DEC MVS:156, HEXA MVS:9C, DEC UNIX:230, HEXA UNIX:e6, CAR=/æ/
 DEC MVS:157, HEXA MVS:9D, DEC UNIX:184, HEXA UNIX:b8, CAR=/,/

DEC MVS:158, HEXA MVS:9E, DEC UNIX:198, HEXA UNIX:c6, CAR=/Æ/
 DEC MVS:159, HEXA MVS:9F, DEC UNIX:164, HEXA UNIX:a4, CAR=/ǻ/
 DEC MVS:160, HEXA MVS:A0, DEC UNIX:181, HEXA UNIX:b5, CAR=/µ/
 DEC MVS:161, HEXA MVS:A1, DEC UNIX:126, HEXA UNIX:7e, CAR=/~/
 DEC MVS:162, HEXA MVS:A2, DEC UNIX:115, HEXA UNIX:73, CAR=/s/
 DEC MVS:163, HEXA MVS:A3, DEC UNIX:116, HEXA UNIX:74, CAR=/t/
 DEC MVS:164, HEXA MVS:A4, DEC UNIX:117, HEXA UNIX:75, CAR=/u/
 DEC MVS:165, HEXA MVS:A5, DEC UNIX:118, HEXA UNIX:76, CAR=/v/
 DEC MVS:166, HEXA MVS:A6, DEC UNIX:119, HEXA UNIX:77, CAR=/w/
 DEC MVS:167, HEXA MVS:A7, DEC UNIX:120, HEXA UNIX:78, CAR=/x/
 DEC MVS:168, HEXA MVS:A8, DEC UNIX:121, HEXA UNIX:79, CAR=/y/
 DEC MVS:169, HEXA MVS:A9, DEC UNIX:122, HEXA UNIX:7a, CAR=/z/
 DEC MVS:170, HEXA MVS:AA, DEC UNIX:161, HEXA UNIX:a1, CAR=/ı/
 DEC MVS:171, HEXA MVS:AB, DEC UNIX:191, HEXA UNIX:bf, CAR=/¿/
 DEC MVS:172, HEXA MVS:AC, DEC UNIX:208, HEXA UNIX:d0, CAR=/Ð/
 DEC MVS:173, HEXA MVS:AD, DEC UNIX:221, HEXA UNIX:dd, CAR=/Ÿ/
 DEC MVS:174, HEXA MVS:AE, DEC UNIX:222, HEXA UNIX:de, CAR=/Ě/
 DEC MVS:175, HEXA MVS:AF, DEC UNIX:174, HEXA UNIX:ae, CAR=/®/
 DEC MVS:176, HEXA MVS:B0, DEC UNIX:162, HEXA UNIX:a2, CAR=/¢/
 DEC MVS:177, HEXA MVS:B1, DEC UNIX:163, HEXA UNIX:a3, CAR=/£/
 DEC MVS:178, HEXA MVS:B2, DEC UNIX:165, HEXA UNIX:a5, CAR=/¥/
 DEC MVS:179, HEXA MVS:B3, DEC UNIX:183, HEXA UNIX:b7, CAR=/·/
 DEC MVS:180, HEXA MVS:B4, DEC UNIX:169, HEXA UNIX:a9, CAR=/©/
 DEC MVS:181, HEXA MVS:B5, DEC UNIX:167, HEXA UNIX:a7, CAR=/§/
 DEC MVS:182, HEXA MVS:B6, DEC UNIX:182, HEXA UNIX:b6, CAR=//
 DEC MVS:183, HEXA MVS:B7, DEC UNIX:188, HEXA UNIX:bc, CAR=/¼/
 DEC MVS:184, HEXA MVS:B8, DEC UNIX:189, HEXA UNIX:bd, CAR=/½/

DEC MVS:185, HEXA MVS:B9, DEC UNIX:190, HEXA UNIX:be, CAR=/¼/
 DEC MVS:186, HEXA MVS:BA, DEC UNIX:172, HEXA UNIX:ac, CAR=/ /
 DEC MVS:187, HEXA MVS:BB, DEC UNIX:124, HEXA UNIX:7c, CAR=/ | /
 DEC MVS:188, HEXA MVS:BC, DEC UNIX:175, HEXA UNIX:af, CAR=/ ^ /
 DEC MVS:189, HEXA MVS:BD, DEC UNIX:168, HEXA UNIX:a8, CAR=/ `` /
 DEC MVS:190, HEXA MVS:BE, DEC UNIX:180, HEXA UNIX:b4, CAR=/ ` /
 DEC MVS:191, HEXA MVS:BF, DEC UNIX:215, HEXA UNIX:d7, CAR=/ × /
 DEC MVS:192, HEXA MVS:C0, DEC UNIX:123, HEXA UNIX:7b, CAR=/ { /
 DEC MVS:193, HEXA MVS:C1, DEC UNIX:065, HEXA UNIX:41, CAR=/ A /
 DEC MVS:194, HEXA MVS:C2, DEC UNIX:066, HEXA UNIX:42, CAR=/ B /
 DEC MVS:195, HEXA MVS:C3, DEC UNIX:067, HEXA UNIX:43, CAR=/ C /
 DEC MVS:196, HEXA MVS:C4, DEC UNIX:068, HEXA UNIX:44, CAR=/ D /
 DEC MVS:197, HEXA MVS:C5, DEC UNIX:069, HEXA UNIX:45, CAR=/ E /
 DEC MVS:198, HEXA MVS:C6, DEC UNIX:070, HEXA UNIX:46, CAR=/ F /
 DEC MVS:199, HEXA MVS:C7, DEC UNIX:071, HEXA UNIX:47, CAR=/ G /
 DEC MVS:200, HEXA MVS:C8, DEC UNIX:072, HEXA UNIX:48, CAR=/ H /
 DEC MVS:201, HEXA MVS:C9, DEC UNIX:073, HEXA UNIX:49, CAR=/ I /
 DEC MVS:202, HEXA MVS:CA, DEC UNIX:173, HEXA UNIX:ad, CAR=/ - /
 DEC MVS:203, HEXA MVS:CB, DEC UNIX:244, HEXA UNIX:f4, CAR=/ ô /
 DEC MVS:204, HEXA MVS:CC, DEC UNIX:246, HEXA UNIX:f6, CAR=/ ö /
 DEC MVS:205, HEXA MVS:CD, DEC UNIX:242, HEXA UNIX:f2, CAR=/ ò /
 DEC MVS:206, HEXA MVS:CE, DEC UNIX:243, HEXA UNIX:f3, CAR=/ ó /
 DEC MVS:207, HEXA MVS:CF, DEC UNIX:245, HEXA UNIX:f5, CAR=/ õ /
 DEC MVS:208, HEXA MVS:D0, DEC UNIX:125, HEXA UNIX:7d, CAR=/ } /
 DEC MVS:209, HEXA MVS:D1, DEC UNIX:074, HEXA UNIX:4a, CAR=/ J /
 DEC MVS:210, HEXA MVS:D2, DEC UNIX:075, HEXA UNIX:4b, CAR=/ K /
 DEC MVS:211, HEXA MVS:D3, DEC UNIX:076, HEXA UNIX:4c, CAR=/ L /

DEC MVS:212, HEXA MVS:D4, DEC UNIX:077, HEXA UNIX:4d, CAR=/M/
 DEC MVS:213, HEXA MVS:D5, DEC UNIX:078, HEXA UNIX:4e, CAR=/N/
 DEC MVS:214, HEXA MVS:D6, DEC UNIX:079, HEXA UNIX:4f, CAR=/O/
 DEC MVS:215, HEXA MVS:D7, DEC UNIX:080, HEXA UNIX:50, CAR=/P/
 DEC MVS:216, HEXA MVS:D8, DEC UNIX:081, HEXA UNIX:51, CAR=/Q/
 DEC MVS:217, HEXA MVS:D9, DEC UNIX:082, HEXA UNIX:52, CAR=/R/
 DEC MVS:218, HEXA MVS:DA, DEC UNIX:185, HEXA UNIX:b9, CAR=/¹/
 DEC MVS:219, HEXA MVS:DB, DEC UNIX:251, HEXA UNIX:fb, CAR=/û/
 DEC MVS:220, HEXA MVS:DC, DEC UNIX:252, HEXA UNIX:fc, CAR=/ü/
 DEC MVS:221, HEXA MVS:DD, DEC UNIX:249, HEXA UNIX:f9, CAR=/ù/
 DEC MVS:222, HEXA MVS:DE, DEC UNIX:250, HEXA UNIX:fa, CAR=/ú/
 DEC MVS:223, HEXA MVS:DF, DEC UNIX:159, HEXA UNIX:9f, CAR=/ÿ/
 DEC MVS:224, HEXA MVS:E0, DEC UNIX:092, HEXA UNIX:5c, CAR=/\/
 DEC MVS:225, HEXA MVS:E1, DEC UNIX:247, HEXA UNIX:f7, CAR=/÷/
 DEC MVS:226, HEXA MVS:E2, DEC UNIX:083, HEXA UNIX:53, CAR=/S/
 DEC MVS:227, HEXA MVS:E3, DEC UNIX:084, HEXA UNIX:54, CAR=/T/
 DEC MVS:228, HEXA MVS:E4, DEC UNIX:085, HEXA UNIX:55, CAR=/U/
 DEC MVS:229, HEXA MVS:E5, DEC UNIX:086, HEXA UNIX:56, CAR=/V/
 DEC MVS:230, HEXA MVS:E6, DEC UNIX:087, HEXA UNIX:57, CAR=/W/
 DEC MVS:231, HEXA MVS:E7, DEC UNIX:088, HEXA UNIX:58, CAR=/X/
 DEC MVS:232, HEXA MVS:E8, DEC UNIX:089, HEXA UNIX:59, CAR=/Y/
 DEC MVS:233, HEXA MVS:E9, DEC UNIX:090, HEXA UNIX:5a, CAR=/Z/
 DEC MVS:234, HEXA MVS:EA, DEC UNIX:178, HEXA UNIX:b2, CAR=/²/
 DEC MVS:235, HEXA MVS:EB, DEC UNIX:212, HEXA UNIX:d4, CAR=/Ô/
 DEC MVS:236, HEXA MVS:EC, DEC UNIX:214, HEXA UNIX:d6, CAR=/Ö/
 DEC MVS:237, HEXA MVS:ED, DEC UNIX:210, HEXA UNIX:d2, CAR=/Ò/
 DEC MVS:238, HEXA MVS:EE, DEC UNIX:211, HEXA UNIX:d3, CAR=/Ó/

```

DEC MVS:239, HEXA MVS:EF, DEC UNIX:213, HEXA UNIX:d5, CAR=/Ö/
DEC MVS:240, HEXA MVS:F0, DEC UNIX:048, HEXA UNIX:30, CAR=/0/
DEC MVS:241, HEXA MVS:F1, DEC UNIX:049, HEXA UNIX:31, CAR=/1/
DEC MVS:242, HEXA MVS:F2, DEC UNIX:050, HEXA UNIX:32, CAR=/2/
DEC MVS:243, HEXA MVS:F3, DEC UNIX:051, HEXA UNIX:33, CAR=/3/
DEC MVS:244, HEXA MVS:F4, DEC UNIX:052, HEXA UNIX:34, CAR=/4/
DEC MVS:245, HEXA MVS:F5, DEC UNIX:053, HEXA UNIX:35, CAR=/5/
DEC MVS:246, HEXA MVS:F6, DEC UNIX:054, HEXA UNIX:36, CAR=/6/
DEC MVS:247, HEXA MVS:F7, DEC UNIX:055, HEXA UNIX:37, CAR=/7/
DEC MVS:248, HEXA MVS:F8, DEC UNIX:056, HEXA UNIX:38, CAR=/8/
DEC MVS:249, HEXA MVS:F9, DEC UNIX:057, HEXA UNIX:39, CAR=/9/
DEC MVS:250, HEXA MVS:FA, DEC UNIX:179, HEXA UNIX:b3, CAR=/³/
DEC MVS:251, HEXA MVS:FB, DEC UNIX:219, HEXA UNIX:db, CAR=/Û/
DEC MVS:252, HEXA MVS:FC, DEC UNIX:220, HEXA UNIX:dc, CAR=/Ü/
DEC MVS:253, HEXA MVS:FD, DEC UNIX:217, HEXA UNIX:d9, CAR=/Ù/
DEC MVS:254, HEXA MVS:FE, DEC UNIX:218, HEXA UNIX:da, CAR=/Ú/
DEC MVS:255, HEXA MVS:FF, DEC UNIX:255, HEXA UNIX:ff, CAR=/ÿ/

```

Validating and Adapting the Transcoding Copy File

Validation

To validate the transcodage of EBCDIC characters to ASCII, compare the lines displayed on the z/OS monitor with the lines displayed on the UNIX/Linux monitor.

Listing 2-5 Examples of Different Displays Under ZOS and UNIX:

```

ZOSDEC MVS:192, HEXA MVS:C0, CAR=/{/
UNIXDEC MVS:192, HEXA MVS:C0, DEC UNIX:123, HEXA UNIX:7b, CAR=/{/

```

```
ZOSDEC MVS:193, HEXA MVS:C1, CAR=/A/  
UNIXDEC MVS:193, HEXA MVS:C1, DEC UNIX:065, HEXA UNIX:41, CAR=/A/  
ZOSDEC MVS:090, HEXA MVS:5A, CAR=/]/  
UNIXDEC MVS:090, HEXA MVS:5A, DEC UNIX:093, HEXA UNIX:5d, CAR=/]/
```

Adapting the COBOL CONVERTMW.cpy Copy File

Adaptations are required when the graphical characters displayed on the UNIX reference monitor are different from the characters displayed on the z/OS reference monitor.

An adaptation consists in modifying the hexadecimal value stored in the TRANSCODE-CIBLE variable of the COBOL CONVERTMW.cpy copy file.

Finding the z/OS Character

Look for the original hexadecimal value in the TRANSCODE-SOURCE section by reading the characters in pairs.

The same value may appear to be present several times it is important to read the hexadecimal values in pairs.

Using a hypothetical example, where under z/OS the source hexadecimal character EA is not "2 " but "# ".

The two monitors indicate

```
ZOS DEC MVS:234, HEXA MVS:EA, CAR=/#/  
UNIX DEC MVS:234, HEXA MVS:EA, DEC UNIX:178, HEXA UNIX:b2, CAR=/2/
```

Figure 2-5 Looking for z/OS Character Part 1

```

01 TRANSCODE-SOURCE.
*
02 FILLER PIC X(32) VALUE X"000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f".
02 FILLER PIC X(32) VALUE X"202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f".
02 FILLER PIC X(32) VALUE X"404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f".
02 FILLER PIC X(32) VALUE X"606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f".
02 FILLER PIC X(32) VALUE X"808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f".
02 FILLER PIC X(32) VALUE X"a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbef".
02 FILLER PIC X(32) VALUE X"c0c1c2c3c4c5c6c7c8c9cacbcccdcecfdd0d1d2d3d4d5d6d7d8d9daddbdcddedf".
02 FILLER PIC X(32) VALUE X"e0e1e2e3e4e5e6e7e8e9eabececedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff".

```

The string "ea" can be found on the sixth and eighth lines. Which string should be chosen?

The "ea" string on the sixth line corresponds to the intersection of two strings "ae" and "af", whereas the "ea" string found on the eighth line is the one we are looking for. It starts in the 21st position of the eighth line, so is the 11th hexadecimal value on the line (a hexadecimal value being composed of two characters).

Figure 2-6 Looking for z/OS Character Part 2

```

01 TRANSCODE-SOURCE.
*
02 FILLER PIC X(32) VALUE X"000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f".
02 FILLER PIC X(32) VALUE X"202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f".
02 FILLER PIC X(32) VALUE X"404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f".
02 FILLER PIC X(32) VALUE X"606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f".
02 FILLER PIC X(32) VALUE X"808182838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9f".
02 FILLER PIC X(32) VALUE X"a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9babbbcbdbef".
02 FILLER PIC X(32) VALUE X"c0c1c2c3c4c5c6c7c8c9cacbcccdcecfdd0d1d2d3d4d5d6d7d8d9daddbdcddedf".
02 FILLER PIC X(32) VALUE X"e0e1e2e3e4e5e6e7e8e9eabececedeeeff0f1f2f3f4f5f6f7f8f9fafbfcfdfeff".

```

Finding the UNIX Characters to Replace

Once the hexadecimal value has been found in the z/OS file, it is easy to retrieve the value of the variable TRANSCODE-CIBLE that is situated in the same location (8th line, 21st character) of TRANSCODE-SOURCE. Using the show_convertmw.sh script we can see that the hexadecimal value in the TRANSCODE-CIBLE is "b2".

Replacing the UNIX Character

The value “b2” should be modified to the hexadecimal equivalent of “#”. The hexadecimal value is “23”, as indicated by the file produced by running `show_convertmw.sh`:

```
DEC MVS:123, HEXA MVS:7B, DEC UNIX:035, HEXA UNIX:23, CAR=/#/
```

Remarks Concerning the Example

- The replacing of " 2 " by " #" is arbitrary and for illustration purposes only.
- After this replacement, the “#” character is twice present in the TRANSCODE-CIBLE variable, the relation is no longer bijective (there is no longer a one-to-one correspondence between the two sets of characters. This asymmetry does not cause problems for the migration of z/OS data to UNIX, but it destroys the relation:
 - One z/OS character = one UNIX character
- The EURO (€) character is displayed graphically as '⌘' because the terminal used in the UNIX example did not contain the " € " character, but only the "Monetary" graphical character.

Special Characters

Special characters such as:

- LOW-VALUE
- SPACE
- HIGH-VALUE

should keep the same decimal value between the source and target platforms to preserve the iso-functional behavior of COBOL between the different platforms.

Using the COBOL CONVERTMW.cpy File

This copy file is used by the COBOL reloading programs generated by the Rehosting Workbench file and database migration tools. The copy file is installed during the installation of the Rehosting Workbench.

Check the directories indicated in the UNIX variable:

```
COBCPY
```

used by the COBOL compiler.

Error Messages

DATART-1001:

Example: COPY file \$convertmwCopyFile not found. Check argument 1.

Explanation: Argument 1 must contain the COBOL copy file name.

DATART-1002:

Example: can not access to directory \$TMPPROJECT: \$TMPPROJECT.

Explanation: UNIX Variable TMPPROJECT must be set.

DATART-1003:

Example: bad status returned by awk.

Explanation: Check previous messages.

See Also

- [DB2-to-Oracle Convertor](#) especially [Reloading the Data](#).
- [File Convertor](#): Introduction, especially [Reloading the Data](#).
- Oracle Tuxedo Application Rehosting Workbench user Guide.

Cataloger

The Oracle Tuxedo Application Rehosting Workbench Cataloger analyzes all the components extracted from the source environment separately and together in order to determine whether the asset is consistent and can be migrated. The Cataloger also produces an internal form to be used by other tools.

Overview of the Cataloger

The Cataloger is one of the migration tools composing the Rehosting Workbench. Its purpose is twofold:

1. Analyze separately and together all the components in the source software system to determine whether this system is consistent and can be migrated. The anomalies detected during this analysis are reported in the cataloging reports, along with other inventory information.
2. Produce an internal form of these components and of their mutual relationships, to be used by the other tools in the Rehosting Workbench.

Inputs to the Cataloger Process

- The source files for the components in the source asset, after they have been transferred from the source platform to the migration platform and converted to be better viewed and processed;
- One or more configuration files:

- The [System Description File](#) (mandatory), which describes how the input source files are organized on the migration platform file system, and also gives additional parameters necessary for their parsing;
- The Cataloger option file (optional), which gives parameters for the analysis phase of the Cataloger (see [Detailed Processing](#)).
- The [JCL-Launcher Specification Files](#) are used to describe the launchers used in a given asset, so that the cataloguer and the JCL translator can extract relevant information such as the name of the real program to launch.
- Hint files (optional), which give information that the Cataloger cannot find out by itself, for instance on dynamic program calls.

Outputs from the Cataloger Process

- A set of cataloging reports, in CSV form, describing the components and their status in the asset (correct, unused, missing);
- For each (parsable) component, a binary pob-file containing the internal form of the component, suitable for further processing such as conversion or translation;
- Other system-wide pob files, such as the Symtab (see [The Cataloger Symtab and Other Miscellaneous Files](#)), for use by the Cataloger and other Oracle Tuxedo Application Rehosting Workbench tools;
- When necessary, some other non-binary files for use by other Oracle Tuxedo Application Rehosting Workbench tools.

The Cataloger Process

The Cataloger process is divided into four logical phases:

1. Parsing: each component in turn is read, parsed (syntactic analysis) and linked (semantic analysis), and the corresponding pob-file is generated.
2. Analysis: for each component, the pob-file is re-read and the most significant constructs in the component are translated into a smaller summary information stored in the Cataloger symbol table (symtab).
3. Post-analysis: working with just the symtab and the summary information, the Cataloger computes some cross-reference links allowing to label each component as correct, unused or missing.

4. Report generation: the symtab decorated with cross-reference links is traversed and information is printed out for each component.

Depending on the needs of the project and the migration-platform configuration, these phases can be executed sequentially or concurrently, in a single run or incrementally. See [Repetitive and Incremental Operation](#) and the Oracle Tuxedo Application Runtime Process Guide for further information.

Description of the Input Components

The Cataloger accepts as input the source files of a complete, working software application running on a z/OS platform. It should be composed entirely of the following types of files which are described in greater detail in the following sections:

- COBOL programs, (possibly containing EXEC SQL and/or EXEC CICS statements),
- JCL scripts,
- SQL DDL scripts defining the database schema,
- CICS RDO files defining the CICS transactions,
- Sub-files of the above types, when relevant.

COBOL

References

The Oracle Tuxedo Application Rehosting Workbench COBOL parser accepts the COBOL language as specified in the IBM Enterprise COBOL for z/OS Language Reference Version 3 Release 4 (document number SC27-1408-04).

Restrictions

The following constructs or features are not accepted:

- Multiple programs per compilation unit, especially nested programs.
- All object-oriented features.
- DBCS (`USAGE DISPLAY-1`) and Unicode (`USAGE NATIONAL`) constructs.
- Millennium Language Extensions (date fields).

Embedded CICS

References

The COBOL parser uses a sub-parser to parse embedded EXEC CICS statements (commands). The parser accepts the language defined in IBM CICS Application Programming Reference Version 3 Release 1 (document number SC34-6434-05).

SQL

References

The same parser is used in standalone mode to parse SQL DDL files and as a sub-parser to parse EXEC SQL code embedded in COBOL programs. It is based on the language specifications in IBM DB2 Version 9.1 for z/OS Application Programming and SQL Guide (document number SC18-9841-00) and IBM DB2 Version 9.1 for z/OS SQL Reference (document number SC18-9854-00).

JCL

References

The JCL parser is based on the language specification in IBM z/OS MVS JCL Reference (document number SA22-7597-09).

General Information

Sub-Files

The parser processes various forms of sub-files and file inclusion directives (EXEC [PROC], INCLUDE, SYSIN, ...) and searches the asset for sub-files as directed in the [System Description File](#). Since the cataloger and other tools such as the JCL translator need to have a complete understanding of all of the steps in all of the JCL scripts that they handle, it is very important that all the referenced sub-files be present in the asset and that file types and search paths be set so that the correct sub-file is found for every reference. The cataloger will report missing sub-files as severe anomalies, since they prevent the correct analysis and translation of the whole affected JCL(s). Translation should not be attempted until all such anomalies have disappeared.

There are two types of SYSIN files:

- SYSINs (or SYSTSINs, or other "command files") for utility programs: in many cases, these files contain information which is of interest for the cataloger (e.g. the name of the program launched by the DB2 launcher IEKJFT01) or for the translator (e.g. operations performed by IDCAMS, or the sort script for DFSORT). In consequence, they must be present in the asset before cataloging (with type JCL-Sysin, see [Type clause](#)).
- SYSINs for applicative (COBOL) programs: these files will be handled like standard data files and do not need to be present in the asset processed by the Rehosting Workbench.

All referenced PROC and INCLUDE sub-files need to be present in the asset.

Note that the parser also handles in-stream PROCs (delimited by PROC and PEND cards) and SYSINs (DD *), but of course these are never missing.

JCL Syntax

A JCL job is a sequence of steps, with or without execution conditions. It must begin with a JOB card, except if the job-card-optional option is given in the cataloger option file, see XXXXX.

All JCL, JES2 and JES3 statements are parsed. The JCL must be directly executable by JES2. The parser performs JES variable substitution. Variables which are not defined locally in the JCL may be set using the JCL-globals option of the system description file, see [Special Options](#).

Comment cards (starting with "/*") are recognized as such and retained for translation.

The parser recognizes all kinds of JCL cards. It handles overrides and rebacks in PROCs, but only for DD cards. It also handles continuation cards.

Restrictions

Using a Sub-file as Both a PROC and an INCLUDE File

In certain conditions, the same sub-file can be used both as a PROC file and as an INCLUDE file. However, the translation to target files is different in each case, so it is necessary to duplicate the file(s) in question, so that one copy is used and translated as a PROC and the other as an INCLUDE file. To achieve this, the two copies must, of course, be placed in separate directories, and the search paths must be set up so that the JCL which use these files as PROCs find the PROC version first and those which use them as INCLUDEs find the INCLUDE version first (it is not possible that the same JCL uses the same file as both a PROC and an INCLUDE).

Using One JOB Card Per JCL

Only one JOB card per JCL is allowed. If you have files with more than one JOB card, you must split them before running the cataloger. Make sure that the job name in the JOB card and the (simple) file name match.

Standard Utility Commands Parsed

The JCL parser fetches (when not in-stream) and parses the contents of command files (SYSIN, SYSTSIN, etc.) for various standard utilities. The current list of such utilities is:

- IEBUPDTE
- IEBGENER
- IEBCOMPR
- IEBCOPY
- IEHPROGM
- IEBDG
- IEHLIST
- IEBMOVE
- IEBTPCH
- IDCAMS
- DFHSORT
- ICETOOL
- IEH-MOVE
- IEKJFT01
- DSNUTILB
- DSNUTIAUL.

Note: The support for some of these utilities, i.e., the ability of the parser to handle their command language, may be partial.

BMS screen definition

The BMS parser handles the BMS language as defined in the following documents:

- Chapter *BMS macros* in appendix *Detailed reference information for the CICS API commands* of the IBM book *CICS Application Programming Reference* (document number SC34-6434-05 for CICS V3R1);
- Chapter *Creating the map* in Part 6, *Basic Mapping Support (BMS)* of the IBM book *CICS Application Programming Guide* (document number SC34-6433-04 for CICS V3R1).

The parser will accept all correct BMS definitions. It will also report the most obvious syntax errors, but it is not meant to recognize all such errors.

CICS Configuration

The CICS configuration parser handles the CICS resource definition language as read by batch utility DFHCSDUP (see IBM CICS Transaction Server for z/OS Resource Definition Guide Version 3 Release 2, document number SC34-6815-00). The following commands are recognized: DELETE ALL, ADD, REMOVE and DEFINE. All resource types and attributes are recognized but only a few are really exploited in the parser and extractor.

Description of the Configuration Files

System Description File

The system description file describes the location, type and possible dependencies of all the source files in the asset to process. As such, it is the key by which not only the Cataloger, but also all of the Rehosting Workbench tools, can access the source files and the corresponding components. The system description file also specifies a number of parameters which influence parsing.

General Structure

Listing 3-1 System Description File Structure

```
Sys-desc-file ::= "system" system-name "root" system-root-path
                global-options special-options
                directories
```

Notes: The format of the file is basically free, lines can be as long as desired. Comments start with the percent character and end at the end of the line.

Notes: The format of symbols (names) can include the following characters [A-Z][a-z][0-9][*-_]. Symbols may start with digits but at least one letter is required. Keywords and symbols (names) are case-insensitive; strings are of course case-sensitive.

system-name

The first element in the system description file is a symbol giving the name of the asset. This name can be freely chosen, since it is used only by the Rehosting Workbench tools for reference. The names of some files and directories produced by the Rehosting Workbench tools also contain this name.

system-root-path

The second element is a string giving the path of the directory which contains all component source files on the Linux migration platform. This directory can be located anywhere convenient on the file system. The path can be given either in absolute form (starting with the slash character) or in relative form. In the latter case, the path is relative to the directory containing the system description file itself (usually located in some “param” directory besides the “source” directory containing the source files, but the Rehosting Workbench tools accept any configuration described here).

Global Options

The elements in this clause specify various settings influencing the parsing, cataloging and, generally speaking, handling of component source files. The generic syntax for this clause is:

Listing 3-2 System Description File Global Options

```
( "options" | "global-options" ) opt-name-1 "=" opt-value-1 ","  
                                opt-name-2 "=" opt-value-2 "," ... "."
```

The value of each option can be an integer number, a symbol, a string or a Boolean indicator. (The following are accepted as Boolean indicators:

- nothing (meaning true).
- the string “true” or “TRUE”.
- The symbol true (case-insensitive).

- the string “false” or “FALSE”.
- The symbol false).

Option names and option values are case-insensitive, except for strings. In general, these settings can apply globally on the whole asset and/or be overridden locally for a specific directory (see below).

The various possible options accepted here are listed in the following table

Table 3-1 Global Options

Option Name	Type	Local	Description
Catalog, catalog-option	String	No	Path of the Cataloger options file (see below). This path can be given in absolute form or in relative form; in the latter case, the path is relative to the directory containing the system description file itself. Note that this clause is optional: if it is not given, then the Cataloger will not attempt to read an option file and will use default values for all options.
Cobol-Right-Ma rgin	integer (> 60)	Yes	Column for start of Area C in COBOL programs. The default value is 66, suitable for fixed-format programs with columns 1-6 and 72-80 removed.
Cobol-Left-Mar gin	integer (< 12)	Yes	Comment column in COBOL programs. The default value is 1, suitable for fixed-format programs with columns 1-6 and 72-80 removed.
Remove-Cobol-R eserved-Word	String	Yes	This option is used to tell the COBOL parser that some keyword it considers as reserved (in the IBM COBOL LE dialect) is in fact not reserved in the actual dialect used for this asset or directory. This option may be given several times for the same system or directory. For each program, the list of such keywords to consider as non-reserved is formed by accumulating all values given in all local (directory) or global (system) options of this name. Default value is of course the empty list.

Table 3-1 Global Options

Option Name	Type	Local	Description
No-END-Xxx-Warnings, No-END-Xxx	Boolean	Yes	If true, don't complain loudly when some statement-containing construct is not closed otherwise than with the appropriate END-xxx keyword. Default value is false, i.e. complain. Note that you can associate a “false” value with this option, which would reverse its meaning.
Yes-End-Xxx-Warnings, Yes-End-Xxx, END-Xxx-Warnings	Boolean	Yes	The reverse of previous option, No-END-Xxx-Warnings. You may not use both options on the same directory or on the global level, but you may have an option at the global level and a different option on some directory(ies). Note that you can associate a “false” value with this option, which would reverse its meaning.
SQL-Schema, Default-SQL-Schema	string or symbol	Yes	Name of the schema to use in SQL code when not explicitly given. This applies to standalone SQL code (DDL, files of type SQL-Script) and to SQL code embedded in COBOL programs.
SQL-Use-Reversed-Delimiters, SQL-Reversed-Delimiters	Boolean	Yes	if true, SQL code in this directory or system uses double quotes for strings and single quotes for delimited identifiers. Default behavior (for value false) is the reverse, single quotes for strings and double quotes for delimited identifiers.
SQL-No-Keyword-Is-Reserved, SQL-Keywords-Not-Reserved	Boolean	Yes	If true, no keyword in the SQL code in this directory or system will be considered as reserved (use for DB2 version 8 or earlier). Default behavior (for value false) is to consider all keywords as reserved, and hence complain if they are used for other purposes (like DB2 version 9).

Table 3-1 Global Options

Option Name	Type	Local	Description
SQL-Right-Marg in	Integer (> 0)	Yes	The “end-of-line” column for SQL-Script files (only—this does not apply to SQL code embedded in COBOL programs). Set it to 66 for fixed-format files with columns 1-6 and 72-80 removed (à la COBOL). Default value is “infinite”, which is suitable for free-format files. Note that left margin is always 1, so you <i>must</i> physically remove columns 1-6 of fixed format files if they are to be ignored.
jclz-launcher- spec-file, jclz-launcher- specs-file	string	Yes	Path of the JCL-launcher specification file to use for this system or directory; see JCL-Launcher Specification Files for more information on the contents and use of such files. This path can be given in absolute form or in relative form; in the latter case, the path is relative to the directory containing the system description file itself.

Special Options

Special options are clauses which cannot be integrated in the previous global options mechanism, mostly for syntactic reasons (values are lists). They can appear before or after global options, but not in-between. They are all of the following syntactic form:

Listing 3-3 System Description File Special Options

```
opt-name "=" opt-value "."
```

The equal sign and trailing period are mandatory. When a value is a list, the items in the list must be separated by commas. The special options are described in the following table:

Table 3-2 Special Options

Option Name	Type	Description
minimum-free-ram -percent	Integer (> 0 and < 100)	The fraction of physical memory which should remain free and available to other processes during execution of the Cataloger and all of the various Oracle Tuxedo Application Rehosting Workbench tools. In general, these tools consume more and more memory, depending on the number of components they process. When this limit is reached, the tool stops and restarts execution; incremental execution ensures that the components already processed are not re-processed, so that eventually, all the required work is achieved.
JCL-globals	List of pairs <i>var-name</i> = <i>var-value</i> , separated by commas	<i>Var-name</i> is a symbol (or string interpreted as a symbol) and <i>var-value</i> is a string. When parsing a JCL script, the parser simulates the JCL-variable substitution process performed by JES2. The name-value pairs given here are used to substitute global variables (as opposed to parameters, etc.). The parser reports an error when it cannot find a suitable value for some variable.
strict-jcl-libra ries	None (Boolean flag)	The presence of this option influences how SYSIN files referenced by a JCL are searched in the whole system. See chapter <i>Sub-file search operation</i> below.
Dbms-version	String or symbol	Version of the DB2 relational DBMS used on the source platform. This is used by the RDBMS tool (q.v.).

Directories

The main component of the system description file is the list of directory clauses, which specifies the location of the various source files for the given asset, their type and their relation with each other. Each such clause has the following syntax:

Listing 3-4 System Description File Directories

```
"directory" directory-path
    type-clause file-clause logical-name-clause options-clause
    libraries-clause sql-libraries-clause
```

subdirectories "."

The (mandatory) directory path must come first. The optional subdirectories, if any, must come last. The other clauses may come in any order. Of these clauses, only the type clause and the file clause are mandatory, the others are optional.

Directory-path

This is a string giving the path (location) of the directory relative to the root directory of the system (see [system-root-path](#)). Although it is not an absolute requirement, it is strongly advised that all the directories of the same system are physical descendants of the system root directory. (A simple and readable way to achieve this is that no directory path contains the "../" upward-going name). Different directories **must** have different paths.

Type clause

"type" directory-type

The type clause specifies the type of the directory, that is the type of the source files (components) it contains. The type is given as a case-insensitive symbol. Only the following types are accepted

Table 3-3 Valid Directory Type Clauses

Type	Description
Cobol-Batch	Main COBOL programs used in batch operation (referenced by JCLs), possibly containing EXEC SQL code (DML, data manipulation language).
Cobol-TPR	Main COBOL programs used in TP operation (referenced by transactions and CICS XCTL commands), possibly containing EXEC SQL and EXEC CICS code.
Cobol-Sub	COBOL subprograms, either batch or TP, possibly containing EXEC SQL and EXEC CICS code.
COBOL-Library, COBOL-Copy	COBOL copy files (copy books), to be included in main program files.
SQL-Script	Standalone SQL code containing essentially DDL (data definition language) statements. The set of SQL-Script files collectively defines the database schema(s) used in the system.

Table 3-3 Valid Directory Type Clauses

Type	Description
JCL	Main JCL files, defining one or more JCL jobs.
JCL-Lib	JCL sub-files, either defining procedures invoked by EXEC or containing statements invoked by INCLUDE
JCL-Sysin	SYSIN files used by utility programs or program launchers in JCL scripts. Not all SYSIN files are required by the parser/Cataloger, see more details in the Rehosting Workbench <i>JCL Translator Reference Manual</i> (section Description of Input Components).
BMS	BMS screen definitions (in source form)
RDO	CICS system definition files (CSD) as used by RDO to configure CICS, see IBM's <i>CICS Resource Definition Guide</i> .

files Clause

```
"files" file-specs ",", ...
```

The `file-specs` are strings designating one or more files in a directory. The string identifies the inclusive members of the asset and excludes the others. The simplest form of `file-spec` is a complete file name such as `toto.cbl`. No indication of directory should be given, the designated files must be located directly in the directory in question. To avoid the task of explicitly listing all components in the directory, you can also use shell-like regular expressions such as `*.cbl` or `[A-F][D-Z]*.jcl`.

Note: It is important that all files designated by the system description file, that is all source files in the asset, have a file extension rather than just a bare file name. The extension can be chosen freely — although we advise the use of “standard” extensions such as `cbl` for COBOL programs, `cpy` for COBOL copy files and `jcl` for main JCL files—but must be present.

logical-name clause

```
logical-name lname
```

This clause is to be used on directories of type JCL-Sysin, together with the special option `strict-jcl-libraries`, see above. Together, they enable the [Strict JCL-Sysin Search](#) mode. `lname` is a string of the form “A.B.C”, naming a library (PDS) of JCL SYSIN files on the source platform. It is assumed that all the files in the directory bearing this clause belong to this library.

If the special option `strict-jcl-libraries` is not set, the logical-name clause is ignored.

options-clause

Listing 3-5 options-clause

```
"options" opt-name-1 "=" opt-value-1 ","  
          opt-name-2 "=" opt-value-2 "," ...
```

Syntactically, the directory-specific `options` clause is similar to the system-wide [Global Options](#) clause, except for the trailing period. Semantically, the listed options and values have the same effect as the global options, but only locally on the files contained in the directory (they override global options with the same name). The same options as the ones marked `yes` in the `local?` column of the global-option table apply to directories, provided that they are relevant for the type of source files in the directory. For instance, the option `cobol-right-margin` is relevant for directories of type COBOL-Batch, COBOL-TPR or COBOL-Sub, but not for type JCL or SQL-Script.

In addition, there exists one directory-specific option: “Right-margin” for directories of type JCL. The value is an integer number which specifies the “end-of-line” column for JCL files. The default value is 72, which is appropriate for most cases of IBM JCL source files.

libraries-clause

```
"libraries" directory-path "," ...
```

The `libraries` clause specifies an ordered search path of (other) directories in the asset. Whenever the Cataloger finds in a source file a reference to another component it searches, from first to last, the list of directories given in this clause, until it finds a component (source file) whose name and type matches those of the reference (see more details in section *Sub-file search operation* below). It is used both for compile and parse-time references, such as a COBOL program referencing a copy file or a JCL file referencing a PROC, and for run-time references, such as a COBOL program calling a COBOL subprogram or a JCL job invoking a COBOL program. This way, it is possible to simulate the effects of various source-platform library-search operations, such as SYSLIB or COPYLIB for COBOL compilation, JOBLIB and STEPLIB for JCL preparation and execution, etc.

Note: The directory paths are strings which must match those given in the definition of the referenced directories. However, the definition of a directory may be placed before or after any of its references.

sql-libraries-clause

`"sql-libraries" directory-path "," ...`

The SQL-libraries clause plays the same role as the libraries clause for resolving EXEC SQL INCLUDE directives in COBOL programs. When it is omitted, the resolution of such references uses the same search path as the normal libraries clause, but sometimes it is necessary to use a different order for normal COPY directives and SQL INCLUDE directives.

Example of System Description File

Listing 3-6 Example System Description File

```
system BNL root "../source"

options catalog = "../options-catalog.desc",
    no-end-xxx-warnings,
    cobol-left-margin = 7,
    cobol-right-margin = 72,
    SQL-Schema = DB2A1,
    SQL-Server = BNL.

minimum-free-ram-percent = 20.

%
% Copies
%

directory "COPY"    type Cobol-Library files "*.cpy".
directory "INCL"    type Cobol-Library files "*.cpy".
```

```

directory "IBMCPY" type Cobol-Library files "*.cpy".

%
% Sysin
%
directory "SYSIN"      type JCL-SYSIN files "*.sysin".
directory "SYSINCDDB"  type JCL-SYSIN files "*.sysin".

%
% DDL
%
directory "DDL"  type SQL-SCRIPT
                files "*.sql"
                options SQL-Schema = "DB2A0".

%
% Batch
%
directory "Batch" type COBOL-Batch files "*.batch"
                libraries "COPY", "INCL", "IBMCPY"
                options cobol-right-margin=73.

%
% TPR
%
directory "TPR" type COBOL-TPR files "*.tpr"
                libraries "COPY", "INCL", "IBMCPY".

```

```

%
% JCL
%
directory "JCL" type JCL files "*.jcl"
        libraries "SYSINCDB", "SYSIN".

%
% CICS
%
directory "MAPS" type BMS files "*.bms".
directory "CICS" type RDO files "*.rdo".

```

This system-description file is for an asset named BNL, for example the name of a customer or a standalone application in a larger system. The location and name of this file are not constrained, but conventionally, the complete path should be something like:

`../../BNL/param/system.desc`. Given this assumption, and since the path for the system root directory given in this file is relative (`../source`), the absolute path for the root directory is `../../BNL/source`. Similarly, the path for the Cataloger options file is given as `../options-catalog`, so its absolute path is `../../BNL/param/options-catalog`. The global options call for the following comments:

- The `no-end-xxx-warnings` option enables the lenient mode of parsing implicitly-closed COBOL constructs.
- The `cobol-left-margin` and `cobol-right-margin` values are set for untransformed, IBM-like fixed-format programs with left-side numbering column and right-side comment column (area C). Note that, while this format causes no trouble for the COBOL parser, the correct operation of the COBOL converter cannot be guaranteed.

The naming and organization of the various directories is quite standard, with source files in the asset being identified only with their file extensions. The only unusual feature here is the special `cobol-right-margin` value for directory “Batch”.

JCL-Launcher Specification Files

Purpose

Most IBM source assets contain JCL steps invoking program launchers, i.e. utility programs that launch applicative programs. Many of these launchers, such as the DB2 launcher IEKJFT01, are recognized directly by the JCL parser and analyzer in the Rehosting Workbench cataloger. However, in many cases, some of these launchers are installation-specific and require specific handling. Fortunately, most of them use the generic JCL-invocation mechanism and syntax (EXEC PGM card) and the relevant launch information is contained in the PARM value.

The purpose of the JCL-launcher specification file are to describe the launchers used in a given asset, so that the cataloger and the JCL translator can extract relevant information such as the name of the real program to launch. The specification is based on the fact that, in most cases, the PARM value is split into individual parameters by some separator character (not always the standard JCL separator, the comma), and that the parameters which give the program name, the PSB name, the PLAN name, etc., have a well-defined position in the sequence.

Syntax

A JCL-launcher specification file is a free-format text file with the following syntax, where all the keywords and symbols are case-insensitive:

Listing 3-7 JCL-launcher Syntax

```
LAUNCHER <Launcher name>

[<option-name> = <option-value> [,
... ]
]
END

...
```

Option List

For the last three options, there is no default value: if the option is absent, then the corresponding information is simply not available.

- **Separator:** Character used as a field separator, must be a one-character string. Default is the comma character ",".
- **IndexProg:** index of program name, must be an integer. This is the only mandatory option for a given launcher.
- **IndexPSB:** index of PSB name, must be an integer.
- **IndexPlan:** index of plan name, must be an integer.
- **IndexParm:** index of parameter name, must be an integer.

Usage and Default Value

The local `jclz-launcher-spec-file` option attached to a directory, when present, overrides the global one, as usual. When no launcher specification file is specified either for a given directory or the whole system, then the default value is as if we used the following file:

Listing 3-8 Default Launcher Value

```
LAUNCHER DFSRRC00

IndexProg : 2,

IndexPSB : 3

END

LAUNCHER DB2BATCH

IndexProg : 2,

IndexPSB : 3

END
```

Description of the Output Files

Catalog Reports

Format and Location

All these reports are produced in CSV format, with fields delimited by a single semi-colon.

They are generated in the `$SYSROOT/Reports-${SYSNAME}` directory, where `$SYSROOT` is the root directory for the current asset and `$SYSNAME` is the asset name, both as defined in the [System Description File](#).

The name of each report also contains `$SYSNAME`, to avoid any confusion.

Field Definitions

The following field definitions are used in several reports:

Path (string)

The identification of the (main) source file defining the entity in question, as a path relative to the root of the "system" given in the system description file.

Status (enumeration: CORRECT, UNUSED or MISSING)

CORRECT

The component is present in the asset and at least one reference to it has been found in one or more other components, i.e. the component is used.

UNUSED

The component is present in the asset but no reference to it could be found in any other component;

MISSING

The component is not present in the asset and at least one reference to it has been found.

Note: UNUSED and MISSING are to be considered as (inter-component) anomalies.

Anomaly level (enumeration: FATAL, ERROR, WARNING, NOTICE, OK)

This is the maximum level of anomalies detected on the component in question during internal analysis.

FATAL

Irrecoverable errors such as syntax errors found. The results of the analysis are incomplete and the component or asset is unsuitable for conversion.

ERROR

Recoverable errors such as undeclared variables found. The results of the analysis may be inaccurate and the component or asset is unsuitable for conversion.

WARNING

Situations which can cause problems (inaccuracies) during analysis or after conversion have been found, but the component is suitable for conversion.

NOTICE

A remarkable situation was detected, but it causes no harm.

OK

No anomaly found.

Note: When the component is MISSING, this field is replaced by indications describing the cause for the component to be absent from the asset (SYSTEM, CORRECT or PROBLEM), depending on information supplied entirely by the user.

MISSING

When a component is MISSING, this field is replaced by indicators describing the cause for the component to be absent from the asset (SYSTEM, CORRECT or PROBLEM), depending on information supplied entirely by the user.

report-`${SYSNAME}`-COBOL-Programs

This report lists all the (COBOL) programs defined or referenced in the asset. It accounts for the -Cobol-Batch, -Cobol-TPR and -Cobol-Sub reports.

The following fields are contained in the report:

Table 3-4 COBOL Report Fields

Field	Type	Description
Name	symbol	Name of the program as defined by the (envelope) file name.
Path	string	See Field Definitions . This is the path of the (main) source file defining the program. Note: Oracle Tuxedo Application Rehosting Workbench does not currently handle multiple programs in the same file.

Table 3-4 COBOL Report Fields

Field	Type	Description
Type	enum: BATCH, TP or SUB	Type of program as defined by its classification in the system description file (Cobol-Batch, Cobol-TPR or Cobol-Sub).
Loc	integer	Total number of lines in the program after copy expansion.
NPar	integer	Number of paragraphs in the Procedure Division.
Status	enum	See Field Definitions .
Anomaly level	enum	See Field Definitions .

The following fields are empty (undefined) when the component is MISSING:

- Path
- LOC
- Npar.

In addition, for components of type SUB, the name may be that of an entry point in a subprogram, rather than the name of the subprogram itself. It is the name as referenced in a `CALL` and the cataloger can't determine whether it designates an entry point or a complete subprogram.

report-\${SYSNAME}-COBOL-Copy

This report lists all the COBOL copy files (copybooks) contained or referenced in the asset. The following fields are contained in the report:

Table 3-5 COBOL Copy Report Fields

Field	Type	Description
Name	string	Logical name of the copy file, as defined by the (envelope) file name and possibly by the logical-name clause of the containing directory in the system description file.
Path	string	See Field Definitions .

Table 3-5 COBOL Copy Report Fields

Field	Type	Description
Loc	integer	Number of lines in the copy file.
Status	enum	See Field Definitions .

The following fields are empty (undefined) when the component is MISSING:

- Path,
- LOC.

report-`${SYSNAME}`-JCL-Files

For JCLs, we separate between reports on (main) source files and reports on jobs, because we handle multiple jobs per file. For (main) source files, the following fields are contained in the report:

Table 3-6 JCL source File Report Fields

Field	Type	Description
Path	string	See Field Definitions .
Loc	integer	Total number of lines in the JCL file after expansion of PROCs, INCLUDEs and SYSINs.
NJob	enum	Number of jobs defined in this file.
Anomaly level	enum	See Field Definitions . It is at least as high as the maximum anomaly level in all the contained jobs, and may be higher in case of syntax errors.

A JCL source file is never MISSING, only JCL jobs can be missing.

report-`${SYSNAME}`-JCL-Sub-Files

This report describes JCL sub-files required for the analysis of main files: PROCs, INCLUDEs and some SYSIN files. The following fields are contained in the report:

Table 3-7 JCL sub-file Report Fields

Field	Type	Description
Name	string	Name of the sub-file, as referenced from main files.
Path	string	See Field Definitions .
Type	enum: PROC, INCLUDE or SYSIN	Type of sub-file, as defined by its classification in the system description file and the construct by which it is referenced in the main files.
Loc	integer	Number of lines in the sub-file.
Status	enum	See Field Definitions .

The following fields are empty (undefined) when the component is MISSING:

- Path
- LOC.

report-`${SYSNAME}`-JCL-Jobs

This report lists all JCL jobs defined or referenced in the asset. The following fields are contained in the report:

Table 3-8 JCL Jobs Report Fields

Field	Type	Description
Name	string	Name of the job, as defined in the JOB card.
Path	string	See Field Definitions . This is the path of the (main) file defining the job.
Loc	integer	Number of lines in the job itself (from the JOB card to the ENDJOB card) after expansion of sub-files.
NStep	integer	Number of steps defined in this job

Table 3-8 JCL Jobs Report Fields

Field	Type	Description
Status	enum	See Field Definitions .
Anomaly level	enum	See Field Definitions . This is the anomaly level of the job itself, and generally does not take into account syntax errors (because the latter prevent the analysis of the job).

report-`${SYSNAME}`-Screens

This report lists all BMS screens defined or referenced in the asset. The following fields are contained in the report:

Table 3-9 Screens Report Fields

Field	Type	Description
Name	string	Name of the screen, in the form <i>mapset-name.map-name</i> .
Path	string	See Field Definitions . This is the path of the file defining the screen.
Line	integer	The number of the line in the source file at which the screen definition begins (the line containing the <code>DFHMDI</code> macro).
NField	integer	Number of fields defined in this screen
Status	enum	See Field Definitions .
Anomaly level	enum	See Field Definitions . In fact, this is the anomaly level of the complete source file; see the anomaly report to see whether the anomalies really apply to this screen definition.

When the screen is `MISSING`, the following fields are empty:

- Path
- Line
- NField
- Anomaly level.

report-`#{SYSNAME}`-SQL-Tables

This report lists all SQL tables defined or referenced in the asset. The following fields are contained in the report:

Table 3-10 SQL Table Report Fields

Field	Type	Description
Name	symbol	Name of the SQL table, as defined in the CREATE TABLE statement.
Schema	symbol	Name of the schema containing the table definition, or of its owner.
Path	string	See Field Definitions . This is the path of the SQL-script file defining the table.
Line	integer	Number of the line in this source file at which the definition of the table begins.
NCol	integer	Number of columns defined in the table.
Status	enum	See Field Definitions .
Comment	string	Comment, if any, associated with the table.

When the table is MISSING, the following fields are empty:

- Path
- Line
- Ncol
- Comment.

report-`#{SYSNAME}`-SQL-Views

This report lists all SQL views defined in the asset. The following fields are contained in the report:

Table 3-11 SQL Views Report Fields

Field	Type	Description
Name	symbol	Name of the SQL view, as defined in the CREATE VIEW statement.
Schema	symbol	Name of the schema containing the view definition, or of its owner.
Path	string	See Field Definitions . This is the path of the file defining the view.
Line	integer	Number of the line in this source file at which the definition of the view begins.
NCol	integer	Number of columns defined in the view.
Status	enum	See Field Definitions . Note: A view is never MISSING, because references to views are indistinguishable from references to tables. So, when a reference to a table-or-view links to no definition of any kind, the Cataloger creates a missing table, not a missing view.
Comment	string	Comment, if any, associated with the table.

report-`${SYSNAME}`-Transactions

This report lists all CICS transactions defined in RDO files in the asset. The following fields are contained in the report

Table 3-12 CICS Transaction Report Fields

Field	Type	Description
Name	symbol	Name of the transaction, as defined in the DEFINE TRANSACTION statement.
Group	symbol	Name of the group containing the transaction definition.

Table 3-12 CICS Transaction Report Fields

Field	Type	Description
Path	string	See Field Definitions . This is the path of the file defining the transaction.
Line	integer	Number of the line in this source file at which the definition of the transaction begins.

When a transaction is referenced in the asset (e.g. in a RETURN TRANSID statement) and it is not defined in an RDO file, it is listed in this report with empty Path and Line fields.

report-`#{SYSNAME}`-Anomalies

This report lists all anomalies found in all components of the asset. The following fields are contained in the report:

Table 3-13 Anomaly Report Fields

Field	Type	Description
Path	string	See Field Definitions . This is the path of the main file defining the component in which the anomaly occurs.
Sub-Path	string	See Field Definitions . If the real location of the error (statement or other construct) is inside some sub-file (COBOL copy file, JCL PROC file, etc.), this is the path of this sub-file, otherwise this field is empty.
Line	integer	Number of the line in the real source file at which the anomaly occurs: the sub-file if previous field is not empty, otherwise the main file.
Sub-Line	integer	If the anomaly occurs in some sub-file, this field contains the number of the line in the main source file at which the sub-file is included, otherwise it is empty.
Severity	enum: FATAL, ERROR, WARNING, NOTICE	The severity of the anomaly, from FATAL as the highest severity to NOTICE with the lowest severity.

Table 3-13 Anomaly Report Fields

Field	Type	Description
Category	enum: SYNTAX, LINKAGE, ANALYSIS, MISC	Defines the category to which the anomaly belongs: <ul style="list-style-type: none">• SYNTAX is for parse errors and all errors related to syntax;• LINKAGE is for errors related to links between a reference to some construct and the corresponding definition, such as undeclared variables;• ANALYSIS is for anomalies related to constructs which do not allow the Cataloger to perform an accurate analysis of the component, such as dynamic calls;• MISC is for all others.
Tag	Symbol	A synthetic but significant name identifying the precise kind of anomaly. In fact, the possible values are a finite enumeration, but too numerous to be all listed here.
Description	string	A precise and specific description of the anomaly, possibly including references to the offending source construct, for instance: "SQL-TABLE variable is not defined: LVDSYS00".

Execution Logs

Description of Other Output Files

The visible result of the Cataloger is the set of cataloging reports described above. These reports are far from the only or even the most important output. This section briefly describes the other result files; these are binary files in a proprietary format, called Persistent Object Base (POB). These files are not suitable for human processing or processing by traditional text-based tools; they are intended for use by the Cataloger itself or with other tools in the Rehosting Workbench.

POB Files for ASTs

During the parsing phase (see [The Cataloger Process](#)), for each parsable-component source file `A/B/C/file.ext` in the system, the Cataloger produces a POB file named `A/B/C/pob/file.ext.pob` (the `pob` directory is created on demand by the Cataloger). This file contains the result of the parsing, namely the [Abstract Syntax Tree \(AST\)](#) of the component. It is re-read by the analysis phase of the Cataloger and by other Oracle Tuxedo Application Rehosting Workbench tools such as the COBOL converter or JCL translator.

CDM Files for COBOL Programs and Copy Files

CDM (Common Data Model) files contain additional information about COBOL variables (so-called data description entries). For each COBOL program `A/B/C/prog.cbl` in the system, the Cataloger produces a CDM file named `A/B/C/pob/prog.cbl.cdm` to store information about variables defined in the main source file. In addition, for each COBOL copy file `D/E/file.cpy` which defines variables (as opposed to copy files containing Procedure Division code, for instance), the Cataloger produces a CDM file named `D/E/pob/file.cpy.cdm` to store information about those variables; this CDM file is shared by all programs which include this copy file.

In some circumstances, the information about a variable apparently defined in a copy file cannot actually be shared by all programs which include this copy file; for instance, this is the case for copy files included with REPLACING directives, or files defining only parts of a complete structure (01-level record). In these cases, the CDM information is stored in the programs CDM files rather than that of the copy file itself. When no shared CDM information at all can be associated with the copy file, the CDM file is not produced.

The Cataloger Syntab and Other Miscellaneous Files

- `$SYSROOT/syntab-$SYSNAME.pob`: this file houses the symbol table created by the Cataloger (during the analysis phase) and contains summary information for all the various components in the asset. This information is used to compute cross-reference information between these components.
- `$SYSROOT/Cobol-dump-map.pob`: contains information (so-called dump descriptors) necessary to read and write [Abstract Syntax Tree \(AST\)](#) pobs for COBOL programs. Do not delete this file or you will not be able to re-read your existing COBOL pobs.
- `$SYSROOT/sql-system-$SYSNAME.pob`,
`$SYSROOT/sql-system-$SYSNAME-State-ments.pob`: contains various internal forms of the complete SQL schema of the asset, derived from the union of all DDL files (SQL-Script files). These files are required for parsing (and linking) COBOL programs.

Detailed Processing

Processing Phases

As described in [The Cataloger Process](#), the operation is logically divided into four phases: parsing, analysis, post-analysis and report generation (see below for more details). Depending on

the needs of the project and the migration-platform configuration, these phases can be executed sequentially or concurrently (parsing phase only), in a single run or incrementally.

Depending on the needs of the project and the migration-platform configuration, these phases can be executed sequentially or concurrently (parsing phase only), in a single run or incrementally. There are three basic Oracle Tuxedo Application Rehosting Workbench commands invoking the Cataloger:

- `preparse` and its variant `preparse-files`: runs the parsing phase only.

This is the only phase which can be run concurrently, at least after the SQL-System files have been generated. This is also the only phase for which you can request the processing of one or more specific components; otherwise, the Cataloger determines itself which components it must process (see [Changes in the Asset: Incremental Operation](#)). In this phase, the Cataloger reads the component source files, any included sub-files and the SQL-System files, and produces (only) the POB-files for the processed components.

- `analyze`: runs the analysis phase: for each component, the pob-file is re-read and the most significant constructs in the component are translated into a smaller summary information stored in the cataloger symbol table (`syntab`).

This phase cannot run in concurrent mode because the Syntab does not support concurrent accesses. In this phase, the Cataloger reads the component POB files (parsing them on demand if necessary) and updates (reads and writes) the Syntab file.

- `fast-final`: runs both the post-analysis and report generation phases.
 - Post-analysis: working with just the syntab and the summary information, the cataloger computes some cross-reference links allowing to label each component as correct, unused or missing.
 - Report generation: the syntab decorated with cross-reference links is traversed and information is printed out for each component.

There is no need to run this phase concurrently, especially since it performs a system-wide operation. In this phase, the Cataloger reads the Syntab file (without trying to update it) and writes the cataloging reports.

There is also a combined command:

- `catalog`: runs in sequence the analysis phase (and hence the parsing phase, on demand), the post-analysis phase and the report generation phase.

Note: For all these commands, the whole configuration information comes from the system description file and the Cataloger option file. Except for `preparse-files`, the only

command-line arguments are the path to the system description file and standard Oracle Tuxedo Application Rehosting Workbench tool arguments.

Command-line Syntax

The Oracle Tuxedo Application Rehosting Workbench Launcher

The Cataloger is designed to be run through the `refine` command. The `refine` command is the generic Oracle Tuxedo Application Rehosting Workbench launcher that is used to launch the major Oracle Tuxedo Application Rehosting Workbench tools. The launcher handles various aspects of the operation of these tools, such as execution log management and the incremental and repetitive operations described below ([Repetitive and Incremental Operation](#)). The Oracle Tuxedo Application Rehosting Workbench launcher also handles a couple of generic command-line options.

Synopsis

The general form used to invoke an Oracle Tuxedo Application Rehosting Workbench tool using the command line is:

```
$REFINEDIR/refine command [ launcher-options... ] \  
    ( -s | -system-desc-file ) system-desc-path \  
    [ command-specific-options-and-arguments... ]
```

Options

The following options relate to the Rehosting Workbench *command*.

-h, -help, --help

Print out a short description (usage) of the *command*, and then exits.

-whoami

Prints out the version number and build history of the command, and then exits.

-archi64 / -archi32

Use the executable tool built for the specified architecture. The default is to use the tool for the native architecture of the host machine.

-quiet

Do not print anything in the log except errors (this is currently not obeyed by all tools).

-time

Display timing information at the end of the command execution.

-nolog

Disable log redirection so that the log appears on the terminal and is not captured into a permanent file.

-n, -N, -verbose, -VERBOSE

Prints out a description of which work (phase) needs to be performed on which components but do not actually undertake the work see [Changes in the Asset: Incremental Operation](#).

The following option is technically not a launcher option, but it is accepted (and in fact mandatory) in all of the Rehosting Workbench tools:

(-s | -system-desc-file) system-desc-path

Specifies the location of the [System Description File](#). As usual for Unix/Linux commands, the given path can be absolute or relative to the current working directory.

Note: Many other paths used by many of the Rehosting Workbench tools are then derived from the location of this file; this makes it easy to run the same command from different working directories.

In addition, the following option is reserved for future use (presently, it is accepted but otherwise ignored):

(-v | -V | -version) version-string**Generic launcher options**

Lastly, the launcher can be invoked without a command, using generic options:

\$REFINEDIR/refine (-h | -help)

Prints out a short generic description (usage) of the launcher itself.

\$REFINEDIR/refine -print-info-version

Prints out version information about the launcher itself, and more generally about the whole of the Rehosting Workbench.

System-Wide Commands

As explained in [Processing Phases](#), system-wide commands are preparse, analyze, fast-final and catalog. They operate globally on the whole asset. The generic command-line syntax for all these commands is:

```
$REFINEDIR/refine command [ launcher-options... ] \
    -s | -system-desc-file ) system-desc-path
```

There is no specific option for these commands: all configuration information is located in the system description file, the Cataloger option file and possibly the hint files.

The preparse-files Command

Description

Unlike the system-wide cataloging commands described above, which operate globally on the whole asset and decide by themselves which components to process, the preparse-files command allows you to specify yourself which component or components to parse.

The fact that you can specify which components to process makes the preparse-files command suitable for use in a makefile. In addition, it is amenable to concurrent execution, especially if you partition the set of source files into several lists and give each list to a separate process.

Note: Before parsing any COBOL programs, the Cataloger ensures that the SQL-system POB file is present and up-to-date with respect to all of the SQL DDL files. This may entail building or rebuilding the POB file, which may take some time.

Synopsis

The command line for preparse-files is as follows:

```
$REFINEDIR/refine preparse-files [ launcher-options... ] \
    ( -s | -system-desc-file ) system-desc-path \
    ( source-file-path | ( -f | -file | -file-list-file ) file-of-files
)...
```

Options

The extra options indicate which component source files to process:

source-file-path

Adds to the work-list the component source file designated by this path. The path must be given as relative to the root directory of the system, \$SYSROOT, even if the current working directory is different.

(-f | -file | -file-list-file) file-of-files

Adds to the work-list the component source files listed in the file designated by this path. The file-of-files itself may be located anywhere, and its path is either absolute or relative to the current working directory. The component source files listed in this file, must however be given relative to the root directory of the system.

You can provide as many individual components and or files-of-files as you wish. The work-list is built when the command line is analyzed by the Cataloger, and each of its elements is examined in turn:

- If the given path does not match any actual component source file in the system, an error message is printed out and the Cataloger skips to the next element.
- If the component identified by the given path is already parsed and its POB file is up-to-date with the source file, no action takes place, and the Cataloger silently skips to the next element.
- Otherwise, the component is parsed normally (and verbosely, unless the `-quiet` option is given in the launcher-options) and its POB file is produced.

Component Search Operation

This section describes how the Cataloger uses the libraries and sql-libraries clauses in the system description file to locate the components referenced in a specific construct of the currently processed component. The operation is slightly different depending on whether the reference is:

- As seen from the source platform
- A compile-time reference
- A run-time reference.

Compile-Time References

The compile-time case applies to references to sub-files which are an integral part of the current component, so that, if the sub-file is not found, the component cannot be analyzed and "understood" correctly. For example a COBOL copybook referenced from a COBOL main (program) source file.

JCL sub-files referenced from a JCL job, such as `PROC` files, `INCLUDE` files and some `SYSIN` files are also included because whereas on the MVS platform these sub-files are searched when the JCL job is run, hence it is a run-time reference; on the migration platform, the Cataloger has to resolve these references at parse-time, to make them available to the JCL translator, and hence

they qualify as compile-time references. Even `SYSIN` files are in this case, since they contain information which is needed at parse time, such as the program invoked by some DB2 launcher. In the Cataloger, "compile-time" is equivalent to parsing, and "run-time" is equivalent to post-analysis.

The search starts with a component identified as *SRCFIL* of a certain type *SRCTYP*, that is located in some directory *SRCDIR* and which references a component named *TGTFIL* of a certain type *TGTTYP*. The following table describes the various possible combinations:

Table 3-14 Compile-Time References

Source type (<i>SRCTYP</i>)	Construct	Target type (<i>TGTTYP</i>)	Search path
COBOL program (Cobol-Batch, Cobol-TPR, Cobol-Sub)	<i>COPY TGTFIL</i> <i>COPY TGTFIL</i> <i>REPLACING ...</i>	COBOL copybook (COBOL-Copy, COBOL-Library)	libraries
COBOL program	<i>EXEC SQL INCLUDE</i> <i>TGTFIL</i> <i>END-EXEC.</i>	COBOL copybook	sql-libraries, or libraries if not specified
JCL job	<i>EXEC TGTFIL, ...</i>	JCL PROC (JCL-Select)	libraries
JCL job	<i>INCLUDE TGTFIL...</i>	JCL INCLUDE (JCL-Select)	libraries
JCL job	<i>SYSIN DD</i> <i>A.B.TGTFIL...</i> <i>SYSIN DD</i> <i>A.B.C(TGTFIL)...</i>	JCL <i>SYSIN</i> (JCL-Sysin)	libraries

Normal Sub-File Search

The search algorithm process is as follows:

- For each directory *SUBDIR* listed in the `libraries` (or `sql-libraries`, if applicable) clause associated with the definition of *SRCDIR* in the system description file, in order, locate the definition of *SUBDIR* in the same file, then:
 - If there is no such definition, complain (this is done once and for all when the Cataloger starts and reads the system description file) and skip to the next element of the list.

- If the type of *SUBDIR* is not the *TGTYP* appropriate to the reference at hand, skip to the next element of the list.
 - Otherwise (type matches), search the list of component files in the directory (those which match the *files* clause):
 - If a component file with the base name *TGTFIL* (and some appropriate extension) exists, return it.
 - Otherwise, skip to the next element of the list.
2. When all library directories have been examined without finding an appropriate *TGTFIL*, return "not found".

Strict JCL-Sysin Search

This algorithm is modified for searching JCL-Sysin files in presence of the `strict-jcl-libraries` special option. When this option is set, the search for SYSIN file *A.B.TGTFIL* or *A.B(TGTFIL)* proceeds as follows:

- if there exists a directory of type JCL-Sysin with logical name “A.B”, then *TGTFIL* is searched exactly in this directory (base name search, according to the *files* clause and the physical extension); note that it is an error if two or more directories have the same logical name;
- otherwise, return “not found”, even if a file with base name *TGTFIL* exists in another JCL-Sysin directory.

This behavior implies that the `libraries` clause is ignored on directories of type JCL, at least when it comes to searching JCL-Sysin files (it is still valid to search JCL-Select files). On the other hand, as described above, if the `strict-jcl-libraries` special option is not set, the `logical-name` clauses on JCL=Sysin directories are ignored.

It is suggested to use strict search rather than path-based search when there exist many cases of duplicate names, i.e. many files with the same name in different libraries (PDS). In this case, indeed, it is easier to transfer the whole contents of each SYSIN library in a separate directory, and give the name of the library as the logical name of the directory, rather than try to order the various JCL-Sysin directory names in the `libraries` clauses of the JCL directories to ensure that the appropriate file is found at each reference.

Run-Time Reference

The run-time case applies to references to external components that are not really part of the referencing component. The referencing component can be analyzed or translated even in the

absence of the referenced component – even though this absence will cause improper execution. For example, a COBOL program calling a subprogram or a JCL job EXECuting a program. In the Cataloger, such references are handled during the post-analysis phase.

The search starts with a component *SRCFIL* of a type *SRCTYP* located in a directory *SRCDIR* referencing a name *TGTFIL* of a type *TGTTYP*. There are two cases to consider.

Unrestricted Search

This case applies when the `libraries` clause associated with directory *SRCDIR* does not contain any element (directory) of the type *TGTTYP*. This can be considered as the "default case". The search algorithm is then:

1. Gather the (unordered) set of components of type *TGTTYP*, by searching the directories of this type and analyzing the components they contain.
2. Record their (base) name.
3. Search this set for components of name *TGTFIL*:
 - If there exists exactly one of them, link it with the reference.
 - If there exists more than one of them, link all of them with the reference, but complain about ambiguous references.
 - If there are none, complain about the missing component.

Directed Search

Directed Search is similar to [Normal Sub-File Search](#) for compile-time references. It applies when the `libraries` clause associated with directory *SRCDIR* contains one or more elements (directories) of the type *TGTTYP*. The search algorithm is then.

1. For each directory *SUBDIR* listed in the `libraries` clause associated with the definition of *SRCDIR* in the system description file, in order, locate the definition of *SUBDIR* in the same file, then:
 - If there is no such definition, complain (this is done once and for all when the Cataloger starts and reads the system description file) and skip to the next element of the list.
 - If the type of *SUBDIR* is not the *TGTTYP* appropriate to the reference at hand, skip to the next element of the list.
 - Otherwise (type matches), search the list of component files in the directory (those which match the `files` clause):

- If a component file with the base name *TGTFIL* (and some appropriate extension) exists, return it.
 - Otherwise, skip to the next element of the list.
2. When all library directories have been examined without finding an appropriate *TGTFIL*, return "not found".

It is clear that, with this algorithm, no “ambiguous reference” anomaly can occur, since there is at most one file with a given base name in a given directory. This algorithm is hence well suited to analyze systems which contain more than one component with the same name in different directories (or libraries). However, it requires additional effort to set up, since care must be taken to define the appropriate *TGTTYP* elements in the libraries clause in the appropriate order, for each directory of the *SRCTYP* at hand.

For a given *TGTTYP* of components to search, and for each appropriate *SRCTYP*, it is possible to use directed search for some *SRCTYP* directory, and unrestricted search for another directory of the same type. Indeed, duplicate component names cause trouble (anomalies) only when they are actually referenced. However, we advise against such practices, which only makes things confusing. For a given *SRCTYP/TGTTYP* combination, either use unrestricted search on all source directories, or use directed search on all of them.

Note: Directed Search is not yet available in the current version of the cataloger; if you use the `libraries` clause to point to components involved in run-time references, these elements will be simply ignored. Directed Search will be added progressively for selected *SRCTYP/TGTTYP* combinations in the forthcoming versions. Check the release notes.

Repetitive and Incremental Operation

Even with the powerful computing platforms easily available nowadays, processing a complete asset using the Rehosting Workbench remains a computing-intensive, long-running, memory-consuming task.

Oracle Tuxedo Application Rehosting Workbench tools are therefore designed to be easily stopped and restarted. The tools use a make-like mechanism to avoid repeating any work which has already been done. This allows efficient operation in all phases of a migration project.

Initial Processing: Repetitive Operation

In the initial phase, when starting with a completely fresh asset and up to the end of the first conversion-translation-generation cycle of a stable asset, the make-like mechanism is used to allow repetitive operation, as follows:

1. When a tool such as the Cataloger starts, it begins with studying the current state of the asset (source files and target files such as the POB files or the Symtab) and determining what work remains to do to reach a complete and consistent set of results.
2. The tool then undertakes this work, producing more and more result files (or updating the Symtab with new results).

As the volume of processed files grows, the Refine process consumes more and more memory.

3. Regularly, the tool checks whether the available physical memory drops below the threshold set by the minimum-free-ram-percent option in the system description file.
 - If the work to be performed is complete before running out of memory, the process definitely stops.
 - Otherwise, the process stops but restarts immediately, after memory is freed. Going back to step 1 above, there is less work to do, so that the process eventually terminates.

This mode is particularly well suited for tools or commands which operate globally on the whole asset, such as the analyze or catalog commands of the Cataloger. This is the normal mode of operation for the Rehosting Workbench tools and there is nothing specific to choose it.

Changes in the Asset: Incremental Operation

The Cataloger knows the dependencies between the various components and associated result files. For instance, it records which copy files are used in which COBOL programs. Using this information, it is able to react incrementally when some change occurs in the asset. For example, when a component source file is added, modified or removed: the Cataloger determines which result files are affected by this change and re-computes only those files. Again, this is the normal mode of operation for the Rehosting Workbench tools and there is nothing specific to choose it.

Note: Important: Incremental operation is enabled only after the initial processing of the asset is complete. If you perform changes in the asset before the end of the initial cycle, some dependencies may not yet be recorded, in which case the evaluation of the work to re-do will be incorrect and the final results will be inconsistent. It is therefore very important that you let the Cataloger run to completion on the initial asset before you make any change in the asset.

DB2-to-Oracle Convertor

This chapter describes the DB2 objects that are migrated from the source platform (z/OS) and the migration tools that are generated. The conversion is performed in the context of other components translated or generated by the other Oracle Tuxedo Application Rehosting Workbench tools.

The Oracle Tuxedo Application Rehosting Workbench DB2-to-Oracle Convertor uses as a starting-point a coherent set of DB2 objects called a schema, see [Oracle Tuxedo Application Rehosting Workbench Schema](#). Several configuration files need to be set, see [Description of the Configuration Files](#), before launching the conversion process.

The different objects generated are described in [Description of the Output Files](#).

Overview of the DB2-to-Oracle Convertor

Purpose

The purpose of this section is to describe precisely all the features of the Rehosting Workbench DB2-to-Oracle Convertor tools including:

- Inventory of DB2 objects migrated.
- Detailed description of converted Oracle objects on the target platform for each DB2 object.
- Description of the different commands to be used with the DB2-to-Oracle Convertor.
- Description of the data unloading options on the source platform.

- Description of the data loading options on the target platform.

Structure

- [Description of the Input Components](#) including [Description of the Configuration Files](#).
- [Description of the Output Files](#) including the [Generated Objects](#).
- [Detailed Processing](#) including the [Command-line Syntax](#).
- For messages, see [DB2-to-Oracle Converter Messages](#).

See Also

The conversion of data is closely linked to the conversion of COBOL programs, see:

- [COBOL Converter](#)

Oracle Tuxedo Application Rehosting Workbench Schema

For the Rehosting Workbench, a schema should consist of a coherent set of objects (for example there should be no CREATE INDEX for a table that does not exist in the schema).

By default, if the SQL commands of the DB2 DDL are prefixed by a qualifier or an authorization ID, the prefix is used by the Rehosting Workbench as the name of the schema, for example,
`CREATE TABLE <qualifier or authorization ID>.table name.`

The schema name can also be determined by the Rehosting Workbench using the global-options clause of the [System Description File](#).

For example:

Listing 4-1 Schema Name Example

```
system STDB2ORA root ".."
global-options
catalog="..",
sql-schema=<schema name>.
```

Another possibility is to implement this option for each directory where it is necessary, an option that is useful when several schemas are used.

Example:

```
directory "BATCH" type Cobol-Batch files "*.cbl" libraries "COPY". %,
"INCLUDE" options sql-schema=<"schema name">.
```

Environment Variables

Before starting the process of migrating data two environment variables should be set:

- `export TMPPROJECT=/$HOME/tmp`

Indicates the location to store temporary objects generated by the process.

- `export PARAM=/$HOME/param`

Indicates the location where the configuration files required by the process are stored.

Description of the Input Components

File Locations

Location of rdbms.sh

The rdbms.sh tool is located in the directory:

```
$REFINEDIR/convert-data/
```

Location of db-param.cfg File

The db-param.cfg configuration file is located in the directory given in the variable:

```
$PARAM
```

DB2 DDL Converted

The following DB2 objects are migrated to Oracle:

Table 4-1 DB2 Objects to Convert

Object Type	File name	Remark
TABLE	TABLE- <target_table_name>.sql	One file per Table. The file contains table construction, with column name, data type and attribute(s). Constraints, except NULL/NOT NULL attributes, are not written in this file
INDEX	INDEX- <target_table_name>.sql	This file contains all indexes associated with the table <target_table_name>. This file will not be generated if there were no indexes defined on the table <target_table_name> Indexes are: unique or not unique constraint.
CONSTRAINT	CONSTRAINT- <target_table_name>.sql	This file contains all constraints associated with the table <target_table_name>. This file will not be generated if there were no constraints defined on the table <target_table_name> Constraints are: Primary Key, Unique, Check and Foreign key
COMMENT	COMMENT- <target_table_name>.sql	Contains all comments for table and columns. One file per table
VIEW	VIEW-<shema_name>.sql	This file contains all Views created in the source database/schema. In this release, the Select statements are not automatically converted into the target database language.
SEQUENCE	SEQUENCE-<shema_name>.sql	For sequence already created on the source database
SYNONYM	SYNONYMS-<shema_name>.sql	
IDENTITY	IDENTITY- <target_table_name>.sql	In case of IDENTITY, when migrating from DB2 to ORACLE the Rehosting Workbench creates a Sequence and Trigger objects.

Conversion of DB2 Data Types

The following table shows all DB2 data types and their conversion to the Oracle database target

Table 4-2 DB2 to Oracle Data Type Conversion

DB2 z/OS Data type	Oracle Format	Notes
CHAR CHAR (length)	CHAR	CHAR without length becomes CHAR(1)
VARCHAR (length)	VARCHAR2 (length)	
DECIMAL (...)	NUMBER (...)	If no precision, DECIMAL becomes NUMBER(5)
NUMERIC (...)	NUMBER (...)	If no precision, NUMERIC becomes NUMBER(5)
DEC (...)	NUMBER (...)	If no precision, DEC becomes NUMBER(5)
SMALLINT	NUMBER (6)	
INTEGER	NUMBER (11)	
TIMESTAMP	TIMESTAMP	
TIMESTMP	TIMESTAMP	
DATE	DATE	
TIME	DATE	
DOUBLE	FLOAT (53)	
FLOAT (prec)	FLOAT (53)	
REAL	FLOAT (24)	

DB2 Column Property Conversion

The following table shows all DB2 column properties and their conversion to the Oracle database target.

Table 4-3 DB2 Column Property Conversion

DB2 Column Property	Oracle Format	Notes
WITH DEFAULT	DEFAULT <value>	<value> depends on DB2 z/OS data type.
WITH DEFAULT ' ' (with nothing between quotes)	CHAR:... DEFAULT ' ' VARCHAR2 ... DEFAULT ' ' ' '	A zero byte length in DB2 becomes NULL flag on Oracle
WITH DEFAULT ' <value> '	DEFAULT ' <value> '	
NOT NULL	NOT NULL	
IDENTITY	Create a Sequence Create a Trigger	Because the IDENTITY attribute does not exist on Oracle, the Rehosting Workbench replaces the attribute with Sequence and Trigger objects.
FOR SBCS ...	Attribute ignored	

Description of the Configuration Files

This section lists the files and their parameters that can be used to control the migration of a DB2 database to an Oracle database.

POB Files

These files are created during cataloging, for further information see [POB Files for ASTs](#).

DB2 DDL POB File

A POB file is created for each DB2 DDL source file.

For example the SQODCSF0.ddl file contains the source of different DB2 objects. A SQODCSF0.ddl.pob file is created in the \$SOURCE/DDL/pobest directory.

Symtab File

symtab-<schema name>.pob

This file is created during cataloging, it must be up-to-date and present so that DB2-to-Oracle Converter can migrate DB2 objects to Oracle. See [The Cataloger Syntab and Other Miscellaneous Files](#).

sql-system File

`sql-system-<project name>.pob`

This file is created during cataloging, it must be up-to-date and present so that DB2-to-Oracle Converter can migrate DB2 objects to Oracle. See [The Cataloger Syntab and Other Miscellaneous Files](#).

`sql-system-<project name>-Statements.pob`

This file is created during cataloging, it must be up-to-date and present so that DB2-to-Oracle Converter can migrate DB2 objects to Oracle. See [The Cataloger Syntab and Other Miscellaneous Files](#).

system.desc

In addition to the parameters concerning schema, one other parameter should be set in the [System Description File](#).

DBMS-VERSION="8".

Indicates the version of the RDBMS to migrate.

db-param.cfg

This file should be created in the directory indicated by the \$PARAM variable:

`$PARAM/db-param.cfg`

Listing 4-2 db-param.cfg Template

```
#
# This configuration file is used by FILE & RDBMS converter
# Lines beginning by "#" are ignored
# write information in lower case
#
# common parameters for FILE and RDBMS
```

```

#
# source information is written into system descriptor file (DBMS=,
DBMS-VERSION=)
target_rdbms_name:<target_rdbms_name>
target_rdbms_version:<target_rdbms_version>
target_os:<target_os>
#
# specific parameters for RDBMS conversion
rdbms:date_format:<date_format>
rdbms:timestamp_format:<timestamp_format>
rdbms:time_format:<time_format>
rdbms:indexsort:<index_sort_option>
rdbms:indexlang:<index_lang_option>
# rename object files
# the file param/rdbms/rename-objects-<schema>.txt is automatically loaded
by the tool if it exists.

```

Parameters and Syntaxes

Table 4-4 db-param-cfg Parameters

Parameter	Description	Value
General Parameters		
<target_rdbms_name>	Name of target RDBMS	oracle
<target_rdbms_version>	Version of target RDBMS	11
<target_os>	Name of target operating system	unix or linux
Parameters concerning the migration of dates, timestamps and times		

Table 4-4 db-param-cfg Parameters

Parameter	Description	Value
<date_format>	Date (in the format expected by Oracle)	
<time_stamp_format>	Timestamp (in the format expected by Oracle)	
<time_format>	Time (in the format expected by Oracle)	
Index and Sort parameters		
rdbms:indexsort:<index_sort_option>		
rdbms:indexlang:<index_lang_option>		

Date, Time Parameters

These three parameters indicate the date, timestamp and time formats used by z/OS DB2 and stored in DSNZPARM. These parameters impact the reloading operations and the COBOL date and time manipulations.

These parameters are optional and only necessary if the DB2 database contains the DATE, TIME or TIMESTAMP fields. These parameters should be supplied respecting the Oracle formats.

If these parameters are supplied, the UNIX/Linux variables:

- NLS_DATE_FORMAT ,
- NLS_TIMESTAMP_FORMAT ,
- NLS_TIME_FORMAT,

should be set according to the instructions in the ORACLE documentation.

WARNING: A correct setting of these parameters is essential.

Index, Sort Parameters

These parameters are optional and are only necessary if the sort order in certain columns must be kept and is maintained in applications by use of an index. They are therefore to be implemented

only to maintain the ISO functionality of applications between the source and target platforms. For these parameters the values can be:

```
<index_sort_option> EBCDIC, FRENCH, ...  
<index_lang_option> FR, UK, ...
```

When these parameters are used, the SQL CREATE INDEX scripts are generated as shown in the example below where the value of the index parameter is *French*:

Listing 4-3 SQL CREATE INDEX Script Using Index Parameter

```
WHenever SQLERROR Continue;  
  
DROP INDEX MYDB.TAB1_IDX;  
  
WHenever SQLERROR Exit 3;  
  
CREATE INDEX MYDB.TAB1_IDX ON MYDB.TAB1  
(  
    nlsort(MYCOL11, 'nls_sort=FRENCH') ASC  
);
```

The variables:

- NLS_SORT,
- NLS_COMP,
- NLS_LANG

should be harmonized with the values implemented for these two parameters. See the Oracle documentation: Oracle Database Globalization Support Guide.

By default, the tools generated by the DB2 to Oracle migration expect NLS_SORT in binary.

WARNING: A correct setting of these parameters is essential.

File Modifying Generated Components

The generated components may be modified using a project's own scripts. These scripts (sed, awk, perl,...) should be placed in:

```
$PARAM/rdbms/rdbms-modif-source.sh
```

When present, this file will be automatically executed at the end of the generation process. It will be called using the `<schema name>` as an argument.

Renaming File

Oracle Tuxedo Application Rehosting Workbench permits the modification of the different names in the DDL source file (table name, column name).

Renaming rules can be implemented for the following cases:

- When Oracle reserved words are found in the DB2 DDL source file.
- When there is a desire to perform a reengineering of the DDL source file.

Note: If, when executing the Rehosting Workbench, an Oracle reserved word is found in the DDL source, an error is reported and the Rehosting Workbench continues the analysis of the DDL.

Renaming rules should be placed in a file named: `rename-objects-<schema name>.txt`.

This file should be placed in the directory indicated by the `$PARAM/rdbms` variable.

Renaming rules have the following format:

table

```
table;<schema name>;<DB2 table name>;<Oracle table name>
```

column

```
column;<schema name>;<DB2 table name>;<DB2 column name>;<Oracle column name>
```

Note: To apply the modifications to all schema names and/or all table names, place a `*` in the position of `<schema name>` and/or `<DB2 table name>`. See the second example below.

Comments can be added in the form of: `% Text`.

Listing 4-4 Renaming File Example

```
% Modification applied to the AUALPH0T table  
column;AUANPR0U;AUALPH0T;NUM_ALPHA;MW_NUM_ALPHA  
column;AUANPR0U;*;ASC;ASC-1
```

rdbms-template.txt

This file is placed during the installation of the Rehosting Workbench, it contains the templates that perform the generation of the different migration tools. The file is located in:

```
$REFINEDIR/convert-data/default/rdbms/rdbms-templates.txt
```

Listing 4-5 rdbms-template.txt

```
% Unloading all TABLE *****
#VAR:TEMPLATES#/unloading/jcl-unload-DB2-table-SQL.pgm
%
% Loading TABLE *****
%
#VAR:TEMPLATES#/loading/convert-ebcdic-for-oracle.pgm
#VAR:TEMPLATES#/loading/reload-tables-ksh.pgm
#VAR:TEMPLATES#/loading/rdbms-reload-tables-txt.pgm
%
% included file to include into modified-components
#VAR:TEMPLATES#/include-modified-components.pgm
%
% *****
% MANDATORY: used when using -r argument
#VAR:TEMPLATES#/remove-schema-name-ksh.pgm
% MANDATORY: used when using -i argument
#VAR:DEFAULT#/rdbms-move-assignation.pgm
%
```

When required, another version of the `rdbms-template.txt` file can be placed in the `$PARAM/rdbms` directory. The use of an alternative file is signaled during the execution of `rdbms.sh` by the message:

Listing 4-6 Execution Log with Alternative Template File

```
#####
Control of templates

    OK: Use Templates list file from current project:

        File name is /home2/wkb9/param/rdbms/rdbms-templates.txt
#####
```

`rdbms_move_assignment.txt`

This file is placed during the installation of the Rehosting Workbench, it controls the transfer of components generated in the different installation directories. This file indicates the location of each component to copy during the installation phase of `rdbms.sh`, when launched using `rdbms.sh -i`.

The file is located in:

`$REFINEDIR/convert-data/default/rdbms/rdbms-move-assignment.pgm`

This file can be modified following the instructions found at the beginning of the file:

Listing 4-7 `rdbms_move_assignment.txt` Modification Instructions

```
[...]

* @ (c) Metaware:file-move-assignment.pgm. $Revision: 1.2 $

* release_format=2.3

*

* format is:

*   <typ>:<source_directory>:<file_name>:<target_directory>

*

```

```

* typ:
*   O: optional copy: if the <file_name> is missing, it is ignored
*   M: Mandatory copy: abort if <file_name> is missing.
*   E: Execution: execute the mandatory script <file_name>.
*
*   Parameters for script to be executed are:
*
*       basedir:          directory of REFINEDIR/convert-data
*       targetoutputdir:  value of "-i <targetdir>"
*       schema:           schema name
*       target_dir:       value written as 4th parameter in this file.
*
*   d: use this tag to display the word which follows
*
* source_directory:
*
*   T: generated components written in <targetdir>/Templates/<schema>
*   O: components written in <targetdir>/outputs/<schema>
*   S: SQL requests (DDL) generated into <targetdir>/SQL/<schema> directory
*   F: fixed components present in REFINEDIR
*
*   s: used with -s arguments: indicates the target directory for DML
utilities
*
*       (in REFINEDIR/modified-components/) which manipulate all schemas.
*
* file_name: (except for typ:d)
*
*   name of the file in <source_directory>
*
*
* target_directory: (except for typ:d, given at 4th argument for typ:E)
*
*   name of the target directory
*
*   if the 1st character is "/", component is copied using static directory
and not in <td> directory
*

```

[...]

Description of the Output Files

File Locations

Location of Temporary Files

The temporary objects generated by the Rehosting Workbench DB2-to-Oracle Converter are stored in:

```
$TMPPROJECT  
$TMPPROJECT/Template/<schema name>  
$TMPPROJECT/outputs/<schema name>
```

Locations of Log Files

The execution log files are stored in:

- Log generated by the option `-c` or `-C`:
`$TMPPROJECT/outputs/<schema name> rdbms-converter-<schema name>.log`
- Log generated by the option `-g`:
`$TMPPROJECT/outputs mapper-log-<schema name>`

The `$TMPPROJECT` variable is set in `$HOME/tmp`.

Location of Generated Files

The unloading and loading components generated with the `-i $HOME/trf` option are placed in the following locations:

Table 4-5 Component Locations

Location	Contents
<code>\$HOME/trf/unload/rdbms/<schema name></code>	<p>The JCL used for each unloading table are generated for each <i><schema name></i>.</p> <p>These JCL are named:</p> <p><i><table name_8>.jclunload</i></p>
<code>\$HOME/trf/SQL/rdbms/<schema name></code>	<p>Location by <i><schema name></i> of the SQL scripts used to create the Oracle objects.</p> <p>The names of these scripts are described in Table 4-1.</p>
<code>\$HOME/trf/reload/rdbms/<schema name>/src</code>	<p>Location by <i><schema name></i> of the COBOL transcoding programs.</p> <p>These programs are named:</p> <p><i>MOD-<target table name>.cbl</i></p>
<code>\$HOME/trf/reload/rdbms/<schema name>/ctl</code>	<p>Location by <i><schema name></i> of the CTL files used by SQL*LOADER.</p> <p>These files are named:</p> <p><i><target table name>.ctl</i></p>
<code>\$HOME/trf/reload/rdbms/<schema name>/ksh</code>	<p>Location by <i><schema name></i> of the reloading Korn shell scripts.</p> <p>These scripts are named:</p> <p><i>loadrdbms-<target table name>.ksh</i></p>

Notes: *<table name_8>* If the table name is shorter or longer than eight characters, the Rehosting Workbench DB2-to-Oracle Converter attributes an eight-character name to the z/OS JCL as close as possible to the original. The renaming process maintains the uniqueness of each table name.

<target table name> Is the table name in the Oracle database.

Generated Objects

The following sections describe the objects generated during the migration from a DB2 to Oracle database and the directories in which they are placed.

Temporary Files

These files are automatically generated during the first step of the Rehosting Workbench DB2-to-Oracle Converter for use in the second step.

Datamap File

This is a configuration file generated and used by the Rehosting Workbench containing the list of tables to be migrated. If a table name is missing in the Datamap file, but present in the mapper file, it will be ignored during the execution of the Rehosting Workbench.

This file is generated using the `-c` or `-C` options of the `rdbms.sh` command.

File Name

The Datamap file is created with this complete name:

```
<-target-directory parameter>/outputs/<schema name>/Datamap-<schema name>.re
```

Parameters and syntaxes used:

Table 4-6 Datamap File Name Parameters

Parameter	Value
<code><-target-directory parameter></code>	Value of the parameter <code>-target-directory</code> : <code>\$HOME/tmp</code>
<code><schema name></code>	Name of the current schema.

Syntax and Parameters

Listing 4-8 Datamap File

```
%% Rdbms-converter. Rev: <revision>. <compilationDate>.  
<BeginGeneratedDate>  
  
%% tables generation only  
  
data map <schema name>-map system cat::<PROJECT_NAME>  
  
file <schema name>.<source_table_name>
```

organization sequential

Parameter	Value
%%	Comment ignored by the Rehosting Workbench.
<revision>	Oracle Tuxedo Application Rehosting Workbench revision number.
<compilation_date>	Oracle Tuxedo Application Rehosting Workbench compilation date.
<BeginGeneratedDate>	Date and time of the execution.
<schema name>	Name of the current schema
<PROJECT_NAME>	Project name as described into System Description File .
<source_table_name>	Table name on the source database.

Mapper File

This is a configuration file used by the Rehosting Workbench DB2-to-Oracle Convertor in the second step.

It contains all information about tables and re-engineering processes like object renaming.

The file is generated by rdbms.sh using the `-c` or `-C` options.

File Name

The Mapper file is created with this complete name:

```
$TMPPROJECT/outputs/<schema>/mapper-<schema name>.re
```

Table 4-7 Mapper file Name Parameters

Parameter	Value
<schema name>	Name of the current schema.

Generation Sample

For the following DB2 DDL:

```
CREATE TABLE ART1 (  
    REF_ART CHAR(19) NOT NULL,  
    REF_ART_REMPL VARCHAR(400),  
    REGR_PEREMT CHAR(1),  
    DUREE_VALID DATE NOT NULL,  
    LONG_ART INTEGER  
) [...]
```

The Mapper file generated by DB2-to-Oracle Convertor is:

Listing 4-9 Mapper File Example

```
%% Rdbms-converter. Rev: 1.4. 20090101 13:25. 20091010 11:25  
%% tables generation only  
ufas mapper GM  
file GM.ART1 transferred converted  
  
    table name GM-ART1  
    map record TAB-ART1 defined in "#VAR:RECS_SOURCE#/GM/ART1.cpy"  
    source record TAB-ART1 in "#VAR:RECS_SOURCE#/GM/ART1.cpy"  
    logical name ART1  
    include "#VAR:RECS_SOURCE#/GM/ART1.cpy"  
    strategies  
  
        field REF-ART                attributes NULL_DISALLOWED  DATATYPE_CHAR  
    ,field REF-ART-REMP              attributes NULL_ALLOWED      DATATYPE_VARCHAR  
    ,field REF-ART-REMP-LEN          attributes TECHNICAL_FIELD_VARCHAR  
    ,field REF-ART-REMP-IND          attributes TECHNICAL_FIELD_NULL
```

,field REGR-PEREMT	attributes NULL_ALLOWED	DATATYPE_CHAR
,field REGR-PEREMT-IND	attributes TECHNICAL_FIELD_NULL	
,field DUREE-VALID	attributes NULL_DISALLOWED	DATATYPE_DATE
,field LONG-ART	attributes NULL_DISALLOWED	DATATYPE_NUMERIC

In this example #VAR:RECS:SOURCE# indicates \$TMPPROJECT/outputs/schema.

Syntax and Parameters

Listing 4-10 Mapper File

```

%% Rdbms-converter. Rev: <revision>. <compilationDate>.
<BeginGeneratedDate>

%% tables generation only

ufas mapper <schema name>

## For each table ...

file <schema name>.<source_table_name> transferred converted
    table name <target_table_name>
    include "#VAR:RECS_SOURCE#/<schema>/<source_table_name>.cpy"
    map record TAB-<source_table_name>
        defined in "#VAR:RECS_SOURCE#/<schema>/<source_table_name>.cpy"
    source record TAB-<source_table_name>.
        in "#VAR:RECS_SOURCE#/<schema>/<source_table_name>..cpy"
    logical name <table_name_8>

## For each column of the table

[ |,]field <target_column_name>

attributes [NULL_ALLOWED|NULL_DISALLOWED]

        [DATATYPE_VARCHAR|DATATYPE_CHAR|

```

```

DATATYPE_NUMERIC |
DATATYPE_DATE | DATATYPE_TIME | DATATYPE_TIMESTAMP
]

## optional: for each technical field written in the copy file

[,field <target_column_name>-IND  attributes TECHNICAL_FIELD_NULL    ]
[,field <target_column_name>-LEN  attributes TECHNICAL_FIELD_VARCHAR ]

```

Each file directive is used to describe a table, each field directive describes a column of the table (the field directives <target_column_name>-IND and field <target_column_name>-LEN are optionally used to modify certain options in the technical fields added to the unloaded file by DSNTIAUL).

[Table 4-8](#) lists the field directives.

Table 4-8 Field Directive Parameters

Parameter	Value
%%	Comment ignored by the Rehosting Workbench.
<revision>	Oracle Tuxedo Application Rehosting Workbench revision number.
<compilation_date>	Oracle Tuxedo Application Rehosting Workbench compilation date.
<BeginGeneratedDate>	Date and time of the execution.
<schema name>	Name of the current schema
<source_table_name>	Table name on source database
<table_name_8>	Logical name of the table limited to 8 characters. Oracle Tuxedo Application Rehosting Workbench creates a unique name only within a schema.
<target_table_name>	Table name on target platform.

Table 4-8 Field Directive Parameters

Parameter	Value
<source_column_name>	Column name from source database.
<target_column_name>	Column name on target database.

Links to COBOL Copy

As seen in the file clause of the example, the Mapper file is linked to a COBOL copy file. This COBOL copy describes the unloaded data file: it contains columns descriptions and also technical fields. This unloaded file is created by DSNTIAUL utility. It contains column data and null indicator values

For each column, a field name and two values for the attributes are generated:

- The NULL information:
 - NULL_ALLOWED
 - NULL_DISALLOWED

Used when the column accepts the NULL flag or has the NOT NULL attribute respectively.

- Datatype information. It indicates the column data type

For each technical field, DB2-to-Oracle Convertor generates:

- For a NULL field (it means the column has NULL attribute), the COBOL copy contains an additional field. The name of this field is the target column name with an -IND suffix.
- For a VARCHAR field, the COBOL copy contains an additional field placed before the data field. The name of this field is the target column name with -LEN suffix.

COBOL Description

Oracle Tuxedo Application Rehosting Workbench DB2-to-Oracle Convertor needs a description associated with each table, so a first step generates a COBOL copy description.

This copy contains:

- Field descriptions, one field per column.
- Technical fields:

- For a VARCHAR column, a length field is added.
- For a column without a NOT NULL attribute, an indicator field.

This description is equal to the downloaded file.

Copy File Name

The copy file is created with this complete name:

```
<-target-directory parameter>/outputs/<schema
name>/<source_table_name>.cpy
```

Table 4-9 Copy File Name Parameters

Parameter	Value
<-target-directory parameter>	Value of the parameter -target-directory
<schema name>	Name of the current schema.
<source_table_name>	Name of the source table name

Listing 4-11 Copy File: SQL Code

```
CREATE TABLE ART1 (
  REF_ART CHAR(19) NOT NULL,
  REF_ART_REMPL VARCHAR(400),
  REGR_PEREMT CHAR(1),
  DUREE_VALID DATE NOT NULL,
  LONG_ART INTEGER
) [...]
```

For this example of DB2 DDL, the copy file generated by the Rehosting Workbench is:

Listing 4-12 Copy File: Generation Sample

```
* Rdbms-converter. Revision: Rev: 1.4. 20090101 13:25. 20091010 11:25

01 TAB-ART1.

    03 REF-ART                      PIC X(19).

    03 REF-ART-REMP-LEN             PIC S9(4) COMP-5.

    03 REF-ART-REMP                 PIC X(400).

    03 REF-ART-REMP-IND             PIC X.

    03 REGR-PEREMT                  PIC X(1).

    03 REGR-PEREMT-IND              PIC X(1).

    03 DUREE-VALID                  PIC X(10).

    03 LONG-ART                     PIC S9(9) COMP-5.

    03 LONG-ART-IND                 PIC X(1).
```

Copy File Syntax and Parameters

The generated copy files have the following format:

Listing 4-13 Copy File

```
* Rdbms-converter. Revision: <revision> <compilationDate>
<BeginGeneratedDate>

01 TAB-<source_table_name>.

    03 <target_column_name>-LEN PIC S9(4) COMP-5.

    03 <target_column_name> PIC <field_Cobol_format>.

    ## Is the column has NULL attribute

    03 <target_column_name>-IND PIC X.
```

Table 4-10 Copy File Parameters

Parameter	Description
<revision>	Oracle Tuxedo Application Rehosting Workbench revision number.
<compilationDate>	Oracle Tuxedo Application Rehosting Workbench compilation date and time.
<BeginGeneratedDate>	Date and time at the beginning of the process.
<source_table_name>	Table name on source database
<target_column_name>	Column name on target database
<target_column_name> -LEN	If the column has VARCHAR datatype
<target_column_name> -IND	If the column has NULL attribute
<field_Cobol_format>	Picture of the field in COBOL.

Unloading JCL

The JCL used to unload the DB2 tables are generated using the `-g` option of the `rdbms.sh` command. They are then (using the `-i` option) installed in:

```
$HOME/trf/unload/rdbms/<schema name>
```

Each JCL contains two steps and unloads one DB2 table using the z/OS DSNTIAUL utility. The JCL return code is equal to 0 or 4 for a normal termination.

- | | | |
|--------|--------|---|
| Step 1 | DEL | IDCAMS DELETE files (deletion of log, data, syspunch files) |
| Step 2 | UNLOAD | DSNTIAUL of the indicated table |

The JCLs are named: `<table name>.jclunload`

If the table name is shorter or longer than eight characters, the Rehosting Workbench attributes an eight-character name to the z/OS JCL as close as possible to the original. The renaming process maintains the uniqueness of each table name within a schema.

Note: The .jclunload extension should be deleted for execution under z/OS.

The generated JCL may need adapting to specific site constraints including:

- JOB cards: <crdjob>,
- library access paths: <db2_runlib_load_library> ,
- access paths to input and output files: <data>.

Listing 4-14 Unload JCL Example

```
//<crdjob> <cardjob_parameter_1>,'DB2 ODCSF0X1',
//          <cardjob_parameter_2>
//          <cardjob_parameter_3>
//          <cardjob_parameter_4>
//*@ (C) Metaware:jcl-unload-DB2-table-SQL.pgm. $Revision: 1.7.2.1 $
//*****
//* UNLOAD THE RDBMS TABLE:
//*          PJ01DB2.ODCSF0
//* INTO <data>.PJ01DB2.ODCSF0X1.DATA
//*****
//*-----*
//* DELETE LOG, DATA AND SYSPUNCH FILES
//*-----*
//DEL          EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
```

```

DELETE <data>.PJ01DB2.ODCSF0X1.LOG

DELETE <data>.PJ01DB2.ODCSF0X1.DATA

DELETE <data>.PJ01DB2.ODCSF0X1.SYSPUNCH

SET MAXCC=0

/*-----*
/* LAUNCH DSNTIAUL UTILITY
/*-----*

//UNLOAD      EXEC PGM=IKJEFT01,DYNAMNBR=20

//SYSTSPRT DD SYSOUT=*

//SYSTSIN DD *

DSN SYS(DSNM)

RUN PROGRAM(DSNTIAUL) PLAN(DSNTIAUL) PARM('SQL') -
      LIBRARY ('<db2_runlib_load_librairy>')

END

//SYSPRINT DD SPACE=(CYL,(150,50),RLSE),
//          DISP=(,CATLG),
//          UNIT=SYSDA,
//          DSN=<data>.PJ01DB2.ODCSF0X1.LOG
//SYSUDUMP DD SYSOUT=*
/*
//SYSREC00 DD SPACE=(CYL,(150,50),RLSE),
//          DISP=(,CATLG),
//          UNIT=SYSDA,
//          DSN=<data>.PJ01DB2.ODCSF0X1.DATA
//SYSPUNCH DD SPACE=(TRK,(15,15),RLSE),
//          DISP=(NEW,CATLG),DCB=(LRECL=80,RECFM=FB),
//          UNIT=SYSDA,

```

```
//          DSN=<data>.PJ01DB2.ODCSF0X1.SYSPUNCH
//SYSIN      DD *
SELECT
    CUSTIDENT
    , CUSTLNAME
    , CUSTFNAME
    , CUSTADDRS
    , CUSTCITY
    , CUSTSTATE
    , CUSTBDATE
    , CUSTEMAIL
    , CUSTPHONE
FROM PJ01DB2.ODCSF0;
/*
```

COBOL Transcoding Programs

The COBOL transcoding programs are generated using the `-g` option of the `rdbms.sh` command. They are then (using the `-i` option) installed in:

```
$HOME/trf/reload/rdbms/<schema name>/src
```

The programs are named: `MOD_<table name>.cbl`

The programs should be compiled using the Microfocus COBOL or COBOL-IT compilation options documented in [Compiler Options](#).

The compilation of these programs requires the presence of a `CONVERTMW.cpy` copy file adapted to the project.

These files read a file on input and write a sequential file on output with fixed length records. The output file will be read by the `SQL*LOADER` utility.

Listing 4-15 FILE CONTROL Section - for Transcoding Programs

```
SELECT MW-ENTREE

      ASSIGN TO "ENTREE"

      ORGANIZATION IS SEQUENTIAL

      ACCESS IS SEQUENTIAL

      FILE STATUS IS IO-STATUS.


SELECT MW-SORTIE

      ASSIGN TO "SORTIE"

      ORGANIZATION IS RECORD SEQUENTIAL

      ACCESS IS SEQUENTIAL

      FILE STATUS IS IO-STATUS.
```

A record count is written to the output file and is displayed at the end of processing via:

```
DISPLAY "CONVERTING TERMINATED OK".

      DISPLAY "Nb rows reloaded: " D-NB-RECS.

      DISPLAY " " .
```

The validity of the numeric fields is tested to prevent any formatting problems of the z/OS produced file (discrepancy between the DB2 DDL and the unloaded table). If a field expected to be numeric is not, an ABORT is triggered.

The same checks are made on the technical fields, (NULL indicator fields).

Reloading Korn Shell Scripts

The Reloading Korn shell scripts are generated using the `-g` option of the `rdbms.sh` command. They are then (using the `-i` option) installed in:

```
$HOME/trf/reload/rdbms/<schema name>/ksh
```

The scripts are named: `loadrdbms-<table name>.ksh`

They contain three phases:

- t – transcodephase
- l – loadingphase
- c – checkphase

The execution of the scripts produces an execution log in \$MT_LOG/<nom de table>.log

The following variables are set at the beginning of each script:

Listing 4-16 Reloading Script Variables

```
f="@ (c) Metaware:reload-tables-ksh.pgm. $Revision: 1.14 $"
export DD_ENTREE=${DD_ENTREE:-${DATA_SOURCE}/PJ01DB2.ODCSF0X1.DATA}
export DD_SORTIE=${DD_SORTIE:-${DATA_TRANSCODE}/ODCSF0.ascii}
export DD_MVSLOG=${DD_MVSLOG:-${DATA_SOURCE}/PJ01DB2.ODCSF0X1.LOG}
table_name="ODCSF0"
logtab=$MT_LOG/ODCSF0.log
reportfile=${DATA_TRANSCODE}/${table_name}.rpt
[...]
```

To change the file names, set the DD_* variables before calling the script.

Various messages may be generated during the three execution phases of the scripts, these messages are listed in [Oracle Tuxedo Application Rehosting Workbench Messages](#).

On normal end, a return code of 0 is returned.

Transcoding Phase

This step launches the execution of the COBOL transcoding program associated with the Oracle table processed:

```
cobrun MOD_ODCSF0 >>$logtab 2>&1
```

On normal termination the following message is displayed:

```
echo "file ${DD_SORTIE} transcoded"
```

Loading Phase

This step loads the Oracle table using the SQL*LOADER utility:

```
${BIN}/RunSqlLoader.sh $CTL/ODCSF0.ctl $opt >>$logtab 2>&1
```

On normal termination the following message is displayed:

```
echo "Table ${table_name} successfully loaded."
```

Check Phase

This step verifies after the reloading that the reloaded Oracle table contains the same number of records as the equivalent table unloaded from z/OS by the DSNTIAUL utility. If the number of records is different, an error message is produced:

```
if [ "$qteFile" -ne "$qteTranscode" ]
```

If the number of records is equal, this message is produced:

```
echo "Number of rows written in output file is equal to number written  
in the report file: OK"
```

Note: To execute this step, it is necessary to transfer the DSNTIAUL log file to the target environment.

Target DDL

The ORACLE DDL is generated using the `-c` or `-C` option of the `rdbms.sh` command. They are then (using the `-i` option) installed in:

```
$HOME/trf/SQL/rdbms/<schema name>
```

The script naming rules are described in [Table 4-1](#).

TABLE and COLUMNS

Listing 4-17 Oracle Table and Column Generation

```
WHENEVER SQLERROR CONTINUE;  
  
DROP TABLE <schema>.<target_table_name> CASCADE CONSTRAINTS;  
  
WHENEVER SQLERROR EXIT 3;  
  
CREATE TABLE <schema>.<target_table_name>
```

```
(
    <target_column_name> <column_data_type> <attribute(s)....>[, ]
);
```

INDEX

Listing 4-18 Oracle Index Generation

```
WHENEVER SQLERROR CONTINUE;

DROP INDEX <schema>.<target_index_name>;

WHENEVER SQLERROR EXIT 3;

CREATE [UNIQUE] INDEX <schema>.<target_index_name> ON
<schema>.<target_table_name>

(
    [<nls_function>(<target_column_name> [, '<nls_attributes>'])]
    [ASC|DESC][, ]
);
```

The parameters `<nls_function>` and `<nls_attributes>` are optional. If the db-param parameters `rdbms:indexsort` and `rdbms:indexlang` are set in the db-param .cfg file, then the Rehosting Workbench generates these options in the CREATE INDEX command.

Listing 4-19 Oracle Index Generation Without rdbms:indexsort and rdbms:indexlang Parameters

```
WHENEVER SQLERROR CONTINUE;

DROP INDEX MYDB.TAB2_IDX;

WHENEVER SQLERROR EXIT 3;

CREATE INDEX MYDB.TAB2_IDX ON MYDB.TAB2

(
```

```
MYCOL1 ASC,  
MYCOL2 DESC  
);
```

The following samples show an index creation with `rdms:indexsort=french` parameter.

Listing 4-20 Oracle Index Generation With `rdms:indexsort=french` Parameter

```
WHENEVER SQLERROR CONTINUE;  
DROP INDEX MYDB.TAB1_IDX;  
WHENEVER SQLERROR EXIT 3;  
CREATE INDEX MYDB.TAB1_IDX ON MYDB.TAB1  
(  
    nlssort(MYCOL11, 'nls_sort=FRENCH') ASC  
);
```

CONSTRAINT

Listing 4-21 Oracle Constraint Generation

```
WHENEVER SQLERROR CONTINUE;  
ALTER TABLE <schema name>.<target_table_name> DROP CONSTRAINT  
<target_constraint_name>;  
WHENEVER SQLERROR EXIT 3;  
ALTER TABLE <schema name>.<target_table_name> ADD CONSTRAINT  
<target_constraint_name>  
    < the syntax according to the constraint creation is inserted here>  
;
```

COMMENT

Listing 4-22 Oracle Comment Generation

```
CREATE COMMENT ON TABLE <schema name>.<target_table_name>
IS '<comment_table>'
;

CREATE COMMENT ON COLUMN<schema name>
.<target_table_name>.<target_column_name>
IS '<comment_column>'
;
```

Note: The tool accepts only COMMENT on TABLE and COLUMN.

VIEW

Listing 4-23 Oracle View Generation

```
WHenever SQLERROR CONTINUE;
DROP VIEW <target_view_name>;
WHenever SQLERROR EXIT 3;
CREATE VIEW <target_view_name>
AS <the syntax according to the existing view is inserted here>
;
```

In this version the DB2 SELECT statement must be adapted to the Oracle SQL syntax.

SEQUENCE

For a sequence already present in the DB2 database.

Listing 4-24 Oracle Sequence Generation

```
WHenever SQLERROR Continue;  
  
DROP SEQUENCE <schema name>.<target_sequence_name>;  
  
WHenever SQLERROR Exit 3;  
  
CREATE SEQUENCE <schema name> .<target_sequence_name>  
    <the syntax according to the existent sequence is inserted here>  
;
```

SYNONYM

For the synonym of a table, or a synonym of a view:

Listing 4-25 Oracle Synonym Generation

```
WHenever SQLERROR Continue;  
  
DROP SYNONYM <schema name>.<target_synonym_name>;  
  
WHenever SQLERROR Exit 3;  
  
CREATE SYNONYM <schema name>.<target_synonym_name>  
    FOR <schema name>.[<target_table_name> | <target_synonym_name> |  
    <target_view_name>]  
;
```

Identity Engineering

The DB2 column identity is replaced by two Oracle objects:

- Sequence associated with table <target_table_name> .

- Trigger associated with table <target_table_name>.

Listing 4-26 Oracle Sequence and Trigger Generation

```
-- Sequence associated with table <target_table_name>
-- for identity column <target_column_name>
WHENEVER SQLERROR CONTINUE;
DROP SEQUENCE <schema name>.<table_name_26>_SEQ;
WHENEVER SQLERROR EXIT 3;
CREATE SEQUENCE <schema name>.<table_name_26>_SEQ
START WITH 1 INCREMENT BY 1
;
-- Trigger associated with table <target_table_name> for
-- identity column <target_column_name>

CREATE OR REPLACE TRIGGER <schema name>.<table_name_26>_IDY
BEFORE INSERT ON <schema name>.<table_name>
REFERENCING NEW AS NEW FOR EACH ROW
BEGIN
    SELECT <schema name>.<target_table_name>_SEQ.nextval INTO
:NEW.<target_column_name>
    FROM dual;
END;
/
```

After reloading the ORACLE table, the script must be modified to adapt the CREATE SEQUENCE with the MAX value of the column concerned.

Ordered List of Tables File

This file is generated using the `-c` or `-C` option of the `rdbms.sh` command. It is then (using the `-i` option) installed in:

```
$Home/trf/SQL/rdbms/<schema name>
```

It is named `<schema name>.lst`.

This file contains the names of all of the tables in hierarchical sequence (parent table then child tables).

COBOL Conversion Guide File

This file is generated using the `-s` option of the `rdbms.sh` command.

This file is used by the Rehosting Workbench DB2-to-Oracle Converter to rename object names and to modify options in SQL functions. These objects and functions are stored in SQL applicative requests, inside EXEC SQL and END-EXEC verbs.

File Name

The language conversion file is created with this complete name:

```
$PARAM/dynamic-config/rdbms-conv-<schema name>.xml
```

```
$PARAM/dynamic-config/rdbms-conv.txt
```

Where:

<schema name>

Name of the current schema.

Generated Sample

Listing 4-27 Sample COBOL Conversion Guide

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!-- DOCTYPE RDBMS CONVERSION "metaware_rdbms.dtd" -->

<rdbms revision="1.4" compilationdate="20090101 10:00"
      Generateddate="20090608 08:01">

  <comment>
```

```

</comment>

<sourceformat>
    <date>YYYYMMDD</date>
</sourceformat>

<tablelist>
    <table source_name="DESC">
        <target_name>DESC1</target_name>
    </table>

    <table source_name="USER">
        <target_name>USER1</target_name>
    </table>

    <table source_name="MYTB">
        <column source_name="USER">
            <target_name>USER1</target_name>
        </column>
    </table>
</tablelist>

```

SQL*LOADER Control Files

This file is generated using the `-i` option of the `rdbms.sh` command in:

```
$HOME/trf/reload/rdbms/<schema name>/ctl/<target_table_name>.ctl
```

These files are generated for the Sql*Loader Oracle utility. They contain the description of the data file created by the transcoding programs. Data files are read by Sql*Loader and fully loaded into Oracle Tables.

File Name

```
$HOME/trf/reload/rdbms/<schema name>/ctl/<target_table_name>.ctl
```

Where

<schema name>

Name of the current schema.

<target_table_name>

Name of the target table name.

Generated Sample

The following sample is the content of a CTL file (Control file) used by Sql*Loader.

It shows all the Sql*Loader data types and the file structures managed by this Oracle Loader utility:

Listing 4-28 CTL File Example

```
LOAD DATA

  INFILE  'PHAM.ascii' "fix 512"

  BADFILE 'PHAM.bad'

  APPEND

  PRESERVE BLANKS

  INTO TABLE PH0.PHAM

  REENABLE

  (
    MOUV_REF      POSITION(1:19)    CHAR(19) ,
    C01           POSITION(21:30)   CHAR(10) NULLIF (31:31) = "N" ,
    DATE_FAB      POSITION(32:41)   DATE(10) "DD.MM.YYYY" ,
    M_COEFF_TR     POSITION(43:60)   DECIMAL EXTERNAL ,
    PHDESC        POSITION(62:483)  VARCHAR(420)
    DATMOD         POSITION(485:510) TIMESTAMP "YYYY.MM.DD.HH24.MI.SS.FF6" ,
  )
```

DATE and TIMESTAMP formats are replaced by values described in the db-param.cfg file.

DDL Translator Log File

The DDL translator outputs a log file for each schema that it translates. This file contains information about the translation process and describes the major translation actions that were preformed. It also contains any error or translation issue messages.

Below is an annotated example log file.

The header shows translator version information and starting time.

Listing 4-29 DDL Translator Log File — header

```
Rdbms-converter. Rev: 0.0.0.beta. <revision date/time> <build date/time> .  
BeginTime: <starting date/time>
```

The RDBMS Parameters section lists the input parameters for the translation run.

Listing 4-30 DDL Translator Log File — Parameters

```
=====  
RDBMS Parameters  
=====  
System description file (-system-description):  
.../samples/param/system.desc  
Schemas (-ddl): TEST,OTHER  
Schema names file (-ddls-file): None  
Target OS (-target-os): UNIX  
Target RDBMS (-target-rdbms): ORACLE  
Target RDBMS version (-target-rdbms-version): 11g  
Target directory (-target-directory): .../samples/latest/  
Rename objects file (-rename-objects):  
.../samples/param/rename-objects.conf
```

```
REBMS parameters file (-rdbms-parameters):
.../samples/param/rdbms-parameters.conf
Recatalog allowed? (-with-ddl-changes): true
```

The Database Parameters section lists important properties of the source database.

Listing 4-31 DDL Translator Log File — Database Parameters

```
=====
Database Parameters
=====
DATE_FORMAT: DD.MM.YYYY
TIME_FORMAT: HH24.MI.SSXF
TIMESTAMP_FORMAT: YYYY.MM.DD.HH24.MI.SS.FF6
INDEXSORT: FRENCH
```

The Schema Translation section is a transcript of the translation process.

Listing 4-32 DDL Translator Log File — Schema Translation

```
=====
Schema Translation
=====
-----
Beginning translation
Mode: MVS DB2 8 to UNIX ORACLE 11g
Schemas to translate: TEST, OTHER
Schemas to skip:
```

Schemas to not translate:

Translating schema TEST (1/2) ..

setting up pre-translation ..	3 issues
inspecting the original schema ..	0 issues
preparing the schema for translation ..	0 issues
canonicalizing the original schema ..	0 issues
analyzing the original schema ..	0 issues
Alter-Table-Stmt:	1
Base-Table-Def:	2
Comment-On-Def:	3
Index-Def:	1
Sequence-Def:	1
Synonym-Def:	1
View-Def:	2
translating the schema ..	1 issue
outputting the translated schema files ..	0 issues
outputting the language conversion file ..	0 issues
outputting the data mapper file ..	0 issues
outputting the mapper file ..	0 issues
outputting the COBOL copy book files ..	0 issues
outputting the data loader files ..	0 issues
inspecting the final translation ..	0 issues

The DDL Analysis section describes the major translation actions that were performed to change the input schema into the output schema.

Listing 4-33 DDL Translator Log File — DDL Analysis

```
=====
DDL Analysis
=====

table TABLE2
    copy name is TABLE2.cpy
    logical name is TABLE2X2
table TABLE1
    copy name is TABLE1.cpy
    logical name is TABLE1X1
    table TABLE1 has a new target name NEW_TABLE1
    column COL_CHAR, child of table TABLE1, has a new target name NEW_COL_CHAR
```

The Output Files section lists all files that were created during the translation of the schema.

Listing 4-34 DDL Translator Log File — Output Files

```
=====
Output Files
=====

Target directory: ".../samples/latest/"
Output files:
    LOG file:
        outputs/TEST/rdbms-converter-TEST.log
    DDL files:
        SQL/TEST/COMMENT-NEW_TABLE1.sql
        SQL/TEST/CONSTRAINT-NEW_TABLE1.sql
```

```

SQL/TEST/IDENTITY-NEW_TABLE1.sql
SQL/TEST/INDEX-NEW_TABLE1.sql
SQL/TEST/SEQUENCE-SEQUENCE1.sql
SQL/TEST/SYNONYM-TAB1.sql
SQL/TEST/TABLE-NEW_TABLE1.sql
SQL/TEST/TABLE-TABLE2.sql
SQL/TEST/TEST.lst
SQL/TEST/VIEW-VIEW1.sql
SQL/TEST/VIEW-VIEW2.sql
COPYBOOK files:
    outputs/TEST/TABLE1.cpy
    outputs/TEST/TABLE2.cpy
LANGUAGE-CONVERSION file:
    outputs/TEST/rdbms-conv-TEST.xml
DATA-MAP file:
    outputs/TEST/Datamap-TEST.re
MAPPER file:
    outputs/TEST/mapper-TEST.re
SQL*LOADER files:
    outputs/TEST/NEW_TABLE1.ctl
    outputs/TEST/TABLE2.ctl

```

The Translation Issues section summarizes any translation issues that were detected during the translation.

Listing 4-35 DDL Translator Log File — Translation Issues

```
=====
```

Translation Issues

=====

Showing 3 translation issues.

Setup	2 issues
Next-Schema	0 issues
Inspect-Original	0 issues
Setup-Schema-Root	0 issues
Canonicalize-Original	0 issues
Analyze-Original	0 issues
Translate-Ddl	0 issues
Inspect-Translated	0 issues
Output-Ddl	0 issues
Output-Lang-Conv	0 issues
Output-Data-Map	0 issues
Output-Mapper	0 issues
Output-Copy-Books	0 issues
Output-Data-Loaders	0 issues
Inspect-Final	0 issues
Shutdown	0 issues
Total	2 issues

=====

Translation Phase: :SETUP (2 issues).

Original File: unknown

Original Line: unknown

Phase: Setup

RDBMS-0024: Incomplete source information for the system (OS: MVS, DB: None, Version: "None"). Defaulting to (OS: MVS, DB: DB2, Version: "8").

```

-----
Original File: unknown
Original Line: unknown
Phase: Setup

RDBMS-0083: Rename pattern "COLUMN; X; TABLE1; COL_CHAR; NEW_COL_CHAR"
is pre-empted by an earlier pattern and will never be applied.
-----

```

The trailer shows the ending date and time. It also gives a success/failure indication.

Listing 4-36 DDL Translator Log File — Trailer

```

=====
EndTime: <ending date/time>
Status: Failure

```

Execution Reports

`rdbms.sh` creates different execution reports depending on the options chosen. In the following examples the following command is used:

```
rdbms.sh -Cgrmi $HOME/trf PJ01DB2
```

Listing 4-37 Messages Produced When Using the Options -c or -C With `rdbms.sh`

```

#####
##

CONVERSION OF DDLs and CTL files and GENERATION of directive files

CMD : /Qarefine/release/M2_L3_4/scripts/launch rdbms-converter -s
/home2/wkb9/param/system.desc -td /home2/wkb9/tmp -rdbms-parameters

```

```
/home2/wkb9/tmp/config-rdbms-PJ01DB2-param.tmp -ddl PJ01DB2 -target-rdbms
oracle -target-rdbms-version 11 -target-os unix
```

MetaWorld starter

Loading lib: /Qarefine/release/M2_L3_4/Linux64/lib64/localext.so

(funcall BATCH-TRANSLATE-SQL-DDL)

Starting translation at 2010/01/15 11:45:16

Preparing for translation

Loading system description: /home2/wkb9/param/system.desc

Warning! OS clause is absent, assuming OS is IBM

Current OS is IBM-MF

Loading the SQL System:

... Building or Loading SQL-System...

... Loading SQL-System...

Loading /home2/wkb9/source/sql-system-STDB2ORA.pob at 11:45:16... done at
11:45:17

... Loading SQL-System-Statements...

Loading /home2/wkb9/source/sql-system-STDB2ORA-Statements.pob at
11:45:17... done at 11:45:17

... Loading SQL-System-Statements...done: #1<a SOURCE-FILE>

... Building or Loading SQL-System...done: #2<a SQL-SYSTEM>

... 6 elements in 1 schema.

Warning! OS clause is absent, assuming OS is IBM

Beginning translation

Mode: MVS DB2 8 to UNIX ORACLE 11g

Schemas to translate: PJ01DB2

```

Schemas to skip:
Schemas to not translate:
-----
[...]
Ending translation at 2010/01/15 11:45:17

    WARNING: errors still exist but are ignored (Total=5>S+I=5).

    Check /home2/wkb9/tmp/outputs/PJ01DB2/rdbms-converter-PJ01DB2.log log
file

    Process can continue

*-----
Converted DDLs are in /home2/wkb9/tmp/SQL/PJ01DB2 directory
Generated directives files are in /home2/wkb9/tmp/outputs/PJ01DB2 directory
*-----

```

Listing 4-38 Messages Produced When Using the Options -g With rdbms.sh

```

#####
Control of schema PJ01DB2

#####

Control of templates

    Project Templates list file is missing
/home2/wkb9/param/rdbms/rdbms-templates.txt

    OK: Use Default Templates list file

    File name is
/Qarefine/release/M2_L3_4/convert-data/default/rdbms/rdbms-templates.txt
#####
Control of Mapper

```

#####

GENERATION OF PROGRAMS

```
CMD : /Qarefine/release/M2_L3_4/scripts/launch file-converter -s
/home2/wkb9/param/system.desc -mf /home2/wkb9/tmp/mapper-PJ01DB2.re.tmp
-dmf /home2/wkb9/tmp/outputs/PJ01DB2/Datamap-PJ01DB2.re -td
/home2/wkb9/tmp -tmps /home2/wkb9/tmp/rdbms-templates-PJ01DB2.tmp
-target-sgbd oracle11 -target-os unix -varchar2 29 -abort
```

MetaWorld starter

Loading lib: /Qarefine/release/M2_L3_4/Linux64/lib64/localext.so

(funcall LOAD-THE-SYS-AND-APPLY-DMAP-AND-MAPPER)

File-Converter: We are in BATCH mode

Comand line arguments: begining of analyze

recognized argument -s value: /home2/wkb9/param/system.desc

recognized argument -mf value: /home2/wkb9/tmp/mapper-PJ01DB2.re.tmp

recognized argument -dmf value:
/home2/wkb9/tmp/outputs/PJ01DB2/Datamap-PJ01DB2.re

recognized argument -td value: /home2/wkb9/tmp

recognized argument -tmps value:
/home2/wkb9/tmp/rdbms-templates-PJ01DB2.tmp

recognized argument -target-sgbd value: oracle11

recognized argument -target-os value: unix

recognized argument -varchar2 value: 29

recognized argument -abort

End of Analyze

Parsing mapper file /home2/wkb9/tmp/mapper-PJ01DB2.re.tmp ...

Parsing data-map file /home2/wkb9/tmp/outputs/PJ01DB2/Datamap-PJ01DB2.re
...

Parsing system description file /home2/wkb9/param/system.desc ...

Warning! OS clause is absent, assuming OS is IBM

```

Current OS is IBM-MF

Loading /home2/wkb9/source/symtab-STDB2ORA.pob at 11:45:18... done at
11:45:18

... Loading SQL System from POB...

Loading /home2/wkb9/source/sql-system-STDB2ORA.pob at 11:45:18... done at
11:45:18

Build-Symtab-DL1 #1<a SYMTAB-DL1>

    ... Postanalyze-System-RPL...

sym=#2<a SYMTAB>

PostAnalyze-Common #2<a SYMTAB>

    0 classes

    0 classes

    0 classes

    0 classes

    0 classes

    13 classes

Loading /home2/wkb9/source/BATCH/pob/RSSBBB01.cbl.shrec...

Loading /home2/wkb9/source/COPY/pob/ODCSF0.cpy.cdm...

Loading /home2/wkb9/source/COPY/pob/ODCSFU.cpy.cdm...

Point 1 !!

Point 2 !!

Parsing file /home2/wkb9/tmp/outputs/PJ01DB2/ODCSF0.cpy ...

*Parsed 12 lines*

Point 3 !!

Point 4 !!

Point 5 !!

```

```

loading pob file
/Qarefine/release/M2_L3_4/convert-data/templates/rdbms/unloading/jcl-unload-DB2-table-SQL.pgm.pob

Expanding
/Qarefine/release/M2_L3_4/convert-data/templates/rdbms/unloading/jcl-unload-DB2-table-SQL.pgm ...

Writing ODCSF0X1.jclunload

[.}]

Parsing template file
/Qarefine/release/M2_L3_4/convert-data/default/rdbms/rdbms-move-assignment.pgm

Expanding
/Qarefine/release/M2_L3_4/convert-data/default/rdbms/rdbms-move-assignment.pgm ...

Writing rdbms-move-assignment.lst

Rest in peace, Refine...

*-----
Generated components are in /home2/wkb9/tmp/Template/PJ01DB2
*-----

```

Listing 4-39 Messages Produced When Using the Options -m with rdbms.sh

```

#####

FORMATTING COBOL LINES

#####

CHANGE ATTRIBUTE TO KSH or SH scripts

*-----

Components are modified into /home2/wkb9/tmp directory

*-----

```

Listing 4-40 Messages Produced by The -r Options of rdbms.sh

```
#####  
REMOVE SCHEMA INFORMATION IN SPECIFIC SCRIPTS  
Physical_File_is_PJ01DB2.ODCSF0=====
```

Modified <Templates>:loadrdbms-ODCSF0.ksh

Modified <outputs>:ODCSF0.ct1

Modified <SQL>:TABLE-ODCSF0.sql

Modified <SQL>:INDEX-ODCSF0.sql

Modified <SQL>:CONSTRAINT-ODCSF0.sql

IGNORED <SQL>:COMMENT-ODCSF0.sql is missing but is optional

==_all_schema_==

IGNORED <SQL>:VIEW-*.sql is missing but is optional

IGNORED <SQL>:SEQUENCE-*.sql is missing but is optional

IGNORED <SQL>:SYNONYM-*.sql is missing but is optional

IGNORED <SQL>:IDENTITY-*.sql is missing but is optional

=====

Number of modified files: 5

Number of ignored files: 5

*-----

Components are modified: remove schema_name string

*-----

Listing 4-41 Messages Produced by the -i Option of rdbms.sh

```
#####  
INSTALL COMPONENTS INTO SOURCES USING modif-source-rdbms.sh.sh  
=====
```

```

==_PJ01DB2.ODCSF0_==

  Copied    <Templates>:ODCSF0X1.jclunload to
<td>/unload/rdbms/PJ01DB2/ODCSF0X1.jclunload

  Copied    <Templates>:loadrdbms-ODCSF0.ksh to
<td>/reload/rdbms/PJ01DB2/ksh/loadrdbms-ODCSF0.ksh

[...]

Copied    <SQL>:CONSTRAINT-ODCSF0.sql to
<td>/SQL/rdbms/PJ01DB2/CONSTRAINT-ODCSF0.sql

  IGNORED   <SQL>:COMMENT-ODCSF0.sql is missing but is optional
=====

  IGNORED   <SQL>:VIEW-*.sql is missing but is optional
(...)

  Copied    <fixed-components>:CreateReportFromMVS.sh to
<td>/reload/bin/CreateReportFromMVS.sh
=====

Dynamic_configuration

  Copied    <outputs>:rdbms-conv-PJ01DB2.xml to
/home2/wkb9/param/dynamic-config/rdbms-conv-PJ01DB2.xml

=====

Number of copied files:      12

Number of executed scripts:  0

Number of ignored files:     5

#####
*-----
Components are copied into /home2/wkb9/trf directory
*-----

```

Detailed Processing

This section describes the [Command-line Syntax](#) used by the DB2-to-Oracle Converter, the [Process Steps](#) summary and the [Conversion of DB2 Data Types](#).

The processes required on the source and target platforms concern:

- [Configuring the Environments and Installing the Components](#),
- [Unloading Data](#),
- [Transferring the Data](#),
- [Reloading the Data](#),
- [Checking the Transfers](#),

Command-line Syntax

rdbms.sh

Name

`rdbms.sh` - generate DB2 migration components.

Synopsis

```
rdbms.sh [ [-c|-C] [-g] [-m] [-r] [-i <installation directory>] <schema name> ] -s <installation directory> (<schema name>,...) ]
```

Description

`rdbms.sh` generates the Rehosting Workbench components used to migrate z/OS DB2 databases to UNIX/Linux Oracle databases.

Options

Generation Options

-C <schema name>

Triggers the generation in `$TMPPROJECT`, for the schema indicated, of the following components: ORACLE DDL, CTL files of SQL*LOADER, XML file used by the COBOL Converter, configuration file(mapper file and Datamap file).

If an error occurs, the process is aborted.

-c <schema name>

This option has the same result as the -C option, except the process will abort for any error or warning.

-g <schema name>

Triggers the generation in \$TMPPROJECT, for the schema indicated, of the unloading and loading components. This generation depends on the information found in the configuration files.

Modification Options

-m <schema name>

Makes the generated SHELL scripts executable. COBOL programs are adapted to Micro Focus COBOL fixed format. When present, the shell script belonging to a project, (see [File Modifying Generated Components](#)), that modifies the generated sources is executed.

-r <schema name>

Removes the schema name from generated objects (ORACLE DDL, CTL file, KSH). When this option is used, the name of the schema can also be removed from COBOL components by using the option: [remove-sql-qualifier Clause](#) located in the COBOL conversion configuration file used when converting COBOL components.

Installation Option

-i <installation directory> <schema name>

Places the components in the installation directory. This operation uses the information located in the rdbms-move-assignation.pgm file.

Final Option

-s <installation directory> (<schema name 1>, <schema name 2>, ...)

Enables the generation of the COBOL converter configuration file. This file takes all of the unitary XML files of the project.

All these files are created in \$PARAM/dynamic-config

Example

```
rdbms.sh -Cgrmi $HOME/trf PJ01DB2
```

```
rdbms.sh -s $HOME/trf PJ01DB2
```

Unitary Usage Sequence

If the `rdbms.sh` options are used one at a time, they should be used in the following order:

1. => `-c` or `-C`
2. => `-g`
3. => `-m`
4. => `-r`
5. => `-i`
6. => `-s` (should be executed once steps 1 to 5 have been executed for all schemas).

Process Steps

Configuring the Environments and Installing the Components

This section describes the preparation work on the source and target platforms.

Installing the Unloading Components Under z/OS

The components used for the unloading (generated in `$HOME/trf/unload/rdbms`) should be installed on the source z/OS platform (the generated JCL may need adapting to specific site constraints).

Installing the Reloading Components on the Target Platform

The components used for the reloading (generated in `$HOME/trf/reload/rdbms`) should be installed on the target platform.

The following environment variables should be set on the target platform:

Table 4-11 Target Platform Environment Variables

Variable	Value
DATA_SOURCE	The name of the directory containing the unloaded DB2 tables transferred from z/OS to be reloaded into Oracle tables.
BIN	The location of the generic reload and control scripts (<code>\$HOME/trf/reload/bin</code>).

Table 4-11 Target Platform Environment Variables

Variable	Value
TMPPROJECT	The temporary directory.
MT_LOG	Directory to contain execution logs.
CTL	Directory containing the <table name>.ctl files used by the SQL*LOADER (\$HOME/trf/reload/rdbms/<schema name>/ctl).
DATA_TRANSCODE	Temporary directory used by the DB2 binary data transcoding script (contains temporary files in ASCII format).
NLS_LANG	Set according to the instructions in the Oracle documentation: Oracle Database Globalization Support Guide
NLS_SORT	Set according to the instructions in the Oracle documentation: Oracle Database Globalization Support Guide
NLS_COMP	Set according to the instructions in the Oracle documentation: Oracle Database Globalization Support Guide

In addition, the following variable should be set according to the information in the Oracle Tuxedo Application Rehosting Workbench Installation Guide:

- MT_DB_LOGIN.

Installing the MWDB2ORA Package Component on the Target Platform

Oracle Tuxedo Application Rehosting Workbench has to reproduce the same behavior found on the DB2/zOS platform on the Oracle database. In order to handle certain DB2/zOS specific cases, the DB2-to-Oracle Converter includes an Oracle package named MWDB2ORA which contains Oracle functions.

Those functions are used for managing all DATE, TIME and TIMESTAMPS features. They are added to the COBOL programs by the [COBOL Converter SQL Rules](#).

The package enabling the DB2 behavior is located in the directory:

```
REFINEDIR/convert-data/fixed-components/
```

The package is named:

```
MWDB2ORA.plb
```

To activate the package, install it on the target Oracle database:

1. Copy MWDB2ORA.plb onto your target UNIX/Linux platform.
2. Install the package under SQLPLUS:

```
sqlplus $MT_DB_LOGIN <<EOF
start REFINEDIR/convert-data/fixed-components/MWDB2ORA.plb
quit
EOF
```

Listing 4-42 Messages Produced when Installing the MWDB2ORA Package Under SQLPLUS

```
SQL*Plus: Release 11.1.0.6.0 - Production on ...
Copyright (c) 1982, 2007, Oracle. All rights reserved.

Connected to:

Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing
options

SQL>

Package created.

Package body created.

SQL> Disconnected from Oracle Database 11g Enterprise Edition Release
11.1.0.6.0 - 64bit Production

With the Partitioning, OLAP, Data Mining and Real Application Testing
options
```

Unloading Data

To unload each DB2 table, a JCL using the IBM DSNITIAUL utility is executed. The DSNITIAUL utility creates three files:

- a data file,

- a log file,
- a SYSPUNCH file.

These unloading JCLs are named `<table name>.jclunload`

A return code of 0 is sent on normal job end.

If the table name is shorter or longer than eight characters, the Rehosting Workbench attributes an eight-character name to the z/OS JCL as close as possible to the original. The renaming process maintains the uniqueness of each table name.

Transferring the Data

The unloaded data files should be transferred between the source z/OS platform and the target UNIX/Linux platform in binary format using the file transfer tools available at the site (CFT, FTP, ...).

The LOG and SYSPUNCH files should be transferred in text mode.

The files transferred to the target UNIX/Linux platform should be stored in the `$DATA_SOURCE` directory.

Reloading the Data

The scripts enabling the transcoding and reloading of data are generated in the directory:

```
$HOME/trf/reload/rdbms/<schema name>/ksh.
```

The format of the script names is:

```
loadrdbms-<table name>.ksh
```

Each script launches the COBOL program that performs the transcoding and then the SQL*LOADER utility. The CTL files used by SQL*LOADER are named:

```
<table name>.ctl
```

The reloading script uses the SQL*LDR Oracle utility. Because this utility can access to ORACLE server only, this variable should not contain `@<oracle_sid>` string especially for this reloading step.

Transcoding and Reloading Command

Name

`loadrdbms` - transcode and reload data.

Synopsis

`loadrdbms-<table name>.ksh [-t] [-l] [-c: <method>]`

Options

-t

Transcode the file.

-l

Reload the data into Oracle table.

-c dsntiaul

Implement the verification of the transfer (see [Checking the Transfers](#)).

Checking the Transfers

This check uses the following option of the `loadrdbms-<table name>.ksh`

`-c dsntiaul`

This option verifies after the reloading that the reloaded Oracle table contains the same number of records as the equivalent table unloaded from z/OS by the `DSNTIAUL` utility. If the number of records is different, an error message is produced.

File Convertor: Introduction

This chapter introduces the Rehosting Workbench File Convertor used to migrate files from the source platform (z/OS) to Unix/Linux Micro Focus files or to RDBMS tables, it contains common explanations and usages for specific behaviors of those File Convertors.

You can combine the conversion target between files and RDBMS tables, depending on an optional clause in a Configuration file.

The File Convertor documentation is split into four parts:

- [File Convertor: Introduction](#): introduces the file convertors and explains common behavior.
- [File-to-File Converter](#): describes File-to-File conversion.
- [File-to-Oracle Converter](#): describes File-to-Oracle conversion.
- [File-to-Db2/luw \(udb\) Converter](#): describes File to Db2/luw conversion.

This chapter describes the migration tools that are generated. The conversion is performed in the context of other components translated or generated by the other Oracle Tuxedo Application Rehosting Workbench tools.

Several configuration files need to be set, see [List of the Input Components](#), before launching the conversion process.

The different objects generated are described in the target specific sections. Some objects are only generated when migrating VSAM files to Oracle, you can find PCO programs for Oracle, SQB programs for Db2/luw, SQL files, relational module, logical module, utilities, configuration files, unloading JCL and COBOL program conversion, ...

Overview of the File Converter

Purpose

The purpose of this section and the target specific File Converter sections is to describe precisely all the features of the Rehosting Workbench File Converter tools including:

- Inventory of files to migrate.
- Detailed description of converted files and Oracle tables or Db2/luw (udb) tables on the target platform for each file.
- Description of the different commands to be used with the File Converter.
- Description of the data unloading options on the source platform.
- Description of the data loading options on the target platform.

Structure

- [Overview of the File Converter](#).
- [List of the Input Components](#).
- For messages, see [File Converter Messages](#).

See Also

The conversion of data is closely linked to the conversion of COBOL programs, see:

- [COBOL Converter](#)

For information about the specific output components generated see:

- [File-to-File Converter](#)
- [File-to-Oracle Converter](#)
- [File-to-Db2/luw \(udb\) Converter](#)

File Organizations Processed

Note: You cannot generate both Oracle and DB2/Luw target databases at the same time.

z/OS File Organizations

The Oracle Tuxedo Application Rehosting Workbench File Convertor supports different file organizations on the target platform.

[Table 5-1](#) lists the file organizations handled by z/OS.

Table 5-1 z/OS File Organizations

z/OS Source File	Comment
QSAM	Sequential file
VSAM KSDS	Indexed file
VSAM RRDS	Relative file
VSAM ESDS	Sequential file
PDS/PDS2 File Organization	Partitioned DataSet
GDG File Organization	Generation Data Group

File Conversion to File or to RDBMS Table

When migrating files from a z/OS source platform to a target platform, the first question to ask, when VSAM is concerned, is whether to keep a file or migrate the data to an RDBMS table. For example, permanent files to be later used via Oracle or Db2/luw databases or files that needs locking at the record level.

Oracle Tuxedo Application Rehosting Workbench Configuration Name

A configuration name is related to a set of files to be converted. Each set of files can be freely assembled. Each configuration could be related to a different application for example, or a set of files required for tests.

File Descriptions and Managing Files With the Same Structure

For each candidate file for migration, its structure should be described in COBOLCOBOL format. This description is used in a COBOL copy by the Rehosting Workbench COBOL converter, subject to the limitations described in [COBOL Description](#).

Once built, the list of files to migrate can be purged of files with the same structure in order to save work when migrating the files by limiting the number of programs required to transcode and reload data.

Using the purged list of files, a last task consists of building the files:

- Datamap-<datamap name>.re
- mapper-<mapper name>.re

COBOL Description

A COBOL description is related to each file and considered as the representative COBOL description used within the application programs. This description can be a complex COBOL structure using all COBOL data types, including the OCCURS and REDEFINES notions.

This COBOL description will often be more developed than the COBOL file description (FD). For example, an FD field can be described as a PIC X(364) but really contain a three times defined area including, in one case a COMP-3 based numerals table, and in another case a complex description of several characters/digits fields etc.

It is this developed COBOL description which describes the application reality and therefore is used as a base to migrate a specific physical file.

The quality of the file processing execution depends on the quality of this COBOL description. From this point, the COBOL description is not separable from the file and when referring to the file concerned, we mean both the file and its representative COBOL description. The description must be provided in COBOL format, in a file with the following name:

<COPY name>.cpy

Note: If a copy book on the source platform provides a detailed description of the file, the file can be directly used and declared in the Rehosting Workbench.

COBOL Description Format

The format of the COBOL description must conform to the following rules:

- Only one level 01.
- The word FILLER is not allowed.
- Fields names must be unique.
- Some words are reserved, a list is supplied in the Appendix of the Rehosting Workbench Reference Guide.
- The description should begin in column 1 without any preceding COBOL sequence numbers.
- Comments may be inserted by placing an * in column 1.
- Field level numbers can start from column 2.

COBOL Description and Related Discrimination Rules

Within a COBOL description there are several different ways to describe the same area, which means to store objects with different structures and descriptions at the same place.

As the same zone can contain objects with different descriptions, to be able to read the file, we need a mechanism to determine the description to use in order to interpret correctly this data area.

We need a rule which, according to some criteria, generally the content of one or more fields of the record, will enable us to determine (discriminate) the description to use for reading the re-defined area.

In the Rehosting Workbench this rule is called a discrimination rule.

Any redefinition inside a COBOL description lacking discrimination rules presents a major risk during the file transcoding. Therefore, any non-equivalent redefined field requests a discrimination rule. On the other hand, any equivalent redefinition (called technical redefinition) must be subject to a cleansing within the COBOL description (see the example below [COBOL Description Format](#)).

The discrimination rules must be presented per file and highlight the differences and discriminated areas. Regarding the files, it is impossible to reference a field external to the file description.

The discrimination rules are provided in the mapper file. The syntax is described in chapter [Mapper File](#) of this document.

List of the Input Components

The File Converter needs input components to generate the migration components to be used on the source and target platforms. The Input Components required are:

- Configuration file: `db-param.cfg`.
- Script modifying the generated components: `file-modif-source.sh`.
- List of templates: `file-template.txt` or `file-template-db2luw.txt`.
- Transfer of components generated: `file-move-assignation.pgm` or `file-move-assignation-db2luw.pgm`.
- Configuration file: `Datamap-<configuration name>.re`.
- Configuration file: `mapper-<configuration name>.re`.
- COBOL description.
- Internal POB Files.

The two configuration files (`mapper` and `datamap`) are described in this section. The others are described in detail for each target output:

- [File-to-File Converter](#)
- [File-to-Oracle Converter](#)
- [File-to-Db2/luw \(udb\) Converter](#)

Datamap File

This is a configuration file used by the Rehosting Workbench file converter to add or modify information on the physical files of a system.

Each ZOS file to be migrated must be listed in this file; the file only contains the list of files to be migrated.

The Datamap file must be created in the directory: `$PARAM/file` with the complete name:

`Datamap-<configuration name>.re`

Where `<configuration name>` is the name of the current configuration used.

Datamap Syntax and Parameters

Listing 5-1 Datamap File

```
data map <configuration name>-map system cat::<project name>
file <physical file name>
    organization <organization>
    [is-gdg limit <p> [scratch/noscratch] [empty/noempty]
    [keys offset <n> bytes length <m> bytes primary]
    [relkey size <m> bytes]
```

Parameter	Value
<configuration name>	Name of the configuration to process.
<project name>	Project name as described in the System Description File .
<physical file name>	z/OS physical file name.
<organization>	File organization: Indexed, Sequential or Relative.
is-gdg limit <p> [scratch/noscratch] [empty/noempty]	<ul style="list-style-type: none">• p parameter value is used to specify the total number of generations that the GDG may contain.• scratch/noscratch parameters are mutually exclusive. Scratch parameter specifies that whenever an entry of the GDG is removed from the index, it should be deleted physically and uncataloged. Noscratch parameter specifies that whenever an entry of the GDG is removed from the index, it should be uncataloged but not physically deleted.• empty/noempty parameters are mutually exclusive. Empty specifies that all existing generations of the GDG are to be uncataloged whenever the generations of GDG reaches the maximum limit . Noempty specifies that only the oldest generation of the GDG is to be uncataloged if the limit is reached.

Parameter	Value
keys clause ...<n> ... <m> ...	For indexed files, this clause is used to describe the key; where <n> is the start position and <m> is the length of the key.
relkey clause ...<m>	For relative files, this clause is used to describe the key, where <m> is the length of the key.
% text	Comment ignored by the Rehosting Workbench.

Listing 5-2 Datamap Example

```
data map STFILEORA-map system cat::STFILEORA
%% Datamap File PJ01AAA.SS.QSAM.CUSTOMER
file PJ01AAA.SS.QSAM.CUSTOMER
organization Sequential
%% Datamap File PJ01AAA.SS.VSAM.CUSTOMER
file PJ01AAA.SS.VSAM.CUSTOMER
    organization Indexed
    keys offset 1 bytes length 6 bytes primary
```

Mapper File

This is a configuration file used by the Rehosting Workbench File Convertor to associate each file to migrate with:

- A description,
- Discrimination rules (when necessary)
- Reengineering options (as described in the following sections).

Each z/OS file listed in the [Datamap File](#), must be described in the mapper file.

Mapping File Clause

Mapping files consists in choosing, for each physical file to be treated, the associated COBOL description and discrimination rules.

Listing 5-3 Mapper File Clause Structure

```
file <Physical file name>
    [converted] [transferred]
    table name      <Table Name>
    include         <"path/Copy name">
    map record      <record name> defined in <"path/Copy name">
    source record   <record name> defined in <"path/Copy name">
    logical name    <logical file name>
    converter name  <converter name>
    [attributes     <attribute clause>]

    [mapping strategies clauses]
```

Table 5-2 Mapper File Parameters

file <physical filename>	ZOS physical file name, Name used in the Datamap file.
converted	Indicates file is to be converted to an RDBMS table or via an access function (converted clause can be combined with transferred clause)
transferred	Indicates that the file is to be loaded and reloaded (can be combined with converted).
table name	RDBMS table name.

Table 5-2 Mapper File Parameters

<code>include "<path/COPY name>"</code>	Access path and name of the descriptive copy of the file to migrate.
<code>map record <record name> defined in <"path/COPY name"></code>	<ul style="list-style-type: none"> • record name: corresponds to the level 01 field name of the copy description. • path/COPY name: corresponds to the access path and name of the descriptive copy of the file to migrate.
<code>source record <record names> defined in <"path/COPY name"></code>	<ul style="list-style-type: none"> • record name: corresponds to the level 01 field name of the copy description of the file to migrate. • path/COPY name: corresponds to the access path and name of the descriptive copy of the file to migrate.
<code>Logical name <logical file name></code>	The Logical file name is chosen by the user, maximum eight characters. This name is used for naming the objects (COBOL, JCL) created by the different tools in the Rehosting Workbench.
<code>Converter name <program name></code>	Same name and use as logical file name.
<code>attributes <attribute clause></code>	<p>This optional clause has two attributes that can be used:</p> <ul style="list-style-type: none"> • LOGICAL_MODULE_IN_ADDITION • LOGICAL_MODULE_ONLY <p>Their action is described in the next table.</p>

Table 5-3 Mapper File Attributes

<code>attributes <attribute clause></code>	Role
Attribute clause absent	<p>To be used when the target file is an RDBMS table.</p> <p>In this case some access functions and Korn shell utilities are generated.</p>

Table 5-3 Mapper File Attributes

attributes <attribute clause>	Role
LOGICAL_MODULE_IN_ADDITION	To be used when the target file is an RDBMS table. In this case some access functions, logical access functions and Korn shell utilities are generated.
LOGICAL_MODULE_ONLY	This clause can be used when the target file is an RDBMS table or a MicroFocus file. In this case only the ASG_<logical file name> access function is generated. This access function can be called by CICS Oracle Tuxedo Application Runtime

Note: For access functions, see [Access Functions and Utility Programs in File-to-Oracle Converter](#) or [Access Functions and Utility Programs in File-to-Db2/luw \(udb\) Converter](#).

Listing 5-4 Mapper File Example

```
ufas mapper STFILEORA
file PJ01AAA.SS.VSAM.CUSTOMER converted transferred
    table name CUSTOMER
    include "COPY/ODCSF0B.cpy"
    map record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"
    source record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"
    logical name ODCSF0B
    converter name ODCSF0B
    attributes LOGICAL_MODULE_IN_ADDITION
```

In this example the mapper file is named `STFILEORA`. The file processes only one file named `PJ01AAA.SS.VSAM.CUSTOMER` that is migrated to an Oracle RDBMS table using the `convert` option. The `ODCSF0B.cpy` copy file used to describe the file is one of the source copy files.

Choice of Oracle or Db2/luw (udb) is made in the `db-param.cfg` configuration file.

COBOL Description

Oracle Tuxedo Application Rehosting Workbench File Convertor needs a description associated with each table, so a first step generates a COBOL copy description.

Once the COBOL description files have been prepared, the copy files described in the `mapper-<configuration name>.re` file should be placed in the `$PARAM/file/recs-source` directory.

If you use a COBOL copy book from the source platform to describe a file (see [COBOL Description](#)), then it is the location of the copy book that is directly used.

POB Files

These files are created during cataloging, for further information see [POB Files for ASTs](#).

Symtab File

`symtab-<schema name>.pob`

This file is created during cataloging, it must be up-to-date and present so that File Convertor can migrate DB2 objects to Oracle. See [The Cataloger Symtab and Other Miscellaneous Files](#).

File-to-File Converter

This chapter describes the Rehosting Workbench File-to-File Converter used to migrate files from the source platform (z/OS) to Unix/Linux Micro Focus files and describes the migration tools that are generated. The conversion is performed in the context of other components translated or generated by the other Oracle Tuxedo Application Rehosting Workbench tools.

Several configuration files need to be set, see [Description of the Configuration Files](#), before launching the conversion process.

This chapter is a continuation of the [File Converter: Introduction](#) section, and includes several links, for example, the different objects generated are described in [List of the Input Components](#).

Overview of the File-to-File Converter

Purpose

The purpose of this section is to describe precisely all the features of the Rehosting Workbench File-to-File Converter tools including:

- Inventory of files to migrate.
- Detailed description of converted files on the target platform for each file.
- Description of the different commands to be used with the File-to-File Converter.
- Description of the data unloading options on the source platform.
- Description of the data loading options on the target platform.

Structure

- [Overview of the File-to-File Converter](#).
- [Description of the Input Components](#) including [Description of the Configuration Files](#).
- [Description of the Output Files](#) including the [Generated Objects](#).
- [Detailed Processing](#) including the [Command-Line Syntax](#).
- For messages, see [File Converter Messages](#).

See Also

The conversion of data is closely linked to the conversion of COBOL programs, see:

- [COBOL Converter](#)

The previous chapter explains all common usages:

- [File Converter: Introduction](#)

File Organizations Processed

Keeping z/OS File Organization on the Target Platform

The Oracle Tuxedo Application Rehosting Workbench File-to-File Converter is used for those files that keep their source platform format (sequential, relative or indexed files) on the target platform. On the target platform, these files use a Micro Focus COBOL file organization equivalent to the one on the source platform.

The following table lists the file organizations handled by z/OS and indicates the organization proposed on the target platform

Table 6-1 z/OS to UNIX File Organizations

z/OS source file	UNIX ISAM target file
QSAM	Line sequential ISAM
VSAM KSDS	Indexed ISAM

Table 6-1 z/OS to UNIX File Organizations

z/OS source file	UNIX ISAM target file
VSAM RRDS	Relative ISAM
VSAM ESDS	Line sequential ISAM

PDS File Organization

Files that are part of a PDS are identified as such by their physical file name, for example:
`METAW00.NIV1.ESSAI(FIC).`

An unloading JCL adapted to PDS is generated in this case. The source and target file organizations as indicated in the above table are applied.

GDG File Organization

Generation Data Group (GDG) files are handled specially by the unloading and reloading components in order to maintain their specificity (number of GDG archives to unload and reload). They are subsequently managed as generation files by OracleTuxedo Application Runtime Batch (see the Oracle Tuxedo Application Runtime Batch Reference Guide). On the target platform these files have a LINE SEQUENTIAL organization.

Oracle Tuxedo Application Rehosting Workbench Configuration Name

A configuration name is related to a set of files to be converted. Each set of files can be freely assembled. Each configuration could be related to a different application for example, or a set of files required for tests. The set of files can contain both files or RDBMS table targets.

Environment Variables

Before starting the process of migrating data two environment variables should be set:

- `export TMPPROJECT=/ $HOME /tmp`

Indicates the location to store temporary objects generated by the process.

- `export PARAM=/ $HOME /param`

Indicates the location where the configuration files required by the process are stored.

Description of the Input Components

File Locations

Location of file.sh

The file.sh tool is located in the directory:

```
$REFINEDIR/convert-data/
```

Location of db-param.cfg File

The db-param.cfg configuration file is located in the directory given in the variable:

```
$PARAM
```

Description of the Configuration Files

This section lists the files and their parameters that can be used to control the migration of z/OS files to UNIX/Linux files.

db-param.cfg

This file should be created in the directory indicated by the \$PARAM directory:

```
$PARAM/db-param.cfg
```

Listing 6-1 db-param.cfg Template

```
#
# This configuration file is used by FILE & RDBMS converter
# Lines beginning by "#" are ignored
# write information in lower case
#
# common parameters for FILE and RDBMS
#
# source information is written into system descriptor file (DBMS=,
DBMS-VERSION=)
```

```

target_rdbms_name:<target_rdbms_name>
target_rdbms_version:<target_rdbms_version>
target_os:<target_os>
#
# specific parameters for FILE to RDBMS conversion
file:char_limit_until_varchar:<char_limit>

```

Parameters and Syntaxes

Table 6-2 db-param.cfg Parameters

Parameter	Description	Value
General Parameters		
<target_rdbms_name>	Name of target RDBMS	oracle
<target_rdbms_version>	Version of target RDBMS	11
<target_os>	Name of target operating system	unix or linux
Specific file-to-oracle conversion parameters		
<char_limit>	Not applied	

Note: This section describes the File Converter tool. A configuration file can be used to migrate z/OS files to UNIX/Linux files or RDBMS tables.

File Modifying Generated Components

The generated components may be modified using a project's own scripts. These scripts (sed, awk, perl,...) should be placed in:

```
$PARAM/file/file-modif-source.sh
```

When present, this file will be automatically executed at the end of the generation process. It will be called using the <configuration name> as an argument.

file-template.txt

This file is put in place during the installation of the Rehosting Workbench, it contains the templates that perform the generation of the different migration tools. The file is located in:

```
$REFINEDIR/convert-data/default/file/file-templates.txt
```

Listing 6-2 file-template.txt

```
% Unloading all File *****
% All SAM file were transfered using FTP/Binary
% VSAM unloaded step:
#VAR:TEMPLATES#/unloading/jcl-unload-MVS-REPRO.pgm
%
% To create a specific template, copy this template into :
% -- #VAR:PARAM#/file/specific-templates/unloading/jcl-unload-customer.pgm
%
% Loading *****
#VAR:TEMPLATES#/loading/file-reload-files-txt.pgm
% Loading File to File *****
#VAR:TEMPLATES#/loading/unix-file/reload-files-ksh.pgm
#VAR:TEMPLATES#/loading/unix-file/reload-mono-rec.pgm
% Loading File to Oracle *****
#VAR:TEMPLATES#/loading/unix-oracle/load-tables-ksh.pgm
#VAR:TEMPLATES#/loading/unix-oracle/rel-mono-rec.pgm
#VAR:TEMPLATES#/dml/clean-tables-ksh.pgm
#VAR:TEMPLATES#/dml/drop-tables-ksh.pgm
#VAR:TEMPLATES#/dml/create-tables-ksh.pgm
#VAR:TEMPLATES#/dml/ifempty-tables-ksh.pgm
#VAR:TEMPLATES#/dml/ifexist-tables-ksh.pgm
```

```

%

% Generate Logical & Relational Module *****

#VAR:TEMPLATES#/dml/module/open-multi-assign-free.pgm
#VAR:TEMPLATES#/dml/module/open-mono-rec-idx-perf.pgm
#VAR:TEMPLATES#/dml/module/open-mono-rec-sequential.pgm
#VAR:TEMPLATES#/dml/module/open-mono-rec-relative.pgm

%

% and utilities *****

#VAR:TEMPLATES#/dml/module/decharge-mono-rec.pgm
#VAR:TEMPLATES#/dml/module/recharge-table.pgm
#VAR:TEMPLATES#/dml/module/close-all-files.pgm
#VAR:TEMPLATES#/dml/module/init-all-files.pgm

%

% configuration file for translation and runtime *****

#VAR:TEMPLATES#/dml/generate-config-wb-translator-jcl.pgm
#VAR:TEMPLATES#/dml/generate-rdb-txt.pgm

%

% included file to include into modified-components

#VAR:TEMPLATES#/dml/include-modified-components.pgm

%

% *****

% MANDATORY

% : used just after the generation

#VAR:TEMPLATES#/dml/generate-post-process.pgm

% : used when using -i arguments

#VAR:DEFAULT#/file-move-assignation.pgm

```

Note: This file contains both File-to-File and File-to-Oracle migration parameters.

When required, another version of the `file-template.txt` file can be placed in the `$PARAM/file` directory. The use of an alternative file is signaled during the execution of `file.sh` by the message:

Listing 6-3 Execution log with Alternative Template File

```
#####
Control of templates

    OK: Use Templates list file from current project:

        File name is /home2/wkb9/param/file/file-templates.txt
#####
```

file-move-assignation.pgm

This file is placed during the installation of the Rehosting Workbench, it controls the transfer of components generated in the different installation directories. This file indicates the location of each component to copy during the installation phase of `file.sh`, when launched using `file.sh -i`.

The file is located in:

`$REFINEDIR/convert-data/default/file/file-move-assignation.pgm`

This file can be modified following the instructions found at the beginning of the file:

Listing 6-4 file-move-assignation.pgm Modification Instructions

```
[...]

* @ (c) Metaware:file-move-assignation.pgm. $Revision: 1.2 $

* release_format=2.4

*

* format is:

*   <typ>:<source_directory>:<file_name>:<target_directory>
```

```

*
* typ:
*   O: optional copy: if the <file_name> is missing, it is ignored
*   M: Mandatory copy: abort if <file_name> is missing.
*   E: Execution: execute the mandatory script <file_name>.
*
*   Parameters for script to be executed are:
*
*       basedir:          directory of REFINEDIR/convert-data
*       targetoutputdir:  value of "-i <targetdir>"
*       schema:           schema name
*       target_dir:       value written as 4th parameter in this file.
*
*   d: use this tag to display the word which follows
*
* source_directory:
*
*   T: generated components written in <targetdir>/Templates/<schema>
*   O: components written in <targetdir>/outputs/<schema>
*   S: SQL requests (DDL) generated into <targetdir>/SQL/<schema> directory
*   F: fixed components present in REFINEDIR
*
*   s: used with -s arguments: indicates the target directory for DML
utilities
*
*       (in REFINEDIR/modified-components/) which manipulate all schemas.
*
* file_name: (except for typ:d)
*
*   name of the file in <source_directory>
*
* target_directory: (except for typ:d, given at 4th argument for typ:E)
*
*   name of the target directory
*
*   If the 1st character is "/", component is copied using static directory
*
*   and not in <td> directory

```

```

*      If the 1st character is "!", target directory contains both directory
and
*      target file name.
*
[...]
```

Note: This file contains both File-to-File and File-to-Oracle migration parameters.

Datamap File

This is a configuration file used by the Rehosting Workbench file converter to add or modify information on the physical files of a system.

See [File Converter: Introduction: Datamap File](#) .

Mapper File

This is a configuration file used by the Rehosting Workbench File-to-File Converter to associate each file to migrate.

See [File Converter: Introduction: Mapper File](#).

Note: In the mapper file, the `converted` clause is associated with the RDBMS table target only. Do not use this clause in the File-to-File Converter.

Table 6-3 Mapper File Specific Parameters for the File-to-File Converter

<code>file <physical filename></code>	ZOS physical file name, Name used in the Datamap file.
<code>converted</code>	<p>Used with the attributes <code>LOGICAL_MODULE_ONLY</code> clause, it indicates this file is kept as a MicroFocus file. It is accessed with a logical access COBOL function by Oracle Tuxedo Application Runtime CICS.</p> <p>Without the <code>attributes</code> clause above, it indicates that this file is to be converted to an RDBMS table. You have to ignore this clause to indicate that the file is to be converted to a file.</p> <p>The <code>converted</code> clause can be combined with the <code>transferred</code> clause.</p>

Table 6-3 Mapper File Specific Parameters for the File-to-File Converter

<code>transferred</code>	Indicates that the file is to be loaded and reloaded (can be combined with the converted clause).
<code>include "<path/COPY name>"</code>	Access path and name of the descriptive copy of the file to migrate.
<code>map record <record name> defined in <"path/COPY name"></code>	<ul style="list-style-type: none">• record name: corresponds to the level 01 field name of the copy description.• path/COPY name: corresponds to the access path and name of the descriptive copy of the file to migrate.
<code>source record <record names> defined in <"path/COPY name"></code>	<ul style="list-style-type: none">• record name: corresponds to the level 01 field name of the copy description of the file to migrate.• path/COPY name: corresponds to the access path and name of the descriptive copy of the file to migrate.
<code>Logical name <logical file name></code>	The Logical file name is chosen by the user, maximum eight characters. This name is used for naming the objects (COBOL, JCL) created by the different tools in the Rehosting Workbench.
<code>Converter name <program name></code>	Same name and use as logical file name.
<code>attributes <attribute clause></code>	<p>In File-to-File Converter, this optional clause can be used to allow generation of a logical module access function. Attribute clause can be:</p> <ul style="list-style-type: none">• LOGICAL_MODULE_ONLY <p>In this case only the ASG_<logical file name> access function is generated</p>

Discrimination Rules

A discrimination rule must be set on the redefined field; it describes the code to determine which description of the REDEFINES to use and when.

```
[field <field_name>]
[...]
```

```
rule if <condition> then Field_Name_x
[elseif <condition> then field_Name_y]
[else Field_Name_z]
```

Discrimination Rules Syntax and Parameters

Table 6-4 Discrimination Rules

Syntax	Description
Field_Name_{X,Y,Z}	This is the field that will be used when the associated condition is validated; this field is one of the redefined fields.
Condition	<p>Is a conditional expression composed with field name, operators and COBOL constants.</p> <ul style="list-style-type: none">• Logical operators are: not, and, or• Comparison operators are: = <> < >• Specific operators are: is numeric, is all SPACE• Following COBOL constants may be used: spaces, zeros, high-value, low-value <p>Note: These conditions can be parenthesized.</p>

Discrimination Rules Examples

In the following example the fields `DPODP-DMDCHQ`, `DPONO-PRDTIV`, `DP5CP-VALZONNUM` are redefined.

Listing 6-5 Discrimination Rule COBOL Description

```
01 ZART1.

  05 DPODP PIC X(20).

  05 DPODP-RDCRPHY PIC 9.

  05 DPODP-DMDCHQ PIC X(6).

  05 DPODP-REMCHQ REDEFINES DPODP-DMDCHQ.

    10 DPODP-REMCHQ1 PIC 999.

    10 DPODP-REMCHQ2 PIC 999.

  05 DPODP-VIREXT REDEFINES DPODP-DMDCHQ.

    10 DPODP-VIREXT1 PIC S9(11) COMP-3.

  05 DPONO-NPDT PIC X(5).
```

```

05 DPONO-PRDTIV PIC 9(8)V99.
05 DPONO-PRDPS REDEFINES DPONO-PRDTIV PIC X(10).
05 DP5CP-VALZONNUM PIC 9(6).
05 DP5CP-VALZON REDEFINES DP5CP-VALZONNUM PIC X(6).

```

The following discrimination rules are applied:

Listing 6-6 Discrimination Rules

```

      field DPODP-DMDCHQ
rule if      DPODP-RDCRPHY = 1 then DPODP-DMDCHQ
      elseif DPODP-RDCRPHY = 2 then DPODP-REMCHQ
      elseif DPODP-RDCRPHY = 3 then DPODP-VIREXT
      else DPODP-DMDCHQ,
field DPONO-PRDTIV
rule  if      DPONO-NPDT (1:2) = "01"  then DPONO-PRDTIV
      elseif DPONO-NPDT (1:2) = "02"  then DPONO-PRDPS,
field DP5CP-VALZONNUM
rule if      DPODP-RDCRPHY is numeric then DP5CP-VALZONNUM
      else DP5CP-VALZON

```

The first rule is to test the value of the numeric field `DPODP-RDCRPHY`.

The second rule tests the first two characters of an alphanumeric field `DPONO-NPDT`. Only the values 01 and 02 are allowed.

The third rule tests whether the field `DPODP-RDCRPHY` is numeric.

COBOL Description

Oracle Tuxedo Application Rehosting Workbench File-to-File Converter needs a description associated with each File, so a first step consists in preparing a COBOL copy description.

Once the COBOL description files have been prepared, the copy files described in the `mapper-<configuration name>.re` file should be placed in the `$PARAM/file/recs-source` directory.

If you use a COBOL copy book from the source platform to describe a file (see [COBOL Description](#)), then it is the location of the copy book that is directly used.

Description of the Output Files

File Locations

Location of Temporary Files

The temporary objects generated by the Rehosting Workbench File-to-File Converter are stored in:

```
$TMPPROJECT
$TMPPROJECT/Template/<configuration name>
$TMPPROJECT/outputs/<configuration name>
```

Note: The `$TMPPROJECT` variable is set to: `$HOME/tmp`

Location of Log Files

The execution log files are stored in:

- Log generated by the option `-g`:
`$TMPPROJECT/outputs mapper-log-<configuration name>`

Location of Generated Files

The unloading and loading components generated with the `-i $HOME/trf` option are placed in the following locations:

Table 6-5 Component Locations

Location	Contents
<code>\$HOME/trf/unload/file/<configuration name></code>	<p>The JCL used for each unloading file are generated for each <code><configuration name></code>.</p> <p>These JCL are named:</p> <p><code><file name>.jclunload</code></p>
<code>\$HOME/trf/reload/file/<configuration name></code>	<p>Location by <code><configuration name></code> of the COBOL programs and KSH used for each loading.</p> <p>For a file to file migration, the programs and KSH are named:</p> <p><code>RELFILE-<target file name>.cbl</code></p> <p><code>loadfile-<target file name>.ksh</code></p> <p>For a Generation Data Group file migration, the programs and KSH are named:</p> <p><code>RELFILE-<target file name>.cbl</code></p> <p><code>loadgdg-<target file name>.ksh</code></p> <p><code>loadgds-<target file name>.ksh</code></p>
<code>\$HOME/trf/DML</code>	<p>If you used the <code>attributes</code> clause in the mapper file, an access function will be generated:</p> <p><code>ASG_<target file name>.cbl</code></p>

Note: `<target table name>` is the file name on the target platform, this file name is furnished in the mapper file.

Generated Objects

The following sections describe the objects generated during the migration of z/OS files and the directories in which they are placed.

Unloading JCL

The JCL used to unload the files are generated using the `-g` option of the `file.sh` command. They are then (using the `-i` option) installed in:

\$HOME/trf/unload/file/<configuration name>

Each JCL contains two steps and unloads one file using the z/OS IDCAMS REPRO utility. The JCL return code is equal to 0 or 4 for a normal termination.

- | | | |
|--------|--------|---|
| Step 1 | DEL | IDCAMS DELETE files (deletion of log, data files) |
| Step 2 | UNLOAD | IDCAMS REPRO of the indicated file |
| | | Specifically for GDG files, one STEP for each VERSION |

The JCLs are named: <file name>.jclunload

Note: The .jclunload extension should be deleted for execution under z/OS.

The generated JCL may need adapting to specific site constraints including:

- JOB cards: <cardjob_parameter_id>,
- access paths to input and output files: <data>.

Unloading JCL for QSAM and VSAM files

Listing 6-7 Unload JCL Example

```
//<crdjob> <cardjob_parameter_1>,'FIL QSAM',  
//          <cardjob_parameter_2>  
//          <cardjob_parameter_3>  
//          <cardjob_parameter_4>  
  
//*@ (C) Metaware:jcl-unload-MVS-REPRO.pgm. $Revision: 1.6 $  
//*****  
//* UNLOAD THE FILE:  
//*          <datain>.QSAM.CUSTOMER  
//* INTO <data>.AV.QSAM  
//*          LENGTH=266  
//*****
```

```

// *-----*
// * DELETE DATA AND LOG FILES
// *-----*

//DEL          EXEC PGM=IDCAMS

//SYSPRINT DD SYSOUT=*

//SYSOUT DD SYSOUT=*

//SYSIN DD *

    DELETE <data>.AV.QSAM.LOG

    DELETE <data>.AV.QSAM

    SET MAXCC=0

// *-----*
// * LAUNCH REPRO UTILITY
// *-----*

//COPYFILE EXEC PGM=IDCAMS

//SYSPRINT DD SPACE=(CYL,(150,150),RLSE),

//          DISP=(NEW,CATLG),

//          UNIT=SYSDA,

//          DSN=<data>.AV.QSAM.LOG

//SYSOUT DD SYSOUT=*

//INDD DD DISP=SHR,

          DSN=METAW00.QSAM.CUSTOMER

//OUTD DD SPACE=(CYL,(150,150),RLSE),

//          DISP=(NEW,CATLG),

//          UNIT=SYSDA,

//          DCB=(LRECL=266,RECFM=FB),

//          DSN=<data>.AV.QSAM

//SYSIN DD *

```

```
REPRO INFILE(INDD) OUTFILE(OUTD)
```

```
/*
```

Unloading JCL for Generation Data Group

The JCL used to unload the Generation Data Sets for Generation Data Group organization are also generated using the `-g` option of the `file.sh` command.

This JCL creates two files for each VERSION (number of version is given by the `LIMIT` clause).

Created files are named:

- `<data>.<filename>.DAT<id>`
- `<data>.<filename>.LOG<id>`

where `<id>` is a numerical value which identifies all version (0: current version, 1: first previous, 2: ...)

Listing 6-8 Unload JCL Example for GDG

```
//<crdjob> <cardjob_parameter_1>,'GDG GDG',
//          <cardjob_parameter_2>
//          <cardjob_parameter_3>
//          <cardjob_parameter_4>
//@ (C) Metaware:jcl-unload-GDG-MVS-REPRO.pgm. $Revision: 1.5 $
//*****
/* UNLOAD GDS FILES
/*          <datain>.PJ01DDD.TEST.GDG
/* INTO <data>.M2L3.GDG.DAT<ID>
/* CURRENT FILE HAS <ID>=0
/* OLDEST FILE HAS GREATEST <ID>.
/*          LENGTH=266
/*          LIMIT=5
```

```

// *      NOEMPTY

// *      SCRATCH

// *****

// *-----*

// * DELETE DATA AND LOG FILES

// *-----*

//DEL      EXEC PGM=IDCAMS

//SYSPRINT DD SYSOUT=*

//SYSOUT   DD SYSOUT=*

//SYSIN    DD *

DELETE <data>.M2L3.GDG.LOG0

DELETE <data>.M2L3.GDG.DAT0

DELETE <data>.M2L3.GDG.LOG1

DELETE <data>.M2L3.GDG.DAT1

DELETE <data>.M2L3.GDG.LOG2

DELETE <data>.M2L3.GDG.DAT2

DELETE <data>.M2L3.GDG.LOG3

DELETE <data>.M2L3.GDG.DAT3

DELETE <data>.M2L3.GDG.LOG4

DELETE <data>.M2L3.GDG.DAT4

SET MAXCC=0

// *-----*

// * LAUNCH REPRO UTILITY

// *-----*

// *** 0 ***

//COPYFIO EXEC PGM=IDCAMS

//SYSPRINT DD SPACE=(CYL,(150,150),RLSE),

```

```

//          DISP=(NEW,CATLG),
//          UNIT=SYSDA,
//          DSN=<data>.M2L3.GDG.LOG0
//SYSOUT    DD  SYSOUT=*
//INDD      DD  DISP=SHR,
//          DSN=PJ01DDD.TEST.GDG(0)
//OUTD      DD  SPACE=(CYL,(150,150),RLSE),
//          DISP=(NEW,CATLG),
//          UNIT=SYSDA,
//          DCB=(LRECL=266,RECFM=FB),
//          DSN=<data>.M2L3.GDG.DAT0
//SYSIN     DD  *
            REPRO INFILE(INDD) OUTFILE(OUTD)
/*
//*** 1 ***
//COPYFIL   EXEC  PGM=IDCAMS
//SYSPRINT  DD  SPACE=(CYL,(150,150),RLSE),
//          DISP=(NEW,CATLG),
//          UNIT=SYSDA,
//          DSN=<data>.M2L3.GDG.LOG1
//SYSOUT    DD  SYSOUT=*
//INDD      DD  DISP=SHR,
//          DSN=PJ01DDD.TEST.GDG(-1)
//OUTD      DD  SPACE=(CYL,(150,150),RLSE),
//          DISP=(NEW,CATLG),
//          UNIT=SYSDA,
//          DCB=(LRECL=266,RECFM=FB),

```

```

//          DSN=<data>.M2L3.GDG.DAT1
//SYSIN      DD *
        REPRO INFILE(INDD) OUTFILE(OUTD)
/*
//*** 2 ***
[...]
/*
//*** 3 ***
[...]
/*
//*** 4 ***
//COPYFI4   EXEC PGM=IDCAMS
//SYSPRINT DD SPACE=(CYL,(150,150),RLSE),
//          DISP=(NEW,CATLG),
//          UNIT=SYSDA,
//          DSN=<data>.M2L3.GDG.LOG4
//SYSOUT    DD SYSOUT=*
//INDD      DD DISP=SHR,
//          DSN=PJ01DDD.TEST.GDG(-4)
//OUTD      DD SPACE=(CYL,(150,150),RLSE),
//          DISP=(NEW,CATLG),
//          UNIT=SYSDA,
//          DCB=(LRECL=266,RECFM=FB),
//          DSN=<data>.M2L3.GDG.DAT4
//SYSIN      DD *
        REPRO INFILE(INDD) OUTFILE(OUTD)
/*

```

COBOL Transcoding Programs

Migration of z/OS Files to UNIX/Linux Files

The COBOL transcoding programs are generated using the `-g` option of the `file.sh` command. They are then (using the `-i` option) installed in:

```
$HOME/trf/reload/file/<configuration name>/src
```

The programs are named: `RELFILE-<logical file name>.cbl`

The programs should be compiled using the Microfocus COBOL compilation options documented in [Compiler Options](#).

The compilation of these programs requires the presence of a `CONVERTMW.cpy` copy file adapted to the project, documented in [Codeset Conversion](#) chapter.

These files read a file on input and write an output file of the same organization as on z/OS (Sequential, Relative, Indexed). For sequential files, the organization in the UNIX/Linux Micro Focus environment will be Line Sequential.

Listing 6-9 FILE CONTROL Section - for Transcoding Programs

```
SELECT MW-ENTREE

        ASSIGN TO "ENTREE"

        ORGANIZATION IS SEQUENTIAL

        ACCESS IS SEQUENTIAL

        FILE STATUS IS IO-STATUS.

SELECT MW-SORTIE

        ASSIGN TO "SORTIE"

        ORGANIZATION IS LINE SEQUENTIAL

        FILE STATUS IS IO-STATUS.
```

A record count is written to the output file and is displayed at the end of processing via:

```
DISPLAY "RELOADING TERMINATED OK".

DISPLAY "Nb rows reloaded: " D-NB-RECS.
```

```

DISPLAY " " .
DISPLAY "NUMERIC MOVED WHEN USING CHAR FORMAT: "
DISPLAY "  NUMERIC-BCD : " MW-COUNT-NUMERIC-BCD-USE-X.
DISPLAY "  NUMERIC-DISP: " MW-COUNT-NUMERIC-DISP-USE-X.

```

The last two lines displayed signal the movement of data into fields where the COBOL description does not match the content of the input file (packed numeric fields containing non-numeric data and numeric DISPLAY fields containing non-numeric data). When such cases are encountered, each field name and its value is displayed.

Note: When migrating to a target platform using Intel hardware the message: “PROCESSOR UNIT IS INTEL” is output at the beginning of transcoding.

Listing 6-10 FILE CONTROL Section - for Transcoding Programs

```

SELECT MW-ENTREE

    ASSIGN TO "ENTREE"

    ORGANIZATION IS RECORD SEQUENTIAL

    ACCESS IS SEQUENTIAL

    FILE STATUS IS IO-STATUS.

```

Reloading Korn Shell Scripts

The Reloading Korn shell scripts are generated using the `-g` option of the `file.sh` command. They are then (using the `-i` option) installed in:

```
$HOME/trf/reload/file/<configuration name>
```

Reloading Korn Shell Scripts for Migrating z/OS QSAM/VSAM Files to UNIX/Linux Files

The scripts are named: `loadfile-<logical file name>.ksh`

They contain a transcoding (or loading) phase and a check phase. These different phases can be launched separately.

The execution of the scripts produces an execution log in `$MT_LOG/<logical file name>.log`

The following variables are set at the beginning of each script:

Listing 6-11 Reloading File Script Variables

```
f="@ (c) Metaware:reload-files-ksh.pgm. $Revision: 1.9 $null"
echo "Reloading file ODCSFU ODCSFU"
export DD_ENTREE=${DD_ENTREE:-${DATA_SOURCE}/ODCSFU}
export DD_SORTIE=${DD_SORTIE:-${DATA}/ODCSFU}
logfile=${MT_LOG}/ODCSFU.log
reportfile=${MT_LOG}/ODCSFU.rpt
[...]
```

Note: To change the file names, set the DD_ENTREE and DD_SORTIE variables before calling the script.

Various messages may be generated during the execution phases of the scripts, these messages are explained in [Oracle Tuxedo Application Rehosting Workbench Messages](#).

On normal end, a return code of 0 is returned.

Reloading Korn Shell Scripts for Migrating z/OS Generation Data Set to UNIX/Linux Files

The master scripts are named: loadgdg-<logical file name>.ksh.

For each version - i.e. for each Generation Data Set - they call the script: loadgds-<logical file name>.ksh and do a check phase. The loadgdg-*ksh script contains a transcoding (or loading) phase. These different phases can be launched separately.

The execution of the master script produces an execution log in \$MT_LOG/<logical file name>.log

The following variables are set at the beginning of each script:

Listing 6-12 Reloading File Script Variables

```
f="@ (c) Metaware:reload-GDG-files-ksh.pgm. $Revision: 1.6 $null"
echo "Reloading GDG file GDG GDG"
```

```
# Remarks:

# DD_ENTREE* contains only filename prefix! This script add .DAT<ID> prefix
# DD_SORTIE* contains GDG file name
# Per default, file name transferred from MVS should be :
# $DATA_SOURCE/GDG.DAT<ID>

export DD_ENTREE_ORIGDG=${DD_ENTREE:-${DATA_SOURCE}/GDG}
export DD_SORTIE_ORIGDG=${DD_SORTIE:-${DATA}/PJ01DDD.TEST.GDG}

logfile=${MT_LOG}/GDG.log

tmpfile=${TMPPROJECT}/GDG.tmp

ksh4ejr=${DATA_TRANSCODE}/GDG.ksh

[...]
```

Note: To change the prefix file names, set the DD_ENTREE and DD_SORTIE variables before calling the script.

Various messages may be generated during the execution phases of the scripts, these messages are explained in [Oracle Tuxedo Application Rehosting Workbench Messages](#).

On normal end, a return code of 0 is returned.

Transcoding and Loading Phases

These steps launch the execution of the COBOL transcoding program associated with the file processed:

```
cobrun RELFILE-ODCSFU >> $logfile
```

On normal termination the following message is displayed:

```
echo "File ${DD_ENTREE} successfully transcoded and reloaded into
${DD_SORTIE}"
```

Check Phase

This step verifies after the reloading that the targeted file contains the same number of records as were transferred from the z/OS source platform. If the number of records is different, an error message is produced:

FILELD-0106: the number of rows written in file <f> is not equal to the number calculated using the log file (see created report <rf>) !

File : <recsreloaded>

Report: <recstransferred>

If the number of records is equal, this message is produced:

```
echo "Number of rows written in output file is equal to number calculated
using the log file: OK"
```

Note: To execute this step, it is necessary to transfer the z/OS IDCAMS log file to the target environment.

Access Functions and Utility Programs

Access Functions

These access functions are generated using the `-g` option of `file.sh` and installed in `$HOME/trf/DML` using the `-i` and `-s` options.

Table 6-6 Access Functions

Access Function	Role
ASG_<logical file name>.cbl	Optional module generated when there are multiple assigns. When using the File-to-File Converter tool, this module is generated when the <code>attributes</code> clause is present in the mapper configuration file.
getfileinfo.cbl	This program checks if the <logical file name>.rdp associated with the assign-name given as an input argument exists. This function is called by ASG_<logical file name>.cbl.
init_all_files.cbl	Initialize all variables used by relational module and logical module (function used by Oracle Tuxedo Application Runtime Batch).
init_all_files_<configuration name>.cbl	Initialize all variables used by relational module and ASG_<logical file name> module for the configuration name listed; (function used by Oracle Tuxedo Application Runtime Batch).

Table 6-6 Access Functions

Access Function	Role
<code>dml_locking.cbl</code>	This program manages locking for all configuration files (function used by Oracle Tuxedo Application Runtime Batch).
<code>close_all_files_<configuration name>.cbl</code>	This program closes all cursors opened in tables for the configuration listed and closes all files opened with logical accessor (function used by Oracle Tuxedo Application Runtime Batch).
<code>close_all_files.cbl</code>	This program closes all cursors opened in tables (function used by Oracle Tuxedo Application Runtime Batch).

Access Function Call Arguments

The ASG_<logical file name>.cbl access functions use the following variables

Table 6-7 Access Call Implemented Variables

Variable	Description/origin
Function code	Indicates the type of operation to execute, for example OPEN, WRITE, etc. The code is passed using the FILE-CODE-F variable of the MWFITECH copy file.
File open mode	A file can be opened in different modes: INPUT, OUTPUT, I O, EXTEND. The mode is passed using the FILE-OPEN-MODE variable of the MWFITECH copy file.
IO-STATUS	The IO-STATUS variable is linked to each file providing the execution status of the last relational module operation.
Record to transmit or receive	The record to transmit has an access function for write operations or access by key; the record to receive has a read access function. These are described in the LINKAGE SECTION.

Table 6-7 Access Call Implemented Variables

Variable	Description/origin
Name of secondary key to use	<p>For indexed files with secondary keys, and only for files with this organization, an extra variable is required to identify the secondary key to use for a START operation.</p> <p>The name of the secondary key is passed using the <code>FILE-ALT-KEY-NAME</code> variable of the MWFITECH copy file.</p> <p>For files without secondary keys, this argument is unnecessary.</p>
Relative Key	<p>For a relative file, the value of the relative key is passed to or from the access module using the <code>FILE-REL-KEY</code> variable of the MWFITECH copy file.</p>

Listing 6-13 LINKAGE SECTION Structure

```
LINKAGE SECTION.
```

```
    01  IO-STATUS  PIC XX.
```

```
    COPY MWFITECH.
```

```
*      *COBOL Record Description
```

```
    01  VS-ODCSF0-RECORD.
```

```
        06  X-VS-CUSTIDENT.
```

```
            07  VS-CUSTIDENT          PIC 9(006).
```

```
        06  VS-CUSTLNAME              PIC X(030).
```

```
        06  VS-CUSTFNAME              PIC X(020).
```

```
        06  VS-CUSTADDRS              PIC X(030).
```

```
        06  VS-CUSTCITY                PIC X(020).
```

```
        06  VS-CUSTSTATE              PIC X(002).
```

```
        06  X-VS-CUSTBDATE.
```

```
            07  VS-CUSTBDATE          PIC 9(008).
```

```
        06  VS-CUSTBDATE-G            REDEFINES VS-CUSTBDATE.
```

```

11 X-VS-CUSTBDATE-CC.
           12 VS-CUSTBDATE-CC  PIC 9(002).
11 X-VS-CUSTBDATE-YY.
           12 VS-CUSTBDATE-YY  PIC 9(002).
11 X-VS-CUSTBDATE-MM.
           12 VS-CUSTBDATE-MM  PIC 9(002).
11 X-VS-CUSTBDATE-DD.
           12 VS-CUSTBDATE-DD  PIC 9(002).
06 VS-CUSTEMAIL           PIC X(040).
06 X-VS-CUSTPHONE.
           07 VS-CUSTPHONE     PIC 9(010).
06 VS-FILLER              PIC X(100).

PROCEDURE DIVISION USING IO-STATUS
                        MW-FILE-TECH
                        VS-ODCSF0-RECORD.

```

Call Arguments Used

OPEN

For all OPEN operations, the FILE-CODE-F variable should contain the key-word OPEN.

The FILE-OPEN-MODE variable should contain the type of OPEN to perform as follows:.

Table 6-8 Call Argument File Open Modes

Source	Target
OPEN INPUT filename1	INPUT => FILE-OPEN-MODE
OPEN OUTPUT filename1	OUTPUT => FILE-OPEN-MODE

Table 6-8 Call Argument File Open Modes

Source	Target
OPEN I-O filename1	I-O => FILE-OPEN-MODE
OPEN EXTEND filename1	EXTEND => FILE-OPEN-MODE

CLOSE

For CLOSE operations, the FILE-CODE-F variable should contain the key-word CLOSE.

CLOSE-LOCK

For CLOSE LOCK operations, the FILE-CODE-F variable should contain the key-word CLOSE-LOCK.

DELETE

Depending on the file access mode, the DELETE operation is either the current record or the one indicated by the file key.

The corresponding function code is indicated as follows:

Table 6-9 Call Argument Delete Modes

Access	Source	Target
Sequential	DELETE filename1	DELETE-CUR => FILE-CODE-F
Random or dynamic	DELETE filename1	DELETE-KEY => FILE-CODE-F

READ

The function code depends on the file access mode and the type of read required: sequential read, read primary key or read secondary key

Table 6-10 Read Operation Values Depending on Arguments

Access	Source	Target
Sequential	READ filename1 [NEXT]	READ-NEXT => FILE-CODE-F
Random	READ filename1	READ-KEY => FILE-CODE-F
Dynamic	READ filename1 NEXT	READ-NEXT => FILE-CODE-F
	READ filename1	READ-KEY => FILE-CODE-F
	READ filename1 PREVIOUS	READ-PREV => FILE-CODE-F
If <i>DataName1</i> is a variable corresponding to the keyAltKey1	READ filename1 KEY <i>DataName1</i>	READ-ALT-KEY => FILE-CODE-F "AltKey1" => FILE-ALT-KEY-NAME
<i>DataName1</i> represents the relative key	READ filename1 KEY <i>DataName1</i>	READ-REL-KEY => FILE-CODE-F "RelKeyVar" => FILE-REL-KEY

Note: If the INTO clause is found, a MOVE operation is added after the call in order to set the value of the indicated field.

REWRITE

The function code depends on the file access mode and the type of read required: sequential read, read primary key or read secondary key.

Table 6-11 Rewrite Operation Values Depending on Arguments

Access	Source	Target
Sequential	REWRITE RecName1	REWRITE-CUR => FILE-CODE-F
Random or dynamic	REWRITE RecName1	REWRITE-KEY => FILE-CODE-F

Note: If the FROM clause is found, a MOVE operation is added before the call in order to set the value of the indicated field.

START

Whether the file is relative, indexed, with or without secondary key, the function code depends on the exact type of start.

Table 6-12 Rewrite Operation Values Depending on Arguments

When	Source	Target
	START file1	START-EQUAL => FILE-CODE-F
DataName1 represents the relative key or Primary Key of file1	START file1 KEY { EQUAL = EQUALS } DataName1	START-EQUAL => FILE-CODE-F
	START file1 KEY { EXCEEDS > GREATER } DataName1	START-SUP => FILE-CODE-F
	START file1 KEY { NOT LESS GREATER OR EQUAL NOT < >= } DataName1	START-SUPEQ => FILE-CODE-F
	START file1 KEY {< LESS } DataName1	START-INF => FILE-CODE-F
	START file1 KEY { NOT GREATER LESS OR EQUAL NOT > <= } DataName1	START-INFEQ => FILE-CODE-F

Table 6-12 Rewrite Operation Values Depending on Arguments

When	Source	Target
DataName1 is a variable corresponding to the AltKey1 key	START file1 KEY { EQUAL = EQUALS } DataName1	AltKey1 => FILE-ALT-KEY-NAME START-ALT-EQUAL => FILE-CODE-F
	START file1 KEY { EXCEEDS > GREATER } DataName1	AltKey1 => FILE-ALT-KEY-NAME START-ALT-SUP => FILE-CODE-F
	START file1 KEY { NOT LESS GREATER OR EQUAL NOT < >= } DataName1	AltKey1 => FILE-ALT-KEY-NAME START-ALT-SUPEQ => FILE-CODE-F
	START file1 KEY { < LESS } DataName1	AltKey1 => FILE-ALT-KEY-NAME START-ALT-INF => FILE-CODE-F
	START file1 KEY { NOT GREATER LESS OR EQUAL NOT > <= } DataName1	AltKey1 => FILE-ALT-KEY-NAME START-ALT-INFEQ => FILE-CODE-F

WRITE

The function code depends on the file access mode and the type of read required: sequential read, read primary key or read secondary key.

Table 6-13 Write Operation Values Depending on Arguments

Access	Source	Target
Sequential	WRITE RecName1	WRITE-SEQ => FILE-CODE-F
Random or dynamic	WRITE RecName1	WRITE-KEY => FILE-CODE-F

Note: If the FROM clause is found, a MOVE operation is added before the call in order to set the value of the indicated field.

Copy Files to Be Implemented

The following copy files are used by certain access functions. They should be placed in the directory: `< installation platform>/fixed-copy/` during the installation of the Rehosting Workbench:

- `MW-PARAM-TRACE-VAR.cpy`
- `MW-PARAM-TRACE.cpy`
- `MW-PARAM-GETFILEINFO-VAR.cpy`
- `MW-PARAM-GETFILEINFO.cpy`
- `MW-PARAM-ERROR-VAR.cpy`
- `MW-PARAM-ERROR.cpy`
- `MW-PARAM-DML-LOCKING.cpy`
- `MWFITECH.cpy`
- `ERROR-SQLCODE.cpy`

Execution Reports

`file.sh` creates different execution reports depending on the options chosen. In the following examples the following command is used:

```
file.sh -gmi $HOME/trf STFILEFILE
```

Listing 6-14 Messages Produced when Using the Options -g with file.sh (step 1)

```
#####
Control of configuration STFILEFILE
#####
Control of templates
OK: Use Default Templates list file
```

```

File name is
/Qarefine/release/M3_L3_5/convert-data/default/file/file-templates.txt
#####
Control of Mapper
#####
COMPONENTS GENERATION

CMD : /Qarefine/release/M3_L3_5/scripts/launch file-converter -s
/home2/wkb4/param/system.desc -mf
/home2/wkb4/tmp/mapper-STFILEFILE.re.tmp -dmf
/home2/wkb4/param/file/Datamap-STFILEFILE.re -td /home2/wkb4/tmp -tmps
/home2/wkb4/tmp/file-templates-STFILEFILE.tmp -target-sgbd oracle11
-target-os unix -varchar2 29 -print-ddl -print-dml -abort

MetaWorld starter

Loading lib: /Qarefine/release/M3_L3_5/Linux64/lib64/localext.so
(funcall LOAD-THE-SYS-AND-APPLY-DMAP-AND-MAPPER)

FILE-0092: *File-Converter*: We are in BATCH mode

FILE-0087: * Comand line arguments: begining of analyze

FILE-0088: * recognized argument -s value: /home2/wkb4/param/system.desc

FILE-0088: * recognized argument -mf value:
/home2/wkb4/tmp/mapper-STFILEFILE.re.tmp

FILE-0088: * recognized argument -dmf value:
/home2/wkb4/param/file/Datamap-STFILEFILE.re

FILE-0088: * recognized argument -td value: /home2/wkb4/tmp

FILE-0088: * recognized argument -tmps value:
/home2/wkb4/tmp/file-templates-STFILEFILE.tmp

FILE-0088: * recognized argument -target-sgbd value: oracle11

FILE-0088: * recognized argument -target-os value: unix

FILE-0088: * recognized argument -varchar2 value: 29

FILE-0089: * recognized argument -print-ddl

FILE-0089: * recognized argument -print-dml

```

```

FILE-0089: * recognized argument -abort
FILE-0091: * End of Analyze
FILE-0094: * Parsing mapper file /home2/wkb4/tmp/mapper-STFILEFILE.re.tmp
FILE-0095: * Parsing data-map file
/home2/wkb4/param/file/Datamap-STFILEFILE.re
FILE-0096: * Parsing system description file /home2/wkb4/param/system.desc
Warning! OS clause is absent, assuming OS is IBM
Current OS is IBM-MF
Loading /home2/wkb4/source/symtab-STFILEFILE.pob at 12:10:27... done at
12:10:27
Build-Symtab-DL1 #1<a SYMTAB-DL1>
    ... Postanalyze-System-RPL...
sym=#2<a SYMTAB>
PostAnalyze-Common #2<a SYMTAB>
    0 classes
    0 classes
    0 classes
    0 classes
    1 classes
    13 classes
Loading /home2/wkb4/source/BATCH/pob/RSSABB01.cbl.shrec...
Loading /home2/wkb4/source/COPY/pob/ODCSF0.cpy.cdm...
Loading /home2/wkb4/source/COPY/pob/ODCSF0B.cpy.cdm...
Loading /home2/wkb4/source/COPY/pob/ODCSFU.cpy.cdm...
FILE-0001: * Point 1 !!
FILE-0002: * Point 2 !!
FILE-0010: * Parsing file /home2/wkb4/source/COPY/ODCSF0.cpy ...
*Parsed 22 lines*

```

```

FILE-0010: * Parsing file
/home2/wkb4/source/../../param/file/rec-source/ODCSFR.cpy ...

*Parsed 8 lines*

FILE-0010: * Parsing file /home2/wkb4/source/COPY/ODCSFU.cpy ...

*Parsed 24 lines*

FILE-0010: * Parsing file /home2/wkb4/source/COPY/ODCSF0B.cpy ...

*Parsed 22 lines*

FILE-0003: * Point 3 !!

FILE-0004: * Point 4 !!

FILE-0005: * Point 5 !!

FILE-0052: * loading pob file
/Qarefine/release/M3_L3_5/convert-data/templates/file/unloading/jcl-unload
-MVS-REPRO.pgm.pob

FILE-0085: * Expanding
/Qarefine/release/M3_L3_5/convert-data/templates/file/unloading/jcl-unload
-MVS-REPRO.pgm ...

FILE-0054: * Writing ODCSFR.jclunload

FILE-0054: * Writing ODCSFU.jclunload

FILE-0054: * Writing ODCSF0Q.jclunload

[...]

FILE-0052: * loading pob file
/Qarefine/release/M3_L3_5/convert-data/templates/file/dml/generate-post-pr
ocess.pgm.pob

FILE-0085: * Expanding
/Qarefine/release/M3_L3_5/convert-data/templates/file/dml/generate-post-pr
ocess.pgm ...

FILE-0054: * Writing post-process-file.sh

FILE-0053: * Parsing template file
/Qarefine/release/M3_L3_5/convert-data/default/file/file-move-assignation.
pgm

```

```

FILE-0085: * Expanding
/Qarefine/release/M3_L3_5/convert-data/default/file/file-move-assignment.
pgm ...

FILE-0054: * Writing file-move-assignment.lst

Rest in peace, Refine...

*-----
Generated components are in /home2/wkb4/tmp/Template/STFILEFILE
(Optionaly in /home2/wkb4/tmp/SQL/STFILEFILE)
*-----

```

Listing 6-15 Messages Produced when Using the Options -m with file.sh (step 2)

```

#####
FORMATTING COBOL LINES
#####
CHANGE ATTRIBUTE TO KSH or SH scripts
*-----
Components are modified into /home2/wkb4/tmp directory
*-----
#####
INSTALL COMPONENTS INTO SPECIFIC DIRECTORY USING file-move-assignment.lst
=====
==_PJ01AAA.SS.VSAM.CUSTOMER_==

Copied <Templates>:ASG_ODCSF0.cbl to <td>/DML/ASG_ODCSF0.cbl
=====
==_PJ01AAA.SS.QSAM.CUSTOMER.REPORT_==

Copied <Templates>:ODCSFR.jclunload to
<td>/unload/file/STFILEFILE/ODCSFR.jclunload

```

```

Copied    <Templates>:loadfile-ODCSFR.ksh to
<td>/reload/file/STFILEFILE/loadfile-ODCSFR.ksh

Copied    <Templates>:RELFILE-ODCSFR.cbl to
<td>/reload/file/STFILEFILE/RELFILE-ODCSFR.cbl

=====

==_PJ01AAA.SS.QSAM.CUSTOMER.UPDATE_==

Copied    <Templates>:ODCSFU.jclunload to
<td>/unload/file/STFILEFILE/ODCSFU.jclunload

Copied    <Templates>:loadfile-ODCSFU.ksh to
<td>/reload/file/STFILEFILE/loadfile-ODCSFU.ksh

Copied    <Templates>:RELFILE-ODCSFU.cbl to
<td>/reload/file/STFILEFILE/RELFILE-ODCSFU.cbl

=====

==_PJ01AAA.SS.QSAM.CUSTOMER_==

Copied    <Templates>:ODCSF0Q.jclunload to
<td>/unload/file/STFILEFILE/ODCSF0Q.jclunload

Copied    <Templates>:loadfile-ODCSF0Q.ksh to
<td>/reload/file/STFILEFILE/loadfile-ODCSF0Q.ksh

Copied    <Templates>:RELFILE-ODCSF0Q.cbl to
<td>/reload/file/STFILEFILE/RELFILE-ODCSF0Q.cbl

=====

Copied    <Templates>:close_all_files_STFILEFILE.cbl to
<td>/DML/close_all_files_STFILEFILE.cbl

Copied    <Templates>:init_all_files_STFILEFILE.cbl to
<td>/DML/init_all_files_STFILEFILE.cbl

Copied    <Templates>:reload-files.txt to
<td>/reload/file/STFILEFILE/reload-files.txt

Copied    <fixed-components>:getfileinfo.cbl to <td>/DML/getfileinfo.cbl

Copied    <fixed-components>:RunSqlLoader.sh to
<td>/reload/bin/RunSqlLoader.sh

```

```

Copied    <fixed-components>:CreateReportFromMVS.sh to
<td>/reload/bin/CreateReportFromMVS.sh

=====

Dynamic_configuration

Copied_!  <Templates>:File-in-table-STFILEFILE to
/home2/wkb4/param/dynamic-config/File-in-table-STFILEFILE (is empty)

Copied    <Templates>:../.. /Conv-ctrl-STFILEFILE to
/home2/wkb4/param/dynamic-config/Conv-ctrl-STFILEFILE

=====

post-process

executed <Templates>:post-process-file.sh

/home2/wkb4/param/dynamic-config/Conv-ctrl-STFILEFILE treated

=====

Number of copied files:      18

Number of executed scripts: 1

Number of ignored files:     0

#####

```

Detailed Processing

This section describes the [Command-Line Syntax](#) used by the File-to-File Converter, and the [Process Steps](#) summary.

The processes required on the source and target platforms concern:

- [Configuring the Environments and Installing the Components](#),
- [Unloading Data](#),
- [Transferring the Data](#),
- [Reloading the Data](#),
- [Checking the Transfers](#)

Command-Line Syntax

file.sh

Name

`file.sh` - generate file migration components.

Synopsis

```
file.sh [ [-g] [-m] [-i <installation directory>] <configuration name> | -s  
<installation directory> (<configuration name1>,<configuration name2>,...)  
]
```

Description

`file.sh` generates the Rehosting Workbench components used to migrate z/OS files to UNIX Micro Focus files.

Options

Generation Options

-g <configuration name>

Triggers the generation, for the configuration indicated, of the unloading and loading components in \$TMPPROJECT. This generation depends on the information found in the configuration files.

Modification Options

-m <configuration name>

Makes the generated SHELL scripts executable. COBOL programs are adapted to Micro Focus COBOL fixed format. When present, the shell script described in [File Modifying Generated Components](#) is executed.

Installation Option

-i <installation directory> <configuration name>

Places the components in the installation directory. This operation uses the information located in the [file-move-assignation.pgm](#) file.

Final Option

-s <installation directory> (<configuration name 1>, <configuration name 2>, ...)

Enables the generation of the COBOL and JCL converter configuration files. These generated files take all of the unitary files of the project.

All these files are created in `$PARAM/dynamic-config`

Example

```
file.sh -gmi $HOME/trf FTFIL001
```

Unitary Usage Sequence

If the `file.sh` options are used one at a time, they should be used in the following order:

1. => -g
2. => -m
3. => -i
4. => -s (should be executed once steps 1 to 3 have been executed for all configurations).

Process Steps

Configuring the Environments and Installing the Components

This section describes the preparation work on the source and target platforms.

Installing the Unloading Components Under z/OS

The components used for the unloading (generated in `$HOME/trf/unload/file`) should be installed on the source z/OS platform (the generated JCL may need adapting to specific site constraints including JOB cards, library access paths and access paths to input and out put files).

Installing the Reloading Components on the Target Platform

The components used for the reloading (generated in `$HOME/trf/reload/file`) should be installed on the target platform.

[Table 6-14](#) lists the environment variables that should be set on the target platform.

Table 6-14 Target Platform Environment Variables

Variable	Value
DATA_SOURCE	The name of the directory containing the unloaded files transferred from z/OS to be reloaded into MicroFocus files.
DATA	The name of the directory containing the physical files converted to ASCII format and ready to be used by the applications.
BIN	The location of the generic reload and control scripts (\$HOME/trf/reload/bin).
TMPPROJECT	The temporary directory.
MT_LOG	Directory to contain execution logs.
DATA_TRANSCODE	Temporary directory used by the file binary data transcoding script (contains temporary files in ASCII format).
PATH	This UNIX/Linux variable has to contain the directory of the Oracle Tuxedo Application Runtime for Batch utilities.

Compiling COBOL Transcoding Programs

The COBOL transcoding programs should be compiled using the options specified in [Compiler Options](#).

Compiling these programs requires the presence of a copy of `CONVERTMW.cpy` adapted to the project.

Unloading Data

To unload each file, a JCL using the IBM IDCAMS REPRO utility is executed. The IDCAMS REPRO utility creates two files for each file:

- a data file,
- a log file

These unloading JCLs are named `<logical filename>.jclunload`

A return code of 0 is sent on normal job end.

Transferring the Data

The unloaded data files should be transferred between the source z/OS platform and the target UNIX/Linux platform in binary format using the file transfer tools available at the site (CFT, FTP, ...).

The files transferred to the target UNIX/Linux platform should be stored in the \$DATA_SOURCE directory.

Reloading the Data

The scripts enabling the transcoding and reloading of data are generated in the directory:

```
$HOME/trf/reload/file/<configuration name>/
```

the format of the script names are:

```
loadfile-<logical file name>.ksh
```

```
loadgdg-<logical file name>.ksh and loadgds-<logical file name>.ksh
```

Note: The loadgdg-<logical file name>.ksh script enables the execution of the different loadgds-<logical file name>.ksh scripts. Each loadgds script is used to reload one unitary generation of the file (each data set within a GDG is called a generation or a Generation Data Set – GDS).

Transcoding and Reloading Command for Files

Name

loadfile transcode and reload data to file.

Synopsis

```
loadfile-<logical file name>.ksh [-t] [-l] [-c: <method>]
```

Options

-t

Transcode and reload the file.

-l

Transcode and reload the file (same action as -t parameter).

-c ftp:<...>:<...>

Implement the verification of the transfer (see [Checking the Transfers](#)).

Transcoding and reloading command for Generation Data Group files

Name

loadgdg and loadgds transcode and reload data to file.

Synopsis

```
loadgdg-<logical file name>.ksh [-t] [-l] [-c: <method>]
```

```
loadgds-<logical file name>.ksh [-t] [-l] [-c: <method>]
```

Options

-t

Transcode the member files of the GDG.

-l

Reload the member files of the GDG using the Oracle Tuxedo Application Runtime for Batch utilities.

-c ftp:<...>:<...>

Implement the verification of the transfer (see [Checking the Transfers](#)).

Note: the loadgdg-<logical file name>.ksh script call the loadgds-<logical file name>.ksh script for each Generation Data Set.

Checking the Transfers

This check uses the following option of the loadfile-<logical file name>.ksh or loadfile-<logical file name>.ksh

```
-c ftp:<name of transferred physical file>:<name of FTP log under UNIX>
```

This option verifies, after the reloading, that the physical file transferred from z/OS and the file reloaded on the target platform contains the same number of records. This check is performed using the FTP log and the execution report of the reloading program. If the number of records is different, an error message is produced.

File-to-Oracle Converter

This chapter describes the Rehosting Workbench File-to-Oracle Converter used to migrate files from the source platform (z/OS) to Oracle tables, and describes the migration tools that are generated. The conversion is performed in the context of other components translated or generated by the other Oracle Tuxedo Application Rehosting Workbench tools.

Several configuration files need to be set, see [Description of the Configuration Files](#), before launching the conversion process.

This section is a continuation of the description contained in the [File Converter: Introduction](#) section. Several links in this section are made, for example to the different objects generated are described in [List of the Input Components](#). Some objects are only generated when migrating VSAM files to Oracle tables (PCO programs, SQL files, relational module, logical module, utilities and configuration files for JCL and COBOL conversion).

Overview of the File-to-Oracle Converter

Purpose

The purpose of this section is to describe precisely all the features of the Rehosting Workbench File-to-Oracle Converter tools including:

- Inventory of files to migrate.
- Detailed description of Oracle tables on the target platform for each file.
- Description of the different commands to be used with the File-to-Oracle Converter.

- Description of the data unloading options on the source platform.
- Description of the data loading options on the target platform.

Structure

- [Overview of the File-to-Oracle Converter](#).
- [Description of the Input Components](#) including [Description of the Configuration Files](#).
- [Description of the Output Files](#) including the [Generated Objects](#).
- [Detailed Processing](#) including the [Command-Line Syntax](#).
- For messages, see [File Converter Messages](#).

See Also

The conversion of data is closely linked to the conversion of COBOL programs, see:

- [COBOL Converter](#)

The previous chapter explains all common usages:

- [File Converter: Introduction](#)

File Organizations Processed

When migrating files from a z/OS source platform to a target platform, the first question to ask, when VSAM is concerned, is whether to keep a file or migrate the data to an Oracle table.

Migrating to Oracle Table on the Target Platform

KSDS, RRDS and ESDS VSAM files can be migrated into Oracle tables.

To make this work, the first task is to list all of the VSAM files to be migrated, and then identify those files that should be converted to Oracle tables. For example, permanent files to be later used via Oracle or files that needs locking at the record level.

Oracle Tuxedo Application Rehosting Workbench Configuration Name

A configuration name is related to a set of files to be converted. Each set of files can be freely assembled. Each configuration could be related to a different application for example, or a set of files required for Oracle Tuxedo Application Runtime. The set of files can contain both files or Oracle tables targets.

VSAM Files Becoming Oracle Table

Specific Migration Rules Applied

- Each table name is stipulated in the mapper-<configuration name>.re file using the table name clause.
- Each elementary field name contained in a copy description of the file becomes a column in an Oracle table. Hyphens (-) are replaced by underscore (_) characters.
- For sequential VSAM files (VSAM ESDS): the Rehosting Workbench adds a technical column: *_SEQ_NUM NUMBER.

This column is incremented each time a new line is added to the table and becomes the primary key of the table.

- For relative VSAM files (VSAM RRDS): the Rehosting Workbench adds a technical column: *_RELATIVE_NUM.

The size of the column is deduced from the information supplied in the Datamap parameter file; the column becomes the primary key of the table.

The column:

- is incremented when a sequential write is made to the table, and the relative key is zero.
 - contains a relative key when the relative key is not zero.
- For indexed VSAM files (VSAM KSDS): the Rehosting Workbench does not add a technical column unless duplicate keys are accepted; the primary key of the VSAM file becomes the primary key of the table.

Rules Applied to Picture Clauses

The following rules are applied to COBOL Picture clauses when migrating data from VSAM files to Oracle tables.

Table 7-1 Picture Clause Re-engineering

COBOL Picture	Oracle format
PIC 9(...)	NUMBER(...)
COMP-1	NUMBER(8)
COMP-2	NUMBER(16)
PIC X(...)	Becomes CHAR if length <= 2000 Becomes VARCHAR2 if length > 2000 If the parameter file:char_limit_until_varchar is set in the db-param.cfg file, it takes precedence over the above rule.

Environment Variables

Before starting the process of migrating data two environment variables should be set:

- `export TMPPROJECT=/ $HOME /tmp`
Indicates the location to store temporary objects generated by the process.
- `export PARAM=/ $HOME /param`
Indicates the location where the configuration files required by the process are stored.

Description of the Input Components

File Locations

Location of file.sh

The file.sh tool is located in the directory:

```
$REFINEDIR/convert-data/
```

Location of db-param.cfg File

The db-param.cfg configuration file is located in the directory given in the variable:

```
$PARAM
```

Description of the Configuration Files

This section lists the files and their parameters that can be used to control the migration of z/OS files to Oracle table.

db-param.cfg

This file should be created in the directory indicated by the \$PARAM directory:

```
$PARAM/db-param.cfg
```

Listing 7-1 db-param.cfg Template

```
#
# This configuration file is used by FILE & RDBMS converter
# Lines beginning by "#" are ignored
# write information in lower case
#
# common parameters for FILE and RDBMS
#
# source information is written into system descriptor file (DBMS=,
DBMS-VERSION=)
target_rdbms_name:<target_rdbms_name>
target_rdbms_version:<target_rdbms_version>
target_os:<target_os>
#
# specific parameters for FILE to RDBMS conversion
file:char_limit_until_varchar:<char_limit>
```

Parameters and Syntaxes

Table 7-2 db-param.cfg Parameters

Parameter	Description	Value
General Parameters		
<target_rdbms_name>	Name of target RDBMS	oracle
<target_rdbms_version>	Version of target RDBMS	11
<target_os>	Name of target operating system	unix or linux
Specific file-to-oracle conversion parameters		
<char_limit>	<ul style="list-style-type: none">For a field size <= 2000, a COBOL alphanumeric field is migrated on ORACLE in CHARFor a field size > 2000 it is migrated in VARCHAR2, except if the parameter <code>file:char_limit_until_varchar</code> is used. <p>This parameter indicates the maximum length of a COBOL alphanumeric (PIC X) field before the field will be transformed into an ORACLE VARCHAR2 data type.</p> <p>If the parameter contains: <code>file:char_limit_until_varchar:29</code></p> <p>Then, fields longer than 29 characters will become VARCHAR2, fields shorter than 30 characters will become CHAR fields.</p>	

File Modifying Generated Components

The generated components may be modified using a project's own scripts. These scripts (sed, awk, perl,...) should be placed in:

```
$PARAM/file/file-modif-source.sh
```

When present, this file will be automatically executed at the end of the generation process. It will be called using the <configuration name> as an argument.

file-template.txt

This file is put in place during the installation of the Rehosting Workbench, it contains the templates that perform the generation of the different migration tools. The file is located in:

```
$REFINEDIR/convert-data/default/file/file-templates.txt
```

Listing 7-2 file-template.txt

```
% Unloading all File *****
% All SAM file were transfered using FTP/Binary
% VSAM unloaded step:
#VAR:TEMPLATES#/unloading/jcl-unload-MVS-REPRO.pgm
%
% To create a specific template, copy this template into :
% -- #VAR:PARAM#/file/specific-templates/unloading/jcl-unload-customer.pgm
%
% Loading *****
#VAR:TEMPLATES#/loading/file-reload-files-txt.pgm
% Loading File to File *****
#VAR:TEMPLATES#/loading/unix-file/reload-files-ksh.pgm
#VAR:TEMPLATES#/loading/unix-file/reload-mono-rec.pgm
% Loading File to Oracle *****
#VAR:TEMPLATES#/loading/unix-oracle/load-tables-ksh.pgm
#VAR:TEMPLATES#/loading/unix-oracle/rel-mono-rec.pgm
#VAR:TEMPLATES#/dml/clean-tables-ksh.pgm
#VAR:TEMPLATES#/dml/drop-tables-ksh.pgm
#VAR:TEMPLATES#/dml/create-tables-ksh.pgm
#VAR:TEMPLATES#/dml/ifempty-tables-ksh.pgm
#VAR:TEMPLATES#/dml/ifexist-tables-ksh.pgm
```

```

%

% Generate Logical & Relational Module *****

#VAR:TEMPLATES#/dml/module/open-multi-assign-free.pgm
#VAR:TEMPLATES#/dml/module/open-mono-rec-idx-perf.pgm
#VAR:TEMPLATES#/dml/module/open-mono-rec-sequential.pgm
#VAR:TEMPLATES#/dml/module/open-mono-rec-relative.pgm

%

% and utilities *****

#VAR:TEMPLATES#/dml/module/decharge-mono-rec.pgm
#VAR:TEMPLATES#/dml/module/recharge-table.pgm
#VAR:TEMPLATES#/dml/module/close-all-files.pgm
#VAR:TEMPLATES#/dml/module/init-all-files.pgm

%

% configuration file for translation and runtime *****

#VAR:TEMPLATES#/dml/generate-config-wb-translator-jcl.pgm
#VAR:TEMPLATES#/dml/generate-rdb-txt.pgm

%

% included file to include into modified-components

#VAR:TEMPLATES#/dml/include-modified-components.pgm

%

% *****

% MANDATORY

% : used just after the generation

#VAR:TEMPLATES#/dml/generate-post-process.pgm

% : used when using -i arguments

#VAR:DEFAULT#/file-move-assignation.pgm

```

Note: This file contains both File-to-File and File-to-Oracle migration parameters.

When required, another version of the `file-template.txt` file can be placed in the `$PARAM/file` directory. The use of an alternative file is signaled during the execution of `file.sh` by the message:

Listing 7-3 Execution Log with Alternative Template File

```
#####
Control of templates

    OK: Use Templates list file from current project:

        File name is /home2/wkb9/param/file/file-templates.txt
#####
```

file-move-assignation.pgm

This file is placed during the installation of the Rehosting Workbench, it controls the transfer of components generated in the different installation directories. This file indicates the location of each component to copy during the installation phase of `file.sh`, when launched using `file.sh -i`.

The file is located in:

`$REFINEDIR/convert-data/default/file/file-move-assignation.pgm`

This file can be modified following the instructions found at the beginning of the file:

Listing 7-4 file-move-assignation.pgm Modification Instructions

```
[...]

* @ (c) Metaware:file-move-assignation.pgm. $Revision: 1.2 $

* release_format=2.4

*

* format is:

*   <typ>:<source_directory>:<file_name>:<target_directory>
```

```

*
* typ:
*   O: optional copy: if the <file_name> is missing, it is ignored
*   M: Mandatory copy: abort if <file_name> is missing.
*   E: Execution: execute the mandatory script <file_name>.
*
*   Parameters for script to be executed are:
*
*       basedir:          directory of REFINEDIR/convert-data
*       targetoutputdir:  value of "-i <targetdir>"
*       schema:           schema name
*       target_dir:       value written as 4th parameter in this file.
*
*   d: use this tag to display the word which follows
*
*
* source_directory:
*
*   T: generated components written in <targetdir>/Templates/<schema>
*   O: components written in <targetdir>/outputs/<schema>
*   S: SQL requests (DDL) generated into <targetdir>/SQL/<schema> directory
*   F: fixed components present in REFINEDIR
*
*   s: used with -s arguments: indicates the target directory for DML
utilities
*
*       (in REFINEDIR/modified-components/) which manipulate all schemas.
*
*
* file_name: (except for typ:d)
*
*   name of the file in <source_directory>
*
*
* target_directory: (except for typ:d, given at 4th argument for typ:E)
*
*   name of the target directory
*
*   If the 1st character is "/", component is copied using static directory
*
*   and not in <td> directory

```

```
*      If the 1st character is "!", target directory contains both directory
and
*      target file name.
*
[...]
```

Note: This file contains both File-to-File and File-to-Oracle migration parameters.

Datamap File

This is a configuration file used by the Rehosting Workbench file converter to add or modify information on the physical files of a system.

See [File Converter: Introduction – Datamap File](#) .

Mapper File

This is a configuration file used by the Rehosting Workbench File-to-Oracle Converter to associate each file to migrate

See [File Converter: Introduction – Mapper File](#).

Note: In the mapper file, the `converted` clause has to be used for RDBMS Table target.

Table 7-3 Mapper File Specific Parameters Used with the File-to-Oracle Converter

<code>file <physical filename></code>	ZOS physical file name, Name used in the Datamap file.
<code>converted</code>	Indicates file is to be converted to Oracle table (<code>converted</code> clause can be combined with <code>transferred</code> clause)
<code>transferred</code>	Indicates that the file is to be loaded and reloaded (can be combined with <code>converted</code> clause).
<code>include "<path/COPY name>"</code>	Access path and name of the descriptive copy of the file to migrate.

Table 7-3 Mapper File Specific Parameters Used with the File-to-Oracle Converter

map record <record name> defined in <"path/COPY name">	<ul style="list-style-type: none">• record name: corresponds to the level 01 field name of the copy description.• path/COPY name: corresponds to the access path and name of the descriptive copy of the file to migrate.
source record <record names> defined in <"path/COPY name">	<ul style="list-style-type: none">• record name: corresponds to the level 01 field name of the copy description of the file to migrate.• path/COPY name: corresponds to the access path and name of the descriptive copy of the file to migrate.
Logical name <logical file name>	The Logical file name is chosen by the user, maximum eight characters. This name is used for naming the objects (COBOL, JCL) created by the different tools in the Rehosting Workbench.
Converter name <program name>	Same name and use as logical file name.
table name	Oracle table name.
attributes <attribute clause>	<p>This optional clause has two attributes that can be used:</p> <ul style="list-style-type: none">• LOGICAL_MODULE_IN_ADDITION• LOGICAL_MODULE_ONLY <p>Their action is described in the next table.</p>

Table 7-4 Mapper File Attributes

attributes <attribute clause>	Role
Attribute clause absent	In this case the following access functions are generated: <ul style="list-style-type: none">• RM_<logical file name>,• UL_<logical file name>,• DL_<logical file name> and the Korn shell utilities. See Access Functions and Utility Programs .
LOGICAL_MODULE_IN_ADDITION	In this case the following access functions are generated: <ul style="list-style-type: none">• ASG_<logical file name>• RM_<logical file name>,• UL_<logical file name>,• DL_<logical file name> and the Korn shell utilities. See Access Functions and Utility Programs .
LOGICAL_MODULE_ONLY	In this case only the ASG_<logical file name> access function is generated.

Listing 7-5 Mapper File Example

```
ufas mapper STFILEORA
file PJ01AAA.SS.VSAM.CUSTOMER converted transferred
    table name CUSTOMER
    include "COPY/ODCSF0B.cpy"
    map record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"
    source record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"
    logical name ODCSF0B
    converter name ODCSF0B
    attributes LOGICAL_MODULE_IN_ADDITION
```

In this example the mapper file is named `STFILEORA`. The file processes only one file named `PJ01AAA.SS.VSAM.CUSTOMER` that is migrated to an Oracle table using the convert option. The `ODCSF0B.cpy` copy file used to describe the file is one of the source copy files.

Mapping Strategy Clauses

```
[ field <field_name>

    [ use detail table ]

    [ use opaque field <field name> ]

    [ table name <target table name> ]

    [ mapped type <target data type> ]

    [ discard field <field name> ]

    [ discard subfields <field name> ]

    [ discrimination rule ] ]
```

Mapping Strategy Clause Syntax and Parameters

For OCCURS and REDEFINES clauses, using discrimination rules, three reengineering possibilities are proposed:

- Creation of sub-tables (use detail table)
 - Redefinitions: each description is associated with a sub-table (one sub-table for each description).
 - Occurs: one sub-table is created containing a technical column that references the original table to which the data corresponds.
- Creation of an opaque field (use opaque field).
 - Redefinitions: all the descriptions are stored in an opaque field type CHAR or VARCHAR2.
 - Occurs: all the occurrences are stored in an opaque field type CHAR or VARCHAR2.
- Extended description (default)
 - Redefinitions: all the fields described in the copy file are created as columns in the Oracle table.
 - Occurs: each occurrence of a field in a redefined area is created as a column in the Oracle table, one column for each occurrence in the OCCURS clause.

Table 7-5 Mapping Strategies

Strategy	Description
table name < table name >	Name of sub-table in case of mapping 'use detail table'.
mapped type <target data type>	Enables the modification of the column type chosen by default. Two possibilities are proposed: CHAR or VARCHAR2.
discard field	Enables the deletion of a non-useful redefined field.
discard subfields	When a field has several levels of description, this option allows to keep only the higher level.

Mapping Strategy Examples

Discard Subfield Example

```

...
05 NIV1 .
    10 NIV2A PIC 99 .
    10 NIV2B PIC 999 .
...

```

When discarding subfields at the level NIV1, the Rehosting Workbench File-to-Oracle Converter only processes the field NIV1 PIC 9(5). When not discarding subfields, the NIV1 field is ignored and the two fields NIV2A and NIV2B are processed.

Redefines With Default Option Example

This redefines example is without any specific options:

Listing 7-6 Descriptive Copy of the File: PJ01AAA.SS.VSAM.CUSTOMER

```

01 VS-ODCSF0-RECORD .
    05 VS-CUSTIDENT          PIC 9(006) .
    05 VS-CUSTLNAME          PIC X(030) .
    05 VS-CUSTFNAME          PIC X(020) .

```

```

05 VS-CUSTADDRS          PIC X(030).
05 VS-CUSTCITY            PIC X(020).
05 VS-CUSTSTATE          PIC X(002).
05 VS-CUSTBDATE          PIC 9(008).
05 VS-CUSTBDATE-G        REDEFINES VS-CUSTBDATE.
    10 VS-CUSTBDATE-CC PIC 9(002).
    10 VS-CUSTBDATE-YY PIC 9(002).
    10 VS-CUSTBDATE-MM PIC 9(002).
    10 VS-CUSTBDATE-DD PIC 9(002).
05 VS-CUSTEMAIL          PIC X(040).
05 VS-CUSTPHONE          PIC 9(010).
05 VS-FILLER              PIC X(100).

```

The mapper file implemented is:

Listing 7-7 Mapper File for the File: PJ01AAA.SS.VSAM.CUSTOMER

```

ufas mapper STFILEORA

file PJ01AAA.SS.VSAM.CUSTOMER converted transferred

    table name CUSTOMER

    include "COPY/ODCSF0B.cpy"

    map record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"

    source record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"

    logical name ODCSF0B

    converter name ODCSF0B

    attributes LOGICAL_MODULE_IN_ADDITION

    field VS-CUSTBDATE

```

```
rule if VS-CUSTSTATE = "02" then VS-CUSTBDATE
else VS-CUSTBDATE-G
```

The table is generated as follows (all the unitary fields of the REDEFINES are handled).

Listing 7-8 Table Generation for the File: PJ01AAA.SS.VSAM.CUSTOMER

```
WHENEVER SQLERROR CONTINUE;

DROP TABLE CUSTOMER CASCADE CONSTRAINTS;

WHENEVER SQLERROR EXIT 3;

CREATE TABLE CUSTOMER (

    VS_CUSTIDENT            NUMBER(6) NOT NULL,

    VS_CUSTLNAME            VARCHAR2(30),

    VS_CUSTFNAME            CHAR   (20),

    VS_CUSTADDRS            VARCHAR2(30),

    VS_CUSTCITY             CHAR   (20),

    VS_CUSTSTATE            CHAR   (2),

    VS_CUSTBDATE            NUMBER(8),

    VS_CUSTBDATE_CC         NUMBER(2),

    VS_CUSTBDATE_YY         NUMBER(2),

    VS_CUSTBDATE_MM         NUMBER(2),

    VS_CUSTBDATE_DD         NUMBER(2),

    VS_CUSTEMAIL            VARCHAR2(40),

    VS_CUSTPHONE            NUMBER(10),

    VS_FILLER               VARCHAR2(100),

    CONSTRAINT PK_CUSTOMER PRIMARY KEY (

        VS_CUSTIDENT)

)
```

);

REDEFINES With OPAQUE FIELD Option Example

Listing 7-9 Descriptive Copy of the File: PJ01AAA.SS.VSAM.CUSTOMER

```
01 VS-ODCSF0-RECORD.
    05 VS-CUSTIDENT      PIC 9(006).
    05 VS-CUSTLNAME      PIC X(030).
    05 VS-CUSTFNAME      PIC X(020).
    05 VS-CUSTADDRS      PIC X(030).
    05 VS-CUSTCITY       PIC X(020).
    05 VS-CUSTSTATE      PIC X(002).
    05 VS-CUSTBDATE      PIC 9(008).
    05 VS-CUSTBDATE-G    REDEFINES VS-CUSTBDATE.
        10 VS-CUSTBDATE-CC PIC 9(002).
        10 VS-CUSTBDATE-YY PIC 9(002).
        10 VS-CUSTBDATE-MM PIC 9(002).
        10 VS-CUSTBDATE-DD PIC 9(002).
    05 VS-CUSTEMAIL      PIC X(040).
    05 VS-CUSTPHONE      PIC 9(010).
    05 VS-FILLER         PIC X(100).
```

The mapper file implemented is:

Listing 7-10 Mapper File for the File: PJ01AAA.SS.VSAM.CUSTOMER

```
ufas mapper STFILEORA
```

```

file PJ01AAA.SS.VSAM.CUSTOMER converted transferred

    table name CUSTOMER

    include "COPY/ODCSF0B.cpy"

    map record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"

    source record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"

    logical name ODCSF0B

    converter name ODCSF0B

    attributes LOGICAL_MODULE_IN_ADDITION

    field VS-CUSTBDATE

    use opaque field

    rule if VS-CUSTSTATE = "02" then VS-CUSTBDATE

    else VS-CUSTBDATE-G

```

The table is generated as follows (only the VS_CUSTBDATE field is kept).

Listing 7-11 Table Generation for the File: PJ01AAA.SS.VSAM.CUSTOMER

```

WHENEVER SQLERROR CONTINUE;

DROP TABLE CUSTOMER CASCADE CONSTRAINTS;

WHENEVER SQLERROR EXIT 3;

CREATE TABLE CUSTOMER (

    VS_CUSTIDENT            NUMBER(6) NOT NULL,

    VS_CUSTLNAME            VARCHAR2(30),

    VS_CUSTFNAME            CHAR   (20),

    VS_CUSTADDRS            VARCHAR2(30),

    VS_CUSTCITY             CHAR   (20),

    VS_CUSTSTATE            CHAR   (2),

```

```

VS_CUSTBDATE          RAW      (8),
VS_CUSTEMAIL          VARCHAR2(40),
VS_CUSTPHONE          NUMBER(10),
VS_FILLER             VARCHAR2(100),
CONSTRAINT PK_CUSTOMER PRIMARY KEY (
    VS_CUSTIDENT)
);

```

REDEFINES With DETAIL TABLE Option Example

Listing 7-12 Descriptive Copy of the File: PJ01AAA.SS.VSAM.CUSTOMER

```

01 VS-ODCSF0-RECORD.
    05 VS-CUSTIDENT          PIC 9(006).
    05 VS-CUSTLNAME          PIC X(030).
    05 VS-CUSTFNAME          PIC X(020).
    05 VS-CUSTADDRS          PIC X(030).
    05 VS-CUSTCITY           PIC X(020).
    05 VS-CUSTSTATE          PIC X(002).
    05 VS-CUSTBDATE          PIC 9(008).
    05 VS-CUSTBDATE-G        REDEFINES VS-CUSTBDATE.
    10 VS-CUSTBDATE-CC PIC 9(002).
    10 VS-CUSTBDATE-YY PIC 9(002).
    10 VS-CUSTBDATE-MM PIC 9(002).
    10 VS-CUSTBDATE-DD PIC 9(002).
    05 VS-CUSTEMAIL          PIC X(040).
    05 VS-CUSTPHONE          PIC 9(010).
    05 VS-FILLER             PIC X(100).

```

The mapper file implemented is:

Listing 7-13 Mapper File for the File: PJ01AAA.SS.VSAM.CUSTOMER

```
ufas mapper STFILEORA
file PJ01AAA.SS.VSAM.CUSTOMER converted transferred
    table name CUSTOMER
    include "COPY/ODCSF0B.cpy"
    map record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"
    source record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"
    logical name ODCSF0B
    converter name ODCSF0B
    attributes LOGICAL_MODULE_IN_ADDITION
    field VS-CUSTBDATE
    use detail table
    rule if VS-CUSTSTATE = "02" then VS-CUSTBDATE
    else VS-CUSTBDATE-G
```

The tables are generated as follows (a parent table is generated using the fields not part of the REDEFINES, and two child tables are generated, one for each REDEFINES description).

Listing 7-14 Table Generation for the File: PJ01AAA.SS.VSAM.CUSTOMER

```
WHenever SQLERROR Continue;
Drop Table CUSTOMER Cascade Constraints;
WHenever SQLERROR Exit 3;
Create Table CUSTOMER (
```

```

VS_CUSTIDENT          NUMBER(6) NOT NULL,
VS_CUSTLNAME          VARCHAR2(30),
VS_CUSTFNAME          CHAR   (20),
VS_CUSTADDRS          VARCHAR2(30),
VS_CUSTCITY           CHAR   (20),
VS_CUSTSTATE          CHAR   (2),
VS_CUSTEMAIL          VARCHAR2(40),
VS_CUSTPHONE          NUMBER(10),
VS_FILLER              VARCHAR2(100),
CONSTRAINT PK_CUSTOMER PRIMARY KEY (
    VS_CUSTIDENT)
);

WHENEVER SQLERROR CONTINUE;
DROP TABLE VS_CUSTBDATE CASCADE CONSTRAINTS;
WHENEVER SQLERROR EXIT 3;
CREATE TABLE VS_CUSTBDATE (
    VS_CUSTBDATE_CUSTIDENT    NUMBER(6) NOT NULL,
    VS_CUSTBDATE              NUMBER(8),
CONSTRAINT FK_VS_CUSTBDATE_CUSTOMER FOREIGN KEY (
    VS_CUSTBDATE_CUSTIDENT) REFERENCES CUSTOMER (
    VS_CUSTIDENT),
CONSTRAINT PK_VS_CUSTBDATE PRIMARY KEY (
    VS_CUSTBDATE_CUSTIDENT)
);

```

```

WHENEVER SQLERROR CONTINUE;

DROP TABLE VS_CUSTBDATE_G CASCADE CONSTRAINTS;

WHENEVER SQLERROR EXIT 3;

CREATE TABLE VS_CUSTBDATE_G (

    VS_CUSTBDATE_G_CUSTIDENT    NUMBER(6) NOT NULL,

    VS_CUSTBDATE_CC              NUMBER(2),

    VS_CUSTBDATE_YY              NUMBER(2),

    VS_CUSTBDATE_MM              NUMBER(2),

    VS_CUSTBDATE_DD              NUMBER(2),

    CONSTRAINT FK_VS_CUSTBDATE_G_CUSTOMER FOREIGN KEY (

        VS_CUSTBDATE_G_CUSTIDENT) REFERENCES CUSTOMER (

            VS_CUSTIDENT),

    CONSTRAINT PK_VS_CUSTBDATE_G PRIMARY KEY (

        VS_CUSTBDATE_G_CUSTIDENT));

```

Discrimination Rules

A discrimination rule must be set on the redefined field; it describes the code to determine which description of the REDEFINES to use and when.

```

[field <field_name>]

    [...]

rule if <condition> then Field_Name_x

[elseif <condition> then field_Name_y]

[else Field_Name_z]

```

Discrimination Rules Syntax and Parameters

Table 7-6 Discrimination Rules

Syntax	Description
Field_Name_{X,Y,Z}	This is the field that will be used when the associated condition is validated; this field is one of the redefined fields.
Condition	<p>Is a conditional expression composed with field name, operators and COBOL constants.</p> <ul style="list-style-type: none">• Logical operators are: not, and, or• Comparison operators are: = <> < >• Specific operators are: is numeric, is all SPACE• Following COBOL constants may be used: spaces, zeros, high-value, low-value <p>Note: These conditions can be parenthesized.</p>

Discrimination Rules Examples

In the following example the fields `DPODP-DMDCHQ`, `DPONO-PRDTIV`, `DP5CP-VALZONNUM` are redefined.

Listing 7-15 Discrimination Rule COBOL Description

```
01 ZART1.

  05 DPODP PIC X(20).

  05 DPODP-RDCRPHY PIC 9.

  05 DPODP-DMDCHQ PIC X(6).

  05 DPODP-REMCHQ REDEFINES DPODP-DMDCHQ.

    10 DPODP-REMCHQ1 PIC 999.

    10 DPODP-REMCHQ2 PIC 999.

  05 DPODP-VIREXT REDEFINES DPODP-DMDCHQ.

    10 DPODP-VIREXT1 PIC S9(11) COMP-3.

  05 DPONO-NPDT PIC X(5).
```

```
05 DPONO-PRDTIV PIC 9(8)V99.
05 DPONO-PRDPS REDEFINES DPONO-PRDTIV PIC X(10).
05 DP5CP-VALZONNUM PIC 9(6).
05 DP5CP-VALZON REDEFINES DP5CP-VALZONNUM PIC X(6).
```

The following discrimination rules are applied:

Listing 7-16 Discrimination Rules

```
field DPODP-DMDCHQ
rule if      DPODP-RDCRPHY = 1 then DPODP-DMDCHQ
    elseif DPODP-RDCRPHY = 2 then DPODP-REMCHQ
    elseif DPODP-RDCRPHY = 3 then DPODP-VIREXT
    else DPODP-DMDCHQ,
field DPONO-PRDTIV
rule if      DPONO-NPDT (1:2) = "01" then DPONO-PRDTIV
    elseif DPONO-NPDT (1:2) = "02" then DPONO-PRDPS,
field DP5CP-VALZONNUM
rule if      DPODP-RDCRPHY is numeric then DP5CP-VALZONNUM
    else DP5CP-VALZON
```

The first rule is to test the value of the numeric field `DPODP-RDCRPHY`.

The second rule tests the first two characters of an alphanumeric field `DPONO-NPDT`. Only the values 01 and 02 are allowed.

The third rule tests whether the field `DPODP-RDCRPHY` is numeric.

COBOL Description

Oracle Tuxedo Application Rehosting Workbench File-to-Oracle Converter needs a description associated with each file, so a first step consists in preparing a COBOL copy description.

Once the COBOL description files have been prepared, the copy files described in the `mapper-<configuration name>.re` file should be placed in the `$PARAM/file/recs-source` directory.

If you use a COBOL copy book from the source platform to describe a file (see [COBOL Description](#)), then it is the location of the copy book that is directly used.

Description of the Output Files

File Locations

Location of Temporary Files

The temporary objects generated by the Rehosting Workbench File-to-Oracle Converter are stored in:

```
$TMPPROJECT  
$TMPPROJECT/Template/<configuration name>  
$TMPPROJECT/outputs/<configuration name>
```

Note: The `$TMPPROJECT` variable is set to: `$HOME/tmp`

Location of Log Files

The execution log files are stored in:

- Log generated by the option `-g`:
`$TMPPROJECT/outputs mapper-log-<configuration name>`

Location of Generated Files

The unloading and loading components generated with the `-i $HOME/trf` option are placed in the following locations

Table 7-7 Component Locations

Location	Contents
<code>\$HOME/trf/unload/file/<configuration name></code>	<p>The JCL used for each unloading table are generated for each <code><configuration name></code>.</p> <p>These JCL are named:</p> <p><code><file name>.jclunload</code></p>
<code>\$HOME/trf/reload/file/<configuration name></code>	<p>For a file to Oracle migration, the programs and KSH are named:</p> <p><code>RELTABLE-<target file name>.pco</code></p> <p><code>loadtable-<target file name>.ksh</code></p>
<code>\$HOME/trf/SQL/file/<configuration name></code> (When migrating files to Oracle tables).	Location by <code><configuration name></code> of the SQL scripts used to create the Oracle objects.
<code>\$HOME/trf/config/tux</code>	Location of configuration files used by Oracle Tuxedo Application Runtime for files migrated to tables.
<code>\$HOME/trf/DML</code>	<p>List of components is depending on the optional <code>attributes</code> clause initialized in the mapper file.</p> <p>See Mapping Strategy Clause Syntax and Parameters</p>

Note: `<target table name>` is the file name on the target platform, this file name is furnished in the mapper file.

Generated Objects

The following sections describe the objects generated during the migration of z/OS files and the directories in which they are placed.

Unloading JCL

The JCL used to unload the files are generated using the `-g` option of the `file.sh` command. They are then (using the `-i` option) installed in:

```
$HOME/trf/unload/file/<configuration name>
```

Each JCL contains two steps and unloads one file using the z/OS IDCAMS REPRO utility. The JCL return code is equal to 0 or 4 for a normal termination.

Step 1 DEL IDCAMS DELETE files (deletion of log, data files)

Step 2 UNLOAD IDCAMS REPRO of the indicated file

The JCLs are named: <file name>.jclunload

Note: The .jclunload extension should be deleted for execution under z/OS.

The generated JCL may need adapting to specific site constraints including:

- JOB cards: <cardjob_parameter_id>,
- access paths to input and output files: <data>.

Listing 7-17 Unload JCL Example

```
//<crdjob> <cardjob_parameter_1>,'FIL QSAM',
//          <cardjob_parameter_2>
//          <cardjob_parameter_3>
//          <cardjob_parameter_4>
// *@ (C) Metaware:jcl-unload-MVS-REPRO.pgm. $Revision: 1.6 $
// *****
// * UNLOAD THE FILE:
// *      <datain>.QSAM.CUSTOMER
// * INTO <data>.AV.QSAM
// *      LENGTH=266
// *****
// *-----*
// * DELETE DATA AND LOG FILES
// *-----*
```

```

//DEL          EXEC PGM=IDCAMS

//SYSPRINT DD SYSOUT=*

//SYSOUT       DD SYSOUT=*

//SYSIN        DD *

    DELETE <data>.AV.QSAM.LOG

    DELETE <data>.AV.QSAM

    SET MAXCC=0

// *-----*

// * LAUNCH REPRO UTILITY

// *-----*

//COPYFILE EXEC PGM=IDCAMS

//SYSPRINT DD SPACE=(CYL,(150,150),RLSE),

//          DISP=(NEW,CATLG),

//          UNIT=SYSDA,

//          DSN=<data>.AV.QSAM.LOG

//SYSOUT     DD SYSOUT=*

//INDD       DD DISP=SHR,

              DSN=METAW00.QSAM.CUSTOMER

//OUTD       DD SPACE=(CYL,(150,150),RLSE),

//          DISP=(NEW,CATLG),

//          UNIT=SYSDA,

//          DCB=(LRECL=266,RECFM=FB),

//          DSN=<data>.AV.QSAM

//SYSIN      DD *

    REPRO INFILE(INDD) OUTFILE(OUTD)

/*

```

COBOL Transcoding Programs

Migration of z/OS Files to Oracle Tables

The COBOL transcoding programs are generated using the `-g` option of the `file.sh` command. They are then (using the `-i` option) installed in:

```
$HOME/trf/reload/file/<configuration name>/src
```

The programs are named: `RELTABLE-<logical file name>.pco`

The programs should be compiled using the Microfocus COBOL compilation options and Oracle Precompiler options documented in [Compiler Options](#).

The compilation of these programs requires the presence of a `CONVERTMW.cpy` copy file adapted to the project, documented in [Codeset Conversion](#).

These files read a file on input and directly load an Oracle table using the SQL INSERT verb.

Listing 7-18 FILE CONTROL Section - for Rranscoding Programs

```
SELECT MW-ENTREE

        ASSIGN TO "ENTREE"

        ORGANIZATION IS RECORD SEQUENTIAL

        ACCESS IS SEQUENTIAL

        FILE STATUS IS IO-STATUS.
```

For Oracle table with technical column, a SEQUENCE object is created:

```
CREATE SEQUENCE <table_name>_<type>_SEQ START WITH <num_rows>
```

A commit is made every 1000 records:

```
IF MW-NB-INSERT >= 1000
CALL "do_commit"
```

Note: The `do_commit` module is part of Oracle Tuxedo Application Runtime Batch.

A record count is written to the output file and is displayed at the end of processing via:

```
DISPLAY "RELOADING TERMINATED OK".
```

```

DISPLAY "Nb rows reloaded: " D-NB-RECS.

DISPLAY " ".

DISPLAY "NUMERIC MOVED WHEN USING CHAR FORMAT : "

DISPLAY "  NUMERIC-BCD : " MW-COUNT-NUMERIC-BCD-USE-X.

DISPLAY "  NUMERIC-DISP: " MW-COUNT-NUMERIC-DISP-USE-X.

```

The last two lines displayed signal the movement of data into fields where the COBOL description does not match the content of the input file (packed numeric fields containing non-numeric data and numeric DISPLAY fields containing non-numeric data). When such cases are encountered, each error is displayed.

Note: When migrating to a target platform using Intel hardware the message: “PROCESSOR UNIT IS INTEL” is output at the beginning of transcoding.

Reloading Korn Shell Scripts

The Reloading Korn shell scripts are generated using the `-g` option of the `file.sh` command. They are then (using the `-i` option) installed in:

```
$HOME/trf/reload/file/<configuration name>
```

Reloading Korn Shell Scripts for Migrating z/OS Files to Oracle Tables

The scripts are named: `loadtable-<logical file name>.ksh`

They contain a DDL creation phase, a transcoding (or loading) phase and a check phase. The different phases may be launched separately.

The execution of the scripts produces an execution log in `$MT_LOG/<logical file name>.log`

The following variables are set at the beginning of each script:

Listing 7-19 Reloading Table Script Variables

```

f="@ (c) Metaware:load-tables-ksh.pgm. $Revision: 1.14 $null"

echo "reloading ODSCF0B into ORACLE"

export DD_ENTREE=${DD_ENTREE:-${DATA_SOURCE}/ODSCF0B}

logfile=$MT_LOG/ODSCF0B.log

reportfile=${MT_LOG}/ODSCF0B.rpt

ddlfile=${DDL}/STFILEORA/ODSCF0B.sql

```

[...]

To change the file names, set the `DD_ENTREE` and `DD_SORTIE` variables before calling the script.

Various messages may be generated during the three execution phases of the scripts; explanations of these messages are listed in [Oracle Tuxedo Application Rehosting Workbench Messages](#).

On normal end, a return code of 0 is returned.

Creating Oracle DDL Phase

Oracle objects are created under SQLPLUS using: `${DDL}/STFILEORA/ODCSF0B.sql`

```
sqlplus $MT_DB_LOGIN >>$logfile 2>&1 <<!EOF
WHENEVER SQLERROR EXIT 3;
start ${ddlfile}
exit
!EOF
```

On normal termination the following message is displayed:

```
echo "Table(s) created"
```

Transcoding and Loading Phases

These steps launch the execution of the COBOL transcoding program associated with the file processed:

```
runb RELTABLE-ODCSF0B >> $logfile 2>&1
```

On normal termination the following message is displayed:

```
echo "File ${DD_ENTREE} successfully transcoded and reloaded into ORACLE"
```

Check Phase

This step verifies after the reloading that the reloaded Oracle table contains the same number of records as the records transferred from ZOS on target platform. If the number of records is different, an error message is produced:.. If the number of records is equal, this message is produced:

```
"Number of rows written in output file is equal to number calculated using
the log file: OK"
```

Target DDL

The ORACLE DDL is generated using the `-g` option of the `file.sh` command. They are then (using the `-i` option) installed in:

`$HOME/trf/SQL/file/<schema name>`

They are named: `<target file name>.ddl`.

The format used is:

```
WHENEVER SQLERROR CONTINUE;

DROP TABLE <target_table_name> CASCADE CONSTRAINTS;

WHENEVER SQLERROR EXIT 3;

CREATE TABLE <target_table_name> (
    <target_column_name> <column_data_type> <attribute>[, ...]
        CONSTRAINT <constraint_name> PRIMARY KEY (<target_column_name>)
        CONSTRAINT <fk_constraint_name> FOREIGN KEY
            (<target_column_name>)]
    );
```

Where:

<target_table_name>	Oracle table name.
<target_column_name>	Oracle column name.
<column_data_type>	Oracle data type (CHAR, VARCHAR2 or NUMBER).
<attribute>	NOT NULL when primary key.
<constraint_name>	Constraint name of primary key (PK_<Oracle table name>)
<fk_constraint_name>	Constraint name of foreign key (FK_<Oracle table name>_<parent_table_name>)

Listing 7-20 DDL Generation sql Example

```
WHENEVER SQLERROR CONTINUE;

DROP TABLE CUSTOMER CASCADE CONSTRAINTS;

WHENEVER SQLERROR EXIT 3;
```

```

CREATE TABLE CUSTOMER (
    VS_CUSTIDENT          NUMBER(6) NOT NULL,
    VS_CUSTLNAME          VARCHAR2(30),
    VS_CUSTFNAME          CHAR   (20),
    VS_CUSTADDRS          VARCHAR2(30),
    VS_CUSTCITY           CHAR   (20),
    VS_CUSTSTATE          CHAR   (2),
    VS_CUSTBDATE          NUMBER(8),
    VS_CUSTEMAIL          VARCHAR2(40),
    VS_CUSTPHONE          NUMBER(10),
    VS_FILLER             VARCHAR2(100),
    CONSTRAINT PK_CUSTOMER PRIMARY KEY (
        VS_CUSTIDENT)
);

```

Access Functions and Utility Programs

Access Functions

These access functions are generated using the `-g` option of `file.sh` and installed in `$HOME/trf/DML` using the `-i` and `-s` options.

Table 7-8 Access Functions

Access function	Role
RM_<logical file name>.pco	Relational access module to Oracle table that replaces the specified logical file name.
DL_<logical file name>.cbl	Download module of the specified logical file (function used by Oracle Tuxedo Application Runtime Batch).
UL_<logical file name>.cbl	Upload module of the specified logical file (function used by Oracle Tuxedo Application Runtime Batch).

Table 7-8 Access Functions

Access function	Role
<code>ASG_<logical file name>.cbl</code>	Optional module generated when there are multiple assigns. See Mapper File Attributes
<code>getfileinfo.cbl</code>	This program checks if the <logical file name>.rdb associated with the assign-name given as an input argument exists. This function is called by <code>ASG_<logical file name>.cbl</code> .
<code>init_all_files.cbl</code>	Initialize all variables used by relational module and logical module (function used by Oracle Tuxedo Application Runtime Batch).
<code>init_all_files_<configuration name>.cbl</code>	Initialize all variables used by relational module and <code>ASG_<logical file name></code> module for the configuration name listed; (function used by Oracle Tuxedo Application Runtime Batch).
<code>dml_locking.cbl</code>	This program manages locking for all configuration files (function used by Oracle Tuxedo Application Runtime Batch).
<code>close_all_files_<configuration name>.cbl</code>	This program closes all cursors opened in tables for the configuration listed and closes all files opened with logical accessor (function used by Oracle Tuxedo Application Runtime Batch).
<code>close_all_files.cbl</code>	This program closes all cursors opened in tables (function used by Oracle Tuxedo Application Runtime Batch).

Access Function Call Arguments

The `RM_<logical file name>.pco` and `ASG_<logical file name>.cbl` access functions use the following variables:

Table 7-9 Access Call Implemented Variables

Variable	Description/origin
Function code	Indicates the type of operation to execute, for example OPEN, WRITE, etc. The code is passed using the FILE-CODE-F variable of the MWFITECH copy file.
File open mode	A file can be opened in different modes: INPUT, OUTPUT, I O, EXTEND. The mode is passed using the FILE-OPEN-MODE variable of the MWFITECH copy file.
IO-STATUS	The IO-STATUS variable is linked to each file providing the execution status of the last relational module operation.
Record to transmit or receive	The record to transmit has an access function for write operations or access by key; the record to receive has a read access function. These are described in the LINKAGE SECTION.
Name of secondary key to use	For indexed files with secondary keys, and only for files with this organization, an extra variable is required to identify the secondary key to use for a START operation. The name of the secondary key is passed using the FILE-ALT-KEY-NAME variable of the MWFITECH copy file. For files without secondary keys, this argument is unnecessary.
Relative Key	For a relative file, the value of the relative key is passed to or from the access module using the FILE-REL-KEY variable of the MWFITECH copy file.

Listing 7-21 LINKAGE SECTION Structure

```

LINKAGE SECTION.

    01  IO-STATUS  PIC XX.

    COPY MWFITECH.

*      *COBOL Record Description

    01  VS-ODCSF0-RECORD.

        06  X-VS-CUSTIDENT.

```

```

        07 VS-CUSTIDENT          PIC 9(006).
06 VS-CUSTLNAME                  PIC X(030).
06 VS-CUSTFNAME                  PIC X(020).
06 VS-CUSTADDRS                  PIC X(030).
06 VS-CUSTCITY                   PIC X(020).
06 VS-CUSTSTATE                  PIC X(002).
06 X-VS-CUSTBDATE.

        07 VS-CUSTBDATE          PIC 9(008).
06 VS-CUSTBDATE-G                REDEFINES VS-CUSTBDATE.
11 X-VS-CUSTBDATE-CC.
        12 VS-CUSTBDATE-CC       PIC 9(002).
11 X-VS-CUSTBDATE-YY.
        12 VS-CUSTBDATE-YY       PIC 9(002).
11 X-VS-CUSTBDATE-MM.
        12 VS-CUSTBDATE-MM       PIC 9(002).
11 X-VS-CUSTBDATE-DD.
        12 VS-CUSTBDATE-DD       PIC 9(002).
06 VS-CUSTEMAIL                  PIC X(040).
06 X-VS-CUSTPHONE.

        07 VS-CUSTPHONE          PIC 9(010).
06 VS-FILLER                     PIC X(100).

PROCEDURE DIVISION USING IO-STATUS

                                MW-FILE-TECH

                                VS-ODCSF0-RECORD.

```

Call Arguments Used

OPEN

For all OPEN operations, the `FILE-CODE-F` variable should contain the key-word OPEN.

The `FILE-OPEN-MODE` variable should contain the type of OPEN to perform as follows:.

Table 7-10 Call Argument File Open Modes

Source	Target
OPEN INPUT filename1	INPUT => FILE-OPEN-MODE
OPEN OUTPUT filename1	OUTPUT => FILE-OPEN-MODE
OPEN I-O filename1	I-O => FILE-OPEN-MODE
OPEN EXTEND filename1	EXTEND => FILE-OPEN-MODE

CLOSE

For CLOSE operations, the `FILE-CODE-F` variable should contain the key-word CLOSE.

CLOSE-LOCK

For CLOSE LOCK operations, the `FILE-CODE-F` variable should contain the key-word CLOSE-LOCK.

DELETE

Depending on the file access mode, the DELETE operation is either the current record or the one indicated by the file key.

The corresponding function code is indicated as follows:

Table 7-11 Call Argument Delete Modes

Access	Source	Target
Sequential	DELETE filename1	DELETE-CUR => FILE-CODE-F
Random or dynamic	DELETE filename1	DELETE-KEY => FILE-CODE-F

READ

The function code depends on the file access mode and the type of read required: sequential read, read primary key or read secondary key.

Table 7-12 Read Operation Values Depending on Arguments

Access	Source	Target
Sequential	READ filename1 [NEXT]	READ-NEXT => FILE-CODE-F
Random	READ filename1	READ-KEY => FILE-CODE-F
Dynamic	READ filename1 NEXT	READ-NEXT => FILE-CODE-F
	READ filename1	READ-KEY => FILE-CODE-F
	READ filename1 PREVIOUS	READ-PREV => FILE-CODE-F
If <i>DataName1</i> is a variable corresponding to the keyAltKey1	READ filename1 KEY <i>DataName1</i>	READ-ALT-KEY => FILE-CODE-F "AltKey1" => FILE-ALT-KEY-NAME
<i>DataName1</i> represents the relative key	READ filename1 KEY <i>DataName1</i>	READ-REL-KEY => FILE-CODE-F "RelKeyVar" => FILE-REL-KEY

Note: If the INTO clause is found, a MOVE operation is added after the call in order to set the value of the indicated field.

REWRITE

The function code depends on the file access mode and the type of read required: sequential read, read primary key or read secondary key

Table 7-13 Rewrite Operation Values Depending on Arguments

Access	Source	Target
Sequential	REWRITE RecName1	REWRITE-CUR => FILE-CODE-F
Random or dynamic	REWRITE RecName1	REWRITE-KEY => FILE-CODE-F

Note: If the FROM clause is found, a MOVE operation is added before the call in order to set the value of the indicated field.

START

Whether the file is relative, indexed, with or without secondary key, the function code depends on the exact type of start.

Table 7-14 Rewrite Operation Values Depending on Arguments

When	Source	Target
	START file1	START-EQUAL => FILE-CODE-F
DataName1 represents the relative key or Primary Key of file1	START file1 KEY { EQUAL = EQUALS } DataName1	START-EQUAL => FILE-CODE-F
	START file1 KEY { EXCEEDS > GREATER } DataName1	START-SUP => FILE-CODE-F
	START file1 KEY { NOT LESS GREATER OR EQUAL NOT < >= } DataName1	START-SUPEQ => FILE-CODE-F
	START file1 KEY {< LESS } DataName1	START-INF => FILE-CODE-F
	START file1 KEY { NOT GREATER LESS OR EQUAL NOT > <= } DataName1	START-INFEQ => FILE-CODE-F

Table 7-14 Rewrite Operation Values Depending on Arguments

When	Source	Target
DataName1 is a variable corresponding to the AltKey1 key	START file1 KEY { EQUAL = EQUALS } DataName1	AltKey1 => FILE-ALT-KEY-NAME START-ALT-EQUAL => FILE-CODE-F
	START file1 KEY { EXCEEDS > GREATER } DataName1	AltKey1 => FILE-ALT-KEY-NAME START-ALT-SUP => FILE-CODE-F
	START file1 KEY { NOT LESS GREATER OR EQUAL NOT < >= } DataName1	AltKey1 => FILE-ALT-KEY-NAME START-ALT-SUPEQ => FILE-CODE-F
	START file1 KEY { < LESS } DataName1	AltKey1 => FILE-ALT-KEY-NAME START-ALT-INF => FILE-CODE-F
	START file1 KEY { NOT GREATER LESS OR EQUAL NOT > <= } DataName1	AltKey1 => FILE-ALT-KEY-NAME START-ALT-INFEQ => FILE-CODE-F

WRITE

The function code depends on the file access mode and the type of read required: sequential read, read primary key or read secondary key

Table 7-15 Write Operation Values Depending on Arguments

Access	Source	Target
Sequential	WRITE RecName1	WRITE-SEQ => FILE-CODE-F
Random or dynamic	WRITE RecName1	WRITE-KEY => FILE-CODE-F

Note: If the FROM clause is found, a MOVE operation is added before the call in order to set the value of the indicated field.

Copy Files to Be Implemented

The following copy files are used by certain access functions. They should be placed in the directory: `< installation platform>/fixed-copy/` during the installation of the Rehosting Workbench:

- `MW-PARAM-TRACE-VAR.cpy`
- `MW-PARAM-TRACE.cpy`
- `MW-PARAM-GETFILEINFO-VAR.cpy`
- `MW-PARAM-GETFILEINFO.cpy`
- `MW-PARAM-ERROR-VAR.cpy`
- `MW-PARAM-ERROR.cpy`
- `MW-PARAM-DML-LOCKING.cpy`
- `MWFITECH.cpy`
- `ERROR-SQLCODE.cpy`

Korn Shell Utilities

These KSH scripts are generated using the `-g` option of `file.sh` and then installed in `$HOME/trf/SQL/file/<configuration name>` using the `-i` option. When necessary, they are used by Oracle Tuxedo Application Runtime Batch.

Table 7-16 Korn Shell Utilities

Korn shell script name	Role
<code>cleantable-<logical file name>.ksh</code>	Script file that CLEANs all tables associated with this file.
<code>createtable-<logical file name>.ksh</code>	Script file that CREATEs all table, constraint and indexes associated with this file.
<code>droptable-<logical file name>.ksh</code>	Script file that DROPs all tables associated with this file.

Table 7-16 Korn Shell Utilities

Korn shell script name	Role
<code>ifemptytable-<logical file name>.ksh</code>	Script file that checks if all tables are empty.
<code>ifexisttable-<logical file name>.ksh</code>	Script file that checks if all tables exist.

Oracle Tuxedo Application Runtime for CICS Configuration Files

The `desc.vsam` and `envfile_tux` files are generated in the `$HOME/trf/config/tux/` directory when VSAM files are migrated to Oracle tables. They are used by Oracle Tuxedo Application Runtime CICS.

COBOL and JCL Conversion Guide Files

These files are generated using the `-s` option of the `file.sh` command.

This file is used by the Rehosting Workbench [COBOL Converter](#) and JCL Converter to rename object names.

Table 7-17 Conversion File Names

<code>File-in-table-<configuration name></code>	Used by the Rehosting Workbench JCL converter
<code>File-in-table.txt</code>	Used by the Rehosting Workbench JCL converter
<code>Conv-ctrl.txt</code>	Used by the Rehosting Workbench COBOL converter
<code>Conv-ctrl-<configuration name></code>	Used by the Rehosting Workbench COBOL converter

.rdb Files

These files are created when VSAM files are converted to Oracle tables. They are used by Oracle Tuxedo Application Runtime Batch to bridge the technical differences between the z/OS file on the source platform and the corresponding Oracle table on the target platform.

The files are generated in: `$HOME/trf/data`

They are named: `<source platform physical file name>.rdb`

The files contain two lines described in the next section.

Parameters and Syntax

```
${DATA}/<source platform physical file name> <max> <org> <form> UL_<logical  
file name> <asgn_in> DL_<logical file name> <asgn_out> RM_<logical file  
name> <target table name> ${DDL}/<configuration name>/cleantable-<target  
table name>.ksh ${DDL}/<configuration name>/droptable-<target table  
name>.ksh ${DDL}/<configuration name>/createtable-<target table name>.ksh  
${DDL}/<configuration name>/ifemptytable-<target table name>.ksh  
${DDL}/<configuration name>/ifexisttable-<target table name>.ksh
```

```
IDX_KEY <column name> <n m>
```

```
REL_KEY - <m>
```

Table 7-18 .rdb file Parameters

Parameter	Description
First Line:	
<source platform physical file name>	Physical file name
<max>	Maximum Record Size (in COBOL description).
<org>	File organization: <ul style="list-style-type: none">• IDX: indexed without alternate key• IDX_ALT: indexed with alternate key(s)• SEQ: sequential• REL: relative

Table 7-18 .rdb file Parameters

Parameter	Description
<form>	Record format: <ul style="list-style-type: none"> • FIX: fixed file • VAR:<min> variable file with minimal size. If <min> is missing, minimal size will be 1.
UL_<logical file name>	Uploading component name used by Oracle Tuxedo Application Runtime Batch.
<asgn_in>	Assign file name used by the uploading component.
DL_<logical file name>	Downloading component name used by Runtime.
<asgn_out>	Assign file name used by the downloading component.
RM_<logical file name>	Relational module name.
<target table name>	Name of the first table name (master table name or first table name for multi-record).
\${DDL}/<configuration name/cleantable-<target table name>.ksh	Name of the script file that CLEANs all tables associated with this file.
\${DDL}/<configuration name/droptable-<target table name>.ksh	Name of the script file that DROPs all tables associated with this file.
\${DDL}/<configuration name>/createtable-<target table name>.ksh	Name of the script file that CREATEs all tables associated with this file and their objects (constraints, indexes).
\${DDL}/<configuration name>/ifemptytable-<target table name>.ksh	Name of the script file that checks if all tables are empty.
\${DDL}/<configuration name>/ifexisttable-<target table name>.ksh	Name of the script file that checks if all tables exist.
Second Line for indexed file and indexed with alternate key file only:	
IDX_KEY	Constant.

Table 7-18 .rdb file Parameters

Parameter	Description
<VS-column name>	Indexed key name (group zone name or elementary field name as described in COBOL description).
<n m>	<ul style="list-style-type: none">n: offset of the indexed key (in COBOL description).m: length of the indexed key (in COBOL description).
Second Line for relative file:	
REL_KEY	Constant.
-	Constant.
<m>	<ul style="list-style-type: none">m: length of the relative key (in COBOL description).

Example of .rdb File

The following example is generated when migrating an indexed VSAM file to an Oracle table. On the source platform, the VSAM file is named: PJ01AAA.SS.VSAM.CUSTOMER

Listing 7-22 .rdb indexed VSAM Example

```
${DATA}/PJ01AAA.SS.VSAM.CUSTOMER 266 IDX FIX UL_ODCSF0B ENTREE DL_ODCSF0B
SORTIE RM_ODCSF0B CUSTOMER ${DDL}/STFILEORA/cleantable-ODCSF0B.ksh
${DDL}/STFILEORA/droptable-ODCSF0B.ksh
${DDL}/STFILEORA/createtable-ODCSF0B.ksh
${DDL}/STFILEORA/ifemptytable-ODCSF0B.ksh
${DDL}/STFILEORA/ifexisttable-ODCSF0B.ksh

IDX_KEY VS-CUSTIDENT 1 6
```

Execution Reports

file.sh creates different execution reports depending on the options chosen. In the following examples the following command is used:

```
file.sh -gmi $HOME/trf STFILEORA
```

Listing 7-23 Messages Produced when Using the options -g with file.sh (step 1)

```
*-----
#####
Control of configuration STFILEORA
#####
Control of templates
OK: Use Default Templates list file
      File name is
/Qarefine/release/M3_L3_6/convert-data/default/file/file-templates.txt
#####
Control of Mapper
#####
COMPONENTS GENERATION

CMD : /Qarefine/release/M3_L3_6/scripts/launch file-converter  -s
/home2/wkb4/param/system.desc -mf /home2/wkb4/tmp/mapper-STFILEORA.re.tmp
-dmf /home2/wkb4/param/file/Datamap-STFILEORA.re -td /home2/wkb4/tmp -tmps
/home2/wkb4/tmp/file-templates-STFILEORA.tmp -target-sgbd oracle11
-target-os unix -varchar2 29 -print-ddl -print-dml -abort

MetaWorld starter

Loading lib: /Qarefine/release/M3_L3_6/Linux64/lib64/localext.so
(funcall LOAD-THE-SYS-AND-APPLY-DMAP-AND-MAPPER)

FILE-0092: *File-Converter*: We are in BATCH mode

FILE-0087: * Comand line arguments: begining of analyze

FILE-0088: * recognized argument -s value: /home2/wkb4/param/system.desc

FILE-0088: * recognized argument -mf value:
/home2/wkb4/tmp/mapper-STFILEORA.re.tmp
```

```

FILE-0088: * recognized argument -dmf value:
/home2/wkb4/param/file/Datamap-STFILEORA.re
FILE-0088: * recognized argument -td value: /home2/wkb4/tmp
FILE-0088: * recognized argument -tmps value:
/home2/wkb4/tmp/file-templates-STFILEORA.tmp
FILE-0088: * recognized argument -target-sgbd value: oracle11
FILE-0088: * recognized argument -target-os value: unix
FILE-0088: * recognized argument -varchar2 value: 29
FILE-0089: * recognized argument -print-ddl
FILE-0089: * recognized argument -print-dml
FILE-0089: * recognized argument -abort
FILE-0091: * End of Analyze
FILE-0094: * Parsing mapper file /home2/wkb4/tmp/mapper-STFILEORA.re.tmp
...
FILE-0095: * Parsing data-map file
/home2/wkb4/param/file/Datamap-STFILEORA.re ...
FILE-0096: * Parsing system description file /home2/wkb4/param/system.desc
...
Warning! OS clause is absent, assuming OS is IBM
Current OS is IBM-MF
Loading /home2/wkb4/source/symtab-STFILEORA.pob at 16:38:42... done at
16:38:42
Build-Symtab-DL1 #1<a SYMTAB-DL1>
... Postanalyze-System-RPL...
sym=#2<a SYMTAB>
PostAnalyze-Common #2<a SYMTAB>
    0 classes
    0 classes
    0 classes

```

```

0 classes

1 classes

13 classes

Loading /home2/wkb4/source/BATCH/pob/RSSABB01.cbl.shrec...
Loading /home2/wkb4/source/COPY/pob/ODCSF0.cpy.cdm...
Loading /home2/wkb4/source/COPY/pob/ODCSF0B.cpy.cdm...
Loading /home2/wkb4/source/COPY/pob/ODCSFU.cpy.cdm...

FILE-0001: * Point 1 !!

FILE-0002: * Point 2 !!

FILE-0010: * Parsing file /home2/wkb4/source/COPY/ODCSF0.cpy ...
*Parsed 22 lines*

FILE-0010: * Parsing file /home2/wkb4/source/COPY/MW_SYSOUT.cpy ...
*Parsed 8 lines*

FILE-0010: * Parsing file /home2/wkb4/source/COPY/ODCSFU.cpy ...
*Parsed 24 lines*

FILE-0010: * Parsing file /home2/wkb4/source/COPY/ODCSF0B.cpy ...
*Parsed 22 lines*

FILE-0003: * Point 3 !!

FILE-0004: * Point 4 !!

FILE-0005: * Point 5 !!

FILE-0052: * loading pob file
/Qarefine/release/M3_L3_6/convert-data/templates/file/unloading/jcl-unload
-MVS-REPRO.pgm.pob

FILE-0085: * Expanding
/Qarefine/release/M3_L3_6/convert-data/templates/file/unloading/jcl-unload
-MVS-REPRO.pgm ...

FILE-0054: * Writing MW-SYSOUT.jclunload

FILE-0054: * Writing ODCSFU.jclunload

```

```

FILE-0054: * Writing ODCSF0Q.jclunload

[...]

FILE-0052: * loading pob file
/Qarefine/release/M3_L3_6/convert-data/templates/file/dml/generate-post-pr
ocess.pgm.pob

FILE-0085: * Expanding
/Qarefine/release/M3_L3_6/convert-data/templates/file/dml/generate-post-pr
ocess.pgm ...

FILE-0054: * Writing post-process-file.sh

FILE-0053: * Parsing template file
/Qarefine/release/M3_L3_6/convert-data/default/file/file-move-assignment.
pgm

FILE-0085: * Expanding
/Qarefine/release/M3_L3_6/convert-data/default/file/file-move-assignment.
pgm ...

FILE-0054: * Writing file-move-assignment.lst

Rest in peace, Refine...

*-----
Generated components are in /home2/wkb4/tmp/Template/STFILEORA
(Optionaly in /home2/wkb4/tmp/SQL/STFILEORA)
*-----

```

Listing 7-24 Messages Produced when Using the Options -m with file.sh (step 2)

```

#####
  FORMATTING COBOL LINES
#####
  CHANGE ATTRIBUTE TO KSH or SH scripts
*-----

```

Components are modified into /home2/wkb9/tmp directory

*=====

Messages produced using the -i option in file.sh (step 3)

INSTALL COMPONENTS INTO SPECIFIC DIRECTORY USING file-move-assignment.lst

=====

==_PJ01AAA.SS.VSAM.CUSTOMER_==

Copied <Templates>:ODCSF0B.jclunload to
<td>/unloading/file/STFILEORA/ODCSF0B.jclunload

Copied <Templates>:loadtable-ODCSF0B.ksh to
<td>/reload/file/STFILEORA/loadtable-ODCSF0B.ksh

Copied <Templates>:RELTABLE-ODCSF0B.pco to
<td>/reload/file/STFILEORA/RELTABLE-ODCSF0B.pco

Copied <Templates>:ASG_ODCSF0B.cbl to <td>/DML/ASG_ODCSF0B.cbl

Copied <Templates>:RM_ODCSF0B.pco to <td>/DML/RM_ODCSF0B.pco

Copied <Templates>:DL_ODCSF0B.cbl to <td>/DML/DL_ODCSF0B.cbl

Copied <Templates>:UL_ODCSF0B.cbl to <td>/DML/UL_ODCSF0B.cbl

Copied <Templates>:PJ01AAA.SS.VSAM.CUSTOMER.rdb to
<td>/data/PJ01AAA.SS.VSAM.CUSTOMER.rdb

Copied <SQL>:ODCSF0B.sql to <td>/SQL/file/STFILEORA/ODCSF0B.sql

Copied <Templates>:cleantable-ODCSF0B.ksh to
<td>/SQL/file/STFILEORA/cleantable-ODCSF0B.ksh

Copied <Templates>:droptable-ODCSF0B.ksh to
<td>/SQL/file/STFILEORA/droptable-ODCSF0B.ksh

Copied <Templates>:createtable-ODCSF0B.ksh to
<td>/SQL/file/STFILEORA/createtable-ODCSF0B.ksh

Copied <Templates>:ifemptytable-ODCSF0B.ksh to
<td>/SQL/file/STFILEORA/ifemptytable-ODCSF0B.ksh

Copied <Templates>:ifexisttable-ODCSF0B.ksh to
<td>/SQL/file/STFILEORA/ifexisttable-ODCSF0B.ksh

[...]

```

=====

==_PJ01AAA.SS.QSAM.CUSTOMER.REPORT_==

Copied <Templates>:loadfile-MW-SYSOUT.ksh to
<td>/reload/file/STFILEORA/loadfile-MW-SYSOUT.ksh

Copied <Templates>:RELFILE-MW-SYSOUT.cbl to
<td>/reload/file/STFILEORA/RELFILE-MW-SYSOUT.cbl

=====

==_PJ01AAA.SS.QSAM.CUSTOMER.UPDATE_==

Copied <Templates>:loadfile-ODCSFU.ksh to
<td>/reload/file/STFILEORA/loadfile-ODCSFU.ksh

Copied <Templates>:RELFILE-ODCSFU.cbl to
<td>/reload/file/STFILEORA/RELFILE-ODCSFU.cbl

=====

==_PJ01AAA.SS.QSAM.CUSTOMER_==

Copied <Templates>:loadfile-ODCSF0.ksh to
<td>/reload/file/STFILEORA/loadfile-ODCSF0.ksh

Copied <Templates>:RELFILE-ODCSF0.cbl to
<td>/reload/file/STFILEORA/RELFILE-ODCSF0.cbl

=====

Copied <Templates>:close_all_files_STFILEORA.cbl to
<td>/DML/close_all_files_STFILEORA.cbl

Copied <Templates>:init_all_files_STFILEORA.cbl to
<td>/DML/init_all_files_STFILEORA.cbl

Copied <Templates>:reload-files.txt to
<td>/reload/file/STFILEORA/reload-files.txt

Copied <fixed-components>:getfileinfo.cbl to <td>/DML/getfileinfo.cbl

Copied <fixed-components>:MWFITECH.cpy to <td>/fixed-copy/MWFITECH.cpy

Copied <fixed-components>:MW-PARAM-ERROR.cpy to
<td>/fixed-copy/MW-PARAM-ERROR.cpy

```

```

Copied    <fixed-components>:MW-PARAM-ERROR-VAR.cpy to
<td>/fixed-copy/MW-PARAM-ERROR-VAR.cpy

Copied    <fixed-components>:MW-PARAM-TRACE.cpy to
<td>/fixed-copy/MW-PARAM-TRACE.cpy

Copied    <fixed-components>:MW-PARAM-TRACE-VAR.cpy to
<td>/fixed-copy/MW-PARAM-TRACE-VAR.cpy

Copied    <fixed-components>:MW-PARAM-GETFILEINFO.cpy to
<td>/fixed-copy/MW-PARAM-GETFILEINFO.cpy

Copied    <fixed-components>:MW-PARAM-GETFILEINFO-VAR.cpy to
<td>/fixed-copy/MW-PARAM-GETFILEINFO-VAR.cpy

Copied    <fixed-components>:MW-PARAM-DML-LOCKING.cpy to
<td>/fixed-copy/MW-PARAM-DML-LOCKING.cpy

Copied    <fixed-components>:ERROR-SQLCODE.cpy to
<td>/fixed-copy/ERROR-SQLCODE.cpy

Copied    <fixed-components>:RunSqlLoader.sh to
<td>/reload/bin/RunSqlLoader.sh

Copied    <fixed-components>:CreateReportFromMVS.sh to
<td>/reload/bin/CreateReportFromMVS.sh

=====

Dynamic_configuration

Copied    <Templates>:File-in-table-STFILEORA to
/home2/wkb9/param/dynamic-config/File-in-table-STFILEORA

Copied    <Templates>:../../Conv-ctrl-STFILEORA to
/home2/wkb9/param/dynamic-config/Conv-ctrl-STFILEORA

=====

post-process

executed <Templates>:post-process-file.sh

/home2/wkb9/param/dynamic-config/Conv-ctrl-STFILEORA treated

=====

Number of copied files:      37

```

```
Number of executed scripts: 1
```

```
Number of ignored files:    0
```

```
#####
```

```
*-----
```

```
Components are copied into /home2/wkb9/trf directory
```

```
*-----
```

Detailed Processing

This section describes the [Command-Line Syntax](#) used by the File-to-Oracle Converter, and the [Process Steps](#) summary.

The processes required on the source and target platforms concern:

- [Configuring the Environments and Installing the Components](#),
- [Unloading Data](#),
- [Transferring the Data](#),
- [Reloading the Data](#),
- [Checking the Transfers](#),

Command-Line Syntax

file.sh

Name

file.sh - generate file migration components.

Synopsis

```
file.sh [ [-g] [-m] [-i <installation directory>] <configuration name> | -s  
<installation directory> (<configuration name1>,<configuration name2>,...)  
]
```

Description

`file.sh` generates the Rehosting Workbench components used to migrate z/OS files to Oracle databases.

Options

Generation Options

-g <configuration name>

Triggers the generation, for the configuration indicated, of the unloading and loading components in \$TMPPROJECT. This generation depends on the information found in the configuration files.

Modification Options

-m <configuration name>

Makes the generated SHELL scripts executable. COBOL programs are adapted to Micro Focus COBOL fixed format. When present, the shell script described in [File Modifying Generated Components](#) is executed.

Installation Option

-i <installation directory> <configuration name>

Places the components in the installation directory. This operation uses the information located in the [file-move-assignation.pgm](#) file.

Final Option

-s <installation directory> (<configuration name 1>,<configuration name 2>,...)

Enables the generation of the COBOL and JCL converter configuration files and DML utilities. These generated files take all of the unitary files of the project.

All configuration files are created in \$PARAM/dynamic-config and DML files in <trf>/DML directory.

Example

```
file.sh -gmi $HOME/trf FTFIL001
```

Unitary Usage Sequence

If the `file.sh` options are used one at a time, they should be used in the following order:

1. => -g
2. => -m
3. => -i
4. => -s (should be executed once steps 1 to 3 have been executed for all configurations).

Process Steps

Configuring the Environments and Installing the Components

This section describes the preparation work on the source and target platforms.

Installing the Unloading Components Under z/OS

The components used for the unloading (generated in `$HOME/trf/unload/file`) should be installed on the source z/OS platform (the generated JCL may need adapting to specific site constraints including JOB cards, library access paths and access paths to input and out put files).

Installing the Reloading Components on the Target Platform

The components used for the reloading (generated in `$HOME/trf/reload/file`) should be installed on the target platform.

The following environment variables should be set on the target platform:

Table 7-19 Target Platform Environment Variables

Variable	Value
DATA_SOURCE	The name of the directory containing the unloaded files transferred from z/OS to be reloaded into Oracle tables.
BIN	The location of the generic reload and control scripts (<code>\$HOME/trf/reload/bin</code>).

Table 7-19 Target Platform Environment Variables

Variable	Value
TMPPROJECT	The temporary directory.
MT_LOG	Directory to contain execution logs.
DATA_TRANSCODE	Temporary directory used by the file binary data transcoding script (contains temporary files in ASCII format).
DDL	The location of SQL scripts used to create Oracle objects: (<code>\$HOME/trf/SQL/file/<configuration name></code>).
NLS_LANG	Set according to the instructions in the Oracle Database Globalization Support Guide.
PATH	This UNIX/Linux variable has to contain the directory of the Oracle Tuxedo Application Runtime for Batch utilities

In addition, the following variable should be set according to the information in the Oracle Tuxedo Application Rehosting Workbench Installation Guide:

- MT_DB_LOGIN.

Compiling COBOL Transcoding Programs

The COBOL transcoding programs should be compiled using the options specified in [Compiler Options](#).

Compiling these programs requires the presence of a copy of `CONVERTMW.cpy` adapted to the project.

Unloading Data

To unload each file, a JCL using the IBM IDCAMS REPRO utility is executed. The IDCAMS REPRO utility creates two files for each file:

- a data file,
- a log file,

These unloading JCLs are named `<logical filename>.jclunload`

A return code of 0 is sent on normal job end.

Transferring the Data

The unloaded data files should be transferred between the source z/OS platform and the target UNIX/Linux platform in binary format using the file transfer tools available at the site (CFT, FTP, ...).

The files transferred to the target UNIX/Linux platform should be stored in the `$DATA_SOURCE` directory.

Reloading the Data

The scripts enabling the transcoding and reloading of data are generated in the directory:

```
$HOME/trf/reload/file/<configuration name>/
```

For a file-to-Oracle conversion, the format of the script names is:

```
loadtable-<logical file name>.ksh
```

Transcoding and Reloading Command for Tables

Name

loadtable transcode and reload data to table.

Synopsis

```
loadtable-<logical file name>.ksh [-t] [-l] [-c: <method>]
```

Options

-t

Transcode the file.

-l

Reload the file

-c ftp:<...>:<...>

Implement the verification of the transfer (see [Checking the Transfers](#)).

Checking the Transfers

This check uses the following option of the `loadtable-<logical file name>.ksh`

```
-c ftp:<name of transferred physical file>:<name of FTP log under UNIX>
```

This option verifies, after the reloading, that the physical file transferred from z/OS and the Oracle table reloaded on the target platform contains the same number of records. This check is performed using the FTP log and the execution report of the reloading program. If the number of records is different, an error message is produced.

File-to-Db2/luw (udb) Converter

This chapter describes the Rehosting Workbench File-to-Db2/luw (udb) Converter used to migrate files from the source platform (z/OS) to Db2/luw (udb) tables (luw as Linux, UNIX, Windows, old name is udb), and describes the migration tools that are generated. The conversion is performed in the context of other components translated or generated by the other Oracle Tuxedo Application Rehosting Workbench tools.

Several configuration files need to be set, see [Description of the Configuration Files](#), before launching the conversion process.

This section continues from the [File Converter: Introduction](#) section and contains several links to this section for example the different objects generated are described in [List of the Input Components](#). Some objects are only generated when migrating VSAM files to Db2/luw tables (SQB programs, SQL files, relational module, logical module, utilities and configuration files for JCL and COBOL conversion).

Overview of the File-to-Db2/luw (udb) Converter

Purpose

The purpose of this section is to describe precisely all the features of the Rehosting Workbench File-to-Db2/luw (udb) Converter tools including:

- Inventory of files to migrate.
- Detailed description of Db2/luw (udb) tables on the target platform for each file.

- Description of the different commands to be used with the File-to-Db2/luw (udb) Converter.
- Description of the data unloading options on the source platform.
- Description of the data loading options on the target platform.

Structure

- [Overview of the File-to-Db2/luw \(udb\) Converter.](#)
- [Description of the Input Components](#) including [Description of the Configuration Files.](#)
- [Description of the Output Files](#) including the [Generated Objects.](#)
- [Detailed Processing](#) including the [Command-Line Syntax.](#)
- For messages, see [File Converter Messages.](#)

See Also

The conversion of data is closely linked to the conversion of COBOL programs, see:

- [COBOL Converter](#)

This section describes common data conversion usages:

- [File Converter: Introduction](#)

File Organizations Process

When migrating files from a z/OS source platform to a target platform, the first question to ask, when VSAM is concerned, is whether to keep a file or migrate the data to an Db2/luw (udb) table.

Migrating to Db2/luw (udb) Table on the Target Platform

KSDS, RRDS and ESDS VSAM files can be migrated into Db2/luw (udb) tables.

To make this work, the first task is to list all of the VSAM files to be migrated, and then identify those files that should be converted to Db2/luw (udb) tables. For example, permanent files to be later used via Oracle or files that needs locking at the record level.

Oracle Tuxedo Application Rehosting Workbench Configuration Name

A configuration name is related to a set of files to be converted. Each set of files can be freely assembled. Each configuration could be related to a different application for example, or a set of files required for Oracle Tuxedo Application Runtime. The set of files can contain both files or Db2/luw (udb) tables targets.

VSAM Files Becoming Db2/luw (udb) Table

Specific Migration Rules Applied

- Each table name is stipulated in the `mapper-<configuration name>.re` file using the table name clause.
- Each elementary field name contained in a copy description of the file becomes a column in an Db2/luw (udb) table. Hyphens (-) are replaced by underscore (_) characters.
- For sequential VSAM files (VSAM ESDS): the Rehosting Workbench adds a technical column: *_SEQ_NUM NUMERIC.

This column is incremented each time a new line is added to the table and becomes the primary key of the table.

- For relative VSAM files (VSAM RRDS): the Rehosting Workbench adds a technical column: *_RELATIVE_NUM.

The size of the column is deduced from the information supplied in the Datamap parameter file; the column becomes the primary key of the table.

The column:

- is incremented when a sequential write is made to the table, and the relative key is zero.
 - contains a relative key when the relative key is not zero.
- For indexed VSAM files (VSAM KSDS): the Rehosting Workbench does not add a technical column unless duplicate keys are accepted; the primary key of the VSAM file becomes the primary key of the table.

Rules Applied to Picture Clauses

The following rules are applied to COBOL Picture clauses when migrating data from VSAM files to Db2/luw (udb) tables:

Table 8-1 Picture Clause Re-engineering

COBOL Picture	Oracle format
PIC 9 (...)	NUMERIC(...)
COMP-1	NUMERIC(8)
COMP-2	NUMERIC(16)
PIC X(...)	Becomes CHAR if length <= 255 Becomes VARCHAR if length > 255 If the parameter file:char_limit_until_varchar is set in the db-param.cfg file, it takes precedence over the above rule.

Environment Variables

Before starting the process of migrating data two environment variables should be set:

- `export TMPPROJECT=/ $HOME /tmp`
Indicates the location to store temporary objects generated by the process.
- `export PARAM=/ $HOME /param`
Indicates the location where the configuration files required by the process are stored.

Description of the Input Components

File Locations

Location of file.sh

The file.sh tool is located in the directory:

```
$REFINEDIR/convert-data/
```

Location of db-param.cfg File

The db-param.cfg configuration file is located in the directory given in the variable:

```
$PARAM
```

Description of the Configuration Files

This section lists the files and their parameters that can be used to control the migration of z/OS files to Db2/luw (udb) table.

db-param.cfg

This file should be created in the directory indicated by the \$PARAM directory:

```
$PARAM/db-param.cfg
```

Listing 8-1 db-param.cfg Template

```
#
# This configuration file is used by FILE & RDBMS converter
# Lines beginning by "#" are ignored
# write information in lower case
#
# common parameters for FILE and RDBMS
#
# source information is written into system descriptor file (DBMS=,
DBMS-VERSION=)
target_rdbms_name:<target_rdbms_name>
target_rdbms_version:<target_rdbms_version>
target_os:<target_os>
#
# specific parameters for FILE to RDBMS conversion
file:char_limit_until_varchar:<char_limit>
```

Parameters and Syntaxes

Table 8-2 db-param.cfg Parameters

Parameter	Description	Value
General Parameters		
<target_rdbms_name>	Name of target RDBMS	udb
<target_rdbms_version>	Version of target RDBMS	9
<target_os>	Name of target operating system	unix or linux
Specific file-to-oracle conversion parameters		
<char_limit>	<ul style="list-style-type: none">For a field size <= 255, a COBOL alphanumeric field is migrated on Db2/luw (udb) in CHARFor a field size > 255 it is migrated in VARCHAR, except if the parameter <code>file:char_limit_until_varchar</code> is used. <p>This parameter indicates the maximum length of a COBOL alphanumeric (PIC X) field before the field will be transformed into an Db2/luw (udb) VARCHAR data type.</p> <p>If the parameter contains: <code>file:char_limit_until_varchar:29</code></p> <p>Then, fields longer than 29 characters will become VARCHAR, fields shorter than 30 characters will become CHAR fields.</p>	

File Modifying Generated Components

The generated components may be modified using a project's own scripts. These scripts (sed, awk, perl,...) should be placed in:

```
$PARAM/file/file-modif-source.sh
```

When present, this file will be automatically executed at the end of the generation process. It will be called using the <configuration name> as an argument.

file-template-db2luw.txt

This file is put in place during the installation of the Rehosting Workbench, it contains the templates that perform the generation of the different migration tools. The file is located in:

```
$REFINEDIR/convert-data/default/file/file-templates-db2luw.txt
```

Listing 8-2 file-template-db2luw.txt

```
% Unloading all File *****
% All SAM file were transfered using FTP/Binary
% VSAM unloaded step:
#VAR:TEMPLATES#/unloading/jcl-unload-MVS-REPRO.pgm
#VAR:TEMPLATES#/unloading/jcl-unload-GDG-MVS-REPRO.pgm
%
% To create a specific template, copy this template into :
% -- #VAR:PARAM#/file/specific-templates/unloading/jcl-unload-customer.pgm
%
% Loading *****
#VAR:TEMPLATES#/loading/file-reload-files-txt.pgm
% Loading File to File *****
#VAR:TEMPLATES#/loading/unix-file/reload-files-ksh.pgm
#VAR:TEMPLATES#/loading/unix-file/reload-GDG-files-ksh.pgm
#VAR:TEMPLATES#/loading/unix-file/reload-mono-rec.pgm
% Loading File to Oracle *****
#VAR:TEMPLATES#/loading/unix-db2luw/load-tables-ksh-db2luw.pgm
#VAR:TEMPLATES#/loading/unix-db2luw/rel-mono-rec-db2luw.pgm
#VAR:TEMPLATES#/dml/unix-db2luw/clean-tables-ksh-db2luw.pgm
#VAR:TEMPLATES#/dml/unix-db2luw/drop-tables-ksh-db2luw.pgm
#VAR:TEMPLATES#/dml/unix-db2luw/create-tables-ksh-db2luw.pgm
```

```

#VAR:TEMPLATES#/dml/unix-db2luw/ifempty-tables-ksh-db2luw.pgm
#VAR:TEMPLATES#/dml/unix-db2luw/ifexist-tables-ksh-db2luw.pgm
%
% Generate Logical & Relational Module *****
#VAR:TEMPLATES#/dml/module/open-multi-assign-free.pgm
#VAR:TEMPLATES#/dml/module/unix-db2luw/open-mono-rec-idx-db2luw.pgm
#VAR:TEMPLATES#/dml/module/unix-db2luw/open-mono-rec-seq-db2luw.pgm
#VAR:TEMPLATES#/dml/module/unix-db2luw/open-mono-rec-rel-db2luw.pgm
%
% and utilities *****
#VAR:TEMPLATES#/dml/module/decharge-mono-rec.pgm
#VAR:TEMPLATES#/dml/module/recharge-table.pgm
#VAR:TEMPLATES#/dml/module/close-all-files.pgm
#VAR:TEMPLATES#/dml/module/init-all-files.pgm
%
% configuration file for translation and runtime *****
#VAR:TEMPLATES#/dml/generate-config-wb-translator-jcl.pgm
#VAR:TEMPLATES#/dml/generate-rdb-txt.pgm
%
% included file to include into modified-components
#VAR:TEMPLATES#/dml/include-modified-components.pgm
%
% *****
% MANDATORY
% : used just after the generation
#VAR:TEMPLATES#/dml/generate-post-process.pgm
% : used when using -i arguments

```

```
#VAR:DEFAULT#/file-move-assignment-db2luw.pgm
```

```
%
```

Note: This file contains both File-to-File and File-to-Db2/luw (udb) migration parameters.

When required, another version of the `file-template-db2luw.txt` file can be placed in the `$PARAM/file` directory. The use of an alternative file is signaled during the execution of `file.sh` by the message:

Listing 8-3 Execution Log with Alternative Template File

```
#####
Control of templates

    OK: Use Templates list file from current project:

        File name is /home2/wkb9/param/file/file-templates-db2luw.txt
#####
```

file-move-assignment-db2luw.pgm

This file is placed during the installation of the Rehosting Workbench, it controls the transfer of components generated in the different installation directories. This file indicates the location of each component to copy during the installation phase of `file.sh`, when launched using `file.sh -i`.

The file is located in:

```
$REFINEDIR/convert-data/default/file/file-move-assignment-db2luw.pgm
```

This file can be modified following the instructions found at the beginning of the file:

Listing 8-4 file_move_assignment.txt Modification Instructions

```
[...]
```

```
*@ (c) Metaware:file-move-assignment.pgm. $Revision: 1.2 $
```

```

*release_format=2.4
*
* format is:
*   <typ>:<source_directory>:<file_name>:<target_directory>
*
* typ:
*   O: optional copy: if the <file_name> is missing, it is ignored
*   M: Mandatory copy: abort if <file_name> is missing.
*   E: Execution: execute the mandatory script <file_name>.
*       Parameters for script to be executed are:
*       basedir:          directory of REFINEDIR/convert-data
*       targetoutputdir:  value of "-i <targetdir>"
*       schema:           schema name
*       target_dir:       value written as 4th parameter in this file.
*   d: use this tag to display the word which follows
*
* source_directory:
*   T: generated components written in <targetdir>/Templates/<schema>
*   O: components written in <targetdir>/outputs/<schema>
*   S: SQL requests (DDL) generated into <targetdir>/SQL/<schema> directory
*   F: fixed components present in REFINEDIR
*   s: used with -s arguments: indicates the target directory for DML
utilities
*       (in REFINEDIR/modified-components/) which manipulate all schemas.
*
* file_name: (except for typ:d)
*   name of the file in <source_directory>
*

```

```

* target_directory: (except for typ:d, given at 4th argument for typ:E)
*   name of the target directory
*   If the 1st character is "/", component is copied using static directory
*   and not in <td> directory
*   If the 1st character is "!", target directory contains both directory
and
*   target file name.
*
[...]
```

Note: This file contains both File-to-File and File-to-DB2/LUW migration parameters.

Datamap File

This is a configuration file used by the Rehosting Workbench file converter to add or modify information on the physical files of a system.

See [File Converter: Introduction – Datamap File](#).

Mapper File

This is a configuration file used by the Rehosting Workbench File-to-Db2/luw (udb) Converter to associate each file to migrate

See [File Converter: Introduction – Mapper File](#).

Note: In the mapper file, the `converted` clause has to be used for RDBMS Table target.

Table 8-3 Mapper File Specific Parameters to be Used with File-to-Db2/luw (udb) Converter

<code>file <physical filename></code>	ZOS physical file name, Name used in the Datamap file.
<code>converted</code>	Indicates file is to be converted to Db2/luw (udb) table (<code>converted</code> clause can be combined with <code>transferred</code> clause)
<code>transferred</code>	Indicates that the file is to be loaded and reloaded (can be combined with <code>converted</code> clause).

Table 8-3 Mapper File Specific Parameters to be Used with File-to-Db2/luw (udb) Converter

<code>include "<path/COPY name>"</code>	Access path and name of the descriptive copy of the file to migrate.
<code>map record <record name> defined in <"path/COPY name"></code>	<ul style="list-style-type: none">• record name: corresponds to the level 01 field name of the copy description.• path/COPY name: corresponds to the access path and name of the descriptive copy of the file to migrate.
<code>source record <record names> defined in <"path/COPY name"></code>	<ul style="list-style-type: none">• record name: corresponds to the level 01 field name of the copy description of the file to migrate.• path/COPY name: corresponds to the access path and name of the descriptive copy of the file to migrate.
<code>Logical name <logical file name></code>	The Logical file name is chosen by the user, maximum eight characters. This name is used for naming the objects (COBOL, JCL) created by the different tools in the Rehosting Workbench.
<code>Converter name <program name></code>	Same name and use as logical file name.
<code>table name</code>	Db2/luw (udb) table name.
<code>attributes <attribute clause></code>	<p>It is an optional clause. Two attributes can be used:</p> <ul style="list-style-type: none">• LOGICAL_MODULE_IN_ADDITION• LOGICAL_MODULE_ONLY <p>Their action is described in the next table.</p>

Table 8-4 Mapper File Attributes

attributes <attribute clause>	Role
Attribute clause absent	In this case the following access functions are generated: <ul style="list-style-type: none">• RM_<logical file name>,• UL_<logical file name>,• DL_<logical file name> and the Korn shell utilities. See Access Functions and Utility Programs .
LOGICAL_MODULE_IN_ADDITION	In this case the following access functions are generated: <ul style="list-style-type: none">• ASG_<logical file name>• RM_<logical file name>,• UL_<logical file name>,• DL_<logical file name> and the Korn shell utilities. See Access Functions and Utility Programs .
LOGICAL_MODULE_ONLY	In this case only the ASG_<logical file name> access function is generated.

Listing 8-5 Mapper File Example

```
ufas mapper STFILEUDB
file PJ01AAA.SS.VSAM.CUSTOMER converted transferred
    table name CUSTOMER
    include "COPY/ODCSF0B.cpy"
    map record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"
    source record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"
    logical name ODCSF0B
    converter name ODCSF0B
    attributes LOGICAL_MODULE_IN_ADDITION
```

In this example the mapper file is named `STFILEUDB`. The file processes only one file named `PJ01AAA.SS.VSAM.CUSTOMER` that is migrated to an Db2/luw (udb) table using the `convert` option. The `ODCSF0B.cpy` copy file used to describe the file is one of the source copy files.

Mapping Strategy Clauses

```
[ field <field_name>

    [ use detail table ]

    [ use opaque field <field name> ]

    [ table name <target table name> ]

    [ mapped type <target data type> ]

    [ discard field <field name> ]

    [ discard subfields <field name> ]

    [ discrimination rule ] ]
```

Mapping Strategy Clause Syntax and Parameters

For `OCCURS` and `REDEFINES` clauses, using discrimination rules, three reengineering possibilities are proposed:

- Creation of sub-tables (use detail table)
 - Redefinitions: each description is associated with a sub-table (one sub-table for each description).
 - Occurs: one sub-table is created containing a technical column that references the original table to which the data corresponds.
- Creation of an opaque field (use opaque field).
 - Redefinitions: all the descriptions are stored in an opaque field type `CHAR` or `VARCHAR`.
 - Occurs: all the occurrences are stored in an opaque field type `CHAR` or `VARCHAR`.
- Extended description (default)
 - Redefinitions: all the fields described in the copy file are created as columns in the Db2/luw (udb) table.
 - Occurs: each occurrence of a field in a redefined area is created as a column in the Db2/luw (udb) table, one column for each occurrence in the `OCCURS` clause

Table 8-5 Mapping Strategies

Strategy	Description
table name < table name >	Name of sub-table in case of mapping 'use detail table'.
mapped type <target data type>	Enables the modification of the column type chosen by default. Two possibilities are proposed: CHAR or VARCHAR.
discard field	Enables the deletion of a non-useful redefined field.
discard subfields	When a field has several levels of description, this option allows to keep only the higher level.

Mapping Strategy Examples

Discard Subfield Example

```
...
05 NIV1.
    10 NIV2A PIC 99.
    10 NIV2B PIC 999.
...
```

When discarding subfields at the level NIV1, the Rehosting Workbench File-to-Db2/luw (udb) Converter only processes the field `NIV1 PIC 9(5)`. When not discarding subfields, the `NIV1` field is ignored and the two fields `NIV2A` and `NIV2B` are processed.

Redefines With Default Option Example

This redefines example is without any specific options:

Listing 8-6 Descriptive Copy of the File: PJ01AAA.SS.VSAM.CUSTOMER

```
01 VS-ODCSF0-RECORD.
    05 VS-CUSTIDENT          PIC 9(006).
    05 VS-CUSTLNAME          PIC X(030).
    05 VS-CUSTFNAME          PIC X(020).
```

```

05 VS-CUSTADDRS          PIC X(030).
05 VS-CUSTCITY           PIC X(020).
05 VS-CUSTSTATE          PIC X(002).
05 VS-CUSTBDATE          PIC 9(008).
05 VS-CUSTBDATE-G        REDEFINES VS-CUSTBDATE.
    10 VS-CUSTBDATE-CC PIC 9(002).
    10 VS-CUSTBDATE-YY PIC 9(002).
    10 VS-CUSTBDATE-MM PIC 9(002).
    10 VS-CUSTBDATE-DD PIC 9(002).
05 VS-CUSTEMAIL          PIC X(040).
05 VS-CUSTPHONE          PIC 9(010).
05 VS-FILLER             PIC X(100).

```

The mapper file implemented is:

Listing 8-7 Mapper File for the File: PJ01AAA.SS.VSAM.CUSTOMER

```

ufas mapper STFILEUDB
file PJ01AAA.SS.VSAM.CUSTOMER converted transferred
    table name CUSTOMER
    include "COPY/ODCSF0B.cpy"
    map record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"
    source record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"
    logical name ODCSF0B
    converter name ODCSF0B
    attributes LOGICAL_MODULE_IN_ADDITION
    field VS-CUSTBDATE

```

```
rule if VS-CUSTSTATE = "02" then VS-CUSTBDATE
else VS-CUSTBDATE-G
```

The table is generated as follows (all the unitary fields of the REDEFINES are handled).

Listing 8-8 Table Generation for the File: PJ01AAA.SS.VSAM.CUSTOMER

```
DROP TABLE CUSTOMER ;

COMMIT ;

CREATE TABLE CUSTOMER (

    VS_CUSTIDENT          NUMERIC (6) NOT NULL,

    VS_CUSTLNAME          VARCHAR (30),

    VS_CUSTFNAME          CHAR    (20),

    VS_CUSTADDRS          VARCHAR (30),

    VS_CUSTCITY           CHAR    (20),

    VS_CUSTSTATE          CHAR    (2),

    VS_CUSTBDATE          NUMERIC (8),

    VS_CUSTBDATE_CC       NUMERIC (2),

    VS_CUSTBDATE_YY       NUMERIC (2),

    VS_CUSTBDATE_MM       NUMERIC (2),

    VS_CUSTBDATE_DD       NUMERIC (2),

    VS_CUSTEMAIL          VARCHAR (40),

    VS_CUSTPHONE          NUMERIC (10),

    VS_FILLER             VARCHAR (100),

    CONSTRAINT PKCUSTOMER PRIMARY KEY (

        VS_CUSTIDENT)) ;

COMMIT ;
```

REDEFINES With OPAQUE FIELD Option Example

Listing 8-9 Descriptive Copy of the File: PJ01AAA.SS.VSAM.CUSTOMER

```
01 VS-ODCSF0-RECORD.
    05 VS-CUSTIDENT          PIC 9(006).
    05 VS-CUSTLNAME          PIC X(030).
    05 VS-CUSTFNAME          PIC X(020).
    05 VS-CUSTADDRS          PIC X(030).
    05 VS-CUSTCITY           PIC X(020).
    05 VS-CUSTSTATE          PIC X(002).
    05 VS-CUSTBDATE          PIC 9(008).
    05 VS-CUSTBDATE-G        REDEFINES VS-CUSTBDATE.
        10 VS-CUSTBDATE-CC PIC 9(002).
        10 VS-CUSTBDATE-YY PIC 9(002).
        10 VS-CUSTBDATE-MM PIC 9(002).
        10 VS-CUSTBDATE-DD PIC 9(002).
    05 VS-CUSTEMAIL          PIC X(040).
    05 VS-CUSTPHONE          PIC 9(010).
    05 VS-FILLER             PIC X(100).
```

The mapper file implemented is:

Listing 8-10 Mapper File for the File: PJ01AAA.SS.VSAM.CUSTOMER

```
ufas mapper STFILEUDB
file PJ01AAA.SS.VSAM.CUSTOMER converted transferred
table name CUSTOMER
```

```

include "COPY/ODCSF0B.cpy"

map record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"
source record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"

logical name ODCSF0B

converter name ODCSF0B

attributes LOGICAL_MODULE_IN_ADDITION

field VS-CUSTBDATE

use opaque field

rule if VS-CUSTSTATE = "02" then VS-CUSTBDATE
else VS-CUSTBDATE-G

```

The table is generated as follows (only the VS_CUSTBDATE field is kept).

Listing 8-11 Table Generation for the File: PJ01AAA.SS.VSAM.CUSTOMER

```

DROP TABLE CUSTOMER ;

COMMIT ;

CREATE TABLE CUSTOMER (
    VS_CUSTIDENT          NUMERIC (6) NOT NULL,
    VS_CUSTLNAME          VARCHAR (30),
    VS_CUSTFNAME          CHAR    (20),
    VS_CUSTADDRS          VARCHAR (30),
    VS_CUSTCITY            CHAR    (20),
    VS_CUSTSTATE           CHAR    (2),
    VS_CUSTBDATE           CHAR    (8)
                                FOR BIT DATA,
    VS_CUSTEMAIL           VARCHAR (40),

```

```

VS_CUSTPHONE          NUMERIC (10),
VS_FILLER              VARCHAR (100),
CONSTRAINT PKCUSTOMER PRIMARY KEY (
    VS_CUSTIDENT)) ;
COMMIT ;

```

REDEFINES With DETAIL TABLE Option Example

Listing 8-12 Descriptive Copy of the File: PJ01AAA.SS.VSAM.CUSTOMER

```

01 VS-ODCSF0-RECORD.
    05 VS-CUSTIDENT          PIC 9(006).
    05 VS-CUSTLNAME          PIC X(030).
    05 VS-CUSTFNAME          PIC X(020).
    05 VS-CUSTADDRS          PIC X(030).
    05 VS-CUSTCITY           PIC X(020).
    05 VS-CUSTSTATE          PIC X(002).
    05 VS-CUSTBDATE          PIC 9(008).
    05 VS-CUSTBDATE-G        REDEFINES VS-CUSTBDATE.
        10 VS-CUSTBDATE-CC PIC 9(002).
        10 VS-CUSTBDATE-YY PIC 9(002).
        10 VS-CUSTBDATE-MM PIC 9(002).
        10 VS-CUSTBDATE-DD PIC 9(002).
    05 VS-CUSTEMAIL          PIC X(040).
    05 VS-CUSTPHONE          PIC 9(010).
    05 VS-FILLER             PIC X(100).

```

The mapper file implemented is:

Listing 8-13 Mapper File for the File: PJ01AAA.SS.VSAM.CUSTOMER

```
ufas mapper STFILEUDB
file PJ01AAA.SS.VSAM.CUSTOMER converted transferred
    table name CUSTOMER
    include "COPY/ODCSF0B.cpy"
    map record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"
    source record VS-ODCSF0-RECORD defined in "COPY/ODCSF0B.cpy"
    logical name ODCSF0B
    converter name ODCSF0B
    attributes LOGICAL_MODULE_IN_ADDITION
    field VS-CUSTBDATE
    use detail table
    rule if VS-CUSTSTATE = "02" then VS-CUSTBDATE
    else VS-CUSTBDATE-G
```

The tables are generated as follows (a parent table is generated using the fields not part of the REDEFINES, and two child tables are generated, one for each REDEFINES description).

Listing 8-14 Table Generation for the File: PJ01AAA.SS.VSAM.CUSTOMER

```
DROP TABLE CUSTOMER ;
COMMIT ;
CREATE TABLE CUSTOMER (
    VS_CUSTIDENT            NUMERIC (6) NOT NULL,
    VS_CUSTLNAME            VARCHAR (30),
```

```

VS_CUSTFNAME          CHAR      (20),
VS_CUSTADDRS          VARCHAR (30),
VS_CUSTCITY           CHAR      (20),
VS_CUSTSTATE          CHAR      (2),
VS_CUSTEMAIL          VARCHAR (40),
VS_CUSTPHONE          NUMERIC (10),
VS_FILLER              VARCHAR (100),
CONSTRAINT PKCUSTOMER PRIMARY KEY (
    VS_CUSTIDENT)) ;
COMMIT ;
DROP TABLE VS_CUSTBDATE ;
COMMIT ;
CREATE TABLE VS_CUSTBDATE (
    VS_CUSTBDATE_CUSTIDENT    NUMERIC (6) NOT NULL,
    VS_CUSTBDATE              NUMERIC (8),
    CONSTRAINT PKVS_CUSTBDATE PRIMARY KEY (
        VS_CUSTBDATE_CUSTIDENT)) ;
COMMIT ;
ALTER TABLE VS_CUSTBDATE
    FOREIGN KEY FKVS_CUSTBDATECUSTOMER (
        VS_CUSTBDATE_CUSTIDENT)
    REFERENCES CUSTOMER
        ON DELETE CASCADE;
COMMIT ;
DROP TABLE VS_CUSTBDATE_G ;
COMMIT ;
CREATE TABLE VS_CUSTBDATE_G (

```

```

VS_CUSTBDATE_G_CUSTIDENT    NUMERIC (6) NOT NULL,
VS_CUSTBDATE_CC              NUMERIC (2),
VS_CUSTBDATE_YY              NUMERIC (2),
VS_CUSTBDATE_MM              NUMERIC (2),
VS_CUSTBDATE_DD              NUMERIC (2),
CONSTRAINT PKVS_CUSTBDATE_G PRIMARY KEY (
    VS_CUSTBDATE_G_CUSTIDENT)) ;

COMMIT ;

ALTER TABLE VS_CUSTBDATE_G
    FOREIGN KEY FKVS_CUSTBDATE_GCUSTOMER (
        VS_CUSTBDATE_G_CUSTIDENT)
    REFERENCES CUSTOMER
        ON DELETE CASCADE;

COMMIT ;

```

Discrimination Rules

A discrimination rule must be set on the redefined field; it describes the code to determine which description of the REDEFINES to use and when.

```

[field <field_name>]
    [...]
rule if <condition> then Field_Name_x
[elseif <condition> then field_Name_y]
[else Field_Name_z]

```

Discrimination Rules Syntax and Parameters

Table 8-6 Discrimination Rules

Syntax	Description
Field_Name_{X,Y,Z}	This is the field that will be used when the associated condition is validated; this field is one of the redefined fields.
Condition	<p>Is a conditional expression composed with field name, operators and COBOL constants.</p> <ul style="list-style-type: none">• Logical operators are: not, and, or• Comparison operators are: = <> < >• Specific operators are: is numeric, is all SPACE• Following COBOL constants may be used: spaces, zeros, high-value, low-value <p>Note: These conditions can be parenthesized.</p>

Discrimination Rules Examples

In the following example the fields `DPODP-DMDCHQ`, `DPONO-PRDTIV`, `DP5CP-VALZONNUM` are redefined.

Listing 8-15 Discrimination Rule COBOL Description

```
01 ZART1.

05 DPODP PIC X(20).

05 DPODP-RDCRPY PIC 9.

05 DPODP-DMDCHQ PIC X(6).

05 DPODP-REMCHQ REDEFINES DPODP-DMDCHQ.

10 DPODP-REMCHQ1 PIC 999.

10 DPODP-REMCHQ2 PIC 999.

05 DPODP-VIREXT REDEFINES DPODP-DMDCHQ.

10 DPODP-VIREXT1 PIC S9(11) COMP-3.

05 DPONO-NPDT PIC X(5).
```

```

05 DPONO-PRDTIV PIC 9(8)V99.
05 DPONO-PRDPS REDEFINES DPONO-PRDTIV PIC X(10).
05 DP5CP-VALZONNUM PIC 9(6).
05 DP5CP-VALZON REDEFINES DP5CP-VALZONNUM PIC X(6).

```

The following discrimination rules are applied:

Listing 8-16 Discrimination Rules

```

      field DPODP-DMDCHQ
rule if      DPODP-RDCRPHY = 1 then DPODP-DMDCHQ
      elseif DPODP-RDCRPHY = 2 then DPODP-REMCHQ
      elseif DPODP-RDCRPHY = 3 then DPODP-VIREXT
      else DPODP-DMDCHQ,
field DPONO-PRDTIV
rule  if      DPONO-NPDT (1:2) = "01"  then DPONO-PRDTIV
      elseif DPONO-NPDT (1:2) = "02"  then DPONO-PRDPS,
field DP5CP-VALZONNUM
rule if      DPODP-RDCRPHY is numeric then DP5CP-VALZONNUM
      else DP5CP-VALZON

```

The first rule is to test the value of the numeric field `DPODP-RDCRPHY`.

The second rule tests the first two characters of an alphanumeric field `DPONO-NPDT`. Only the values 01 and 02 are allowed.

The third rule tests whether the field `DPODP-RDCRPHY` is numeric.

COBOL Description

Oracle Tuxedo Application Rehosting Workbench File-to-Db2/luw (udb) Converter needs a description associated with each table, so a first step consists in preparing a COBOL copy description.

Once the COBOL description files have been prepared, the copy files described in the `mapper-<configuration name>.re` file should be placed in the `$PARAM/file/recs-source` directory.

If you use a COBOL copy book from the source platform to describe a file (see [COBOL Description](#)), then it is the location of the copy book that is directly used.

Description of the Output Files

File Locations

Location of Temporary Files

The temporary objects generated by the Rehosting Workbench File-to-Db2/luw (udb) Converter are stored in:

```
$TMPPROJECT
$TMPPROJECT/Template/<configuration name>
$TMPPROJECT/outputs/<configuration name>
```

Note: The `$TMPPROJECT` variable is set to: `$HOME/tmp`

Location of Log Files

The execution log files are stored in:

- Log generated by the option `-g`:
`$TMPPROJECT/outputs mapper-log-<configuration name>`

Location of Generated Files

The unloading and loading components generated with the `-i $HOME/trf` option are placed in the following locations:

Table 8-7 Component Locations

Location	Contents
<code>\$HOME/trf/unload/file/<configuration name></code>	<p>The JCL used for each unloading table are generated for each <code><configuration name></code>.</p> <p>These JCL are named:</p> <p><code><file name>.jclunload</code></p>
<code>\$HOME/trf/reload/file/<configuration name></code>	<p>For a file to Db2/luw (udb) migration, the programs and KSH are named:</p> <p><code>RELTABLE-<target file name>.sqb</code></p> <p><code>loadtable-<target file name>.ksh</code></p>
<code>\$HOME/trf/SQL/file/<configuration name></code> (When migrating files to Db2/luw tables).	<p>Location by <code><configuration name></code> of the SQL scripts used to create the Db2/luw (udb) objects.</p>
<code>\$HOME/trf/config/tux</code>	<p>Location of configuration files used by Oracle Tuxedo Application Runtime CICS for files migrated to tables.</p>
<code>\$HOME/trf/DML</code>	<p>Liste of components is depending on the optional attributes clause initialized in the mapper file.</p> <p>See Mapping Strategy Clause Syntax and Parameters</p>

Note: `<target table name>` Is the file name on the target platform; this file name is furnished in the mapper file.

Generated Objects

The following sections describe the objects generated during the migration of z/OS files and the directories in which they are placed.

Unloading JCL

The JCL used to unload the files are generated using the `-g` option of the `file.sh` command. They are then (using the `-i` option) installed in:

`$HOME/trf/unload/file/<configuration name>`

Each JCL contains two steps and unloads one file using the z/OS IDCAMS REPRO utility. The JCL return code is equal to 0 or 4 for a normal termination.

Step 1 DEL IDCAMS DELETE files (deletion of log, data files)

Step 2 UNLOAD IDCAMS REPRO of the indicated file

The JCLs are named: <file name>.jclunload

Note: The .jclunload extension should be deleted for execution under z/OS.

The generated JCL may need adapting to specific site constraints including:

- JOB cards: <cardjob_parameter_id>,
- access paths to input and output files: <data>.

Listing 8-17 Unload JCL Example

```
//<crdjob> <cardjob_parameter_1>,'FIL QSAM',
//          <cardjob_parameter_2>
//          <cardjob_parameter_3>
//          <cardjob_parameter_4>
// *@ (C) Metaware:jcl-unload-MVS-REPRO.pgm. $Revision: 1.6 $
// *****
// * UNLOAD THE FILE:
// *      <datain>.QSAM.CUSTOMER
// * INTO <data>.AV.QSAM
// *      LENGTH=266
// *****
// *-----*
// * DELETE DATA AND LOG FILES
// *-----*
```

```

//DEL          EXEC PGM=IDCAMS

//SYSPRINT DD SYSOUT=*

//SYSOUT       DD SYSOUT=*

//SYSIN        DD *

    DELETE <data>.AV.QSAM.LOG

    DELETE <data>.AV.QSAM

    SET MAXCC=0

// *-----*

// * LAUNCH REPRO UTILITY

// *-----*

//COPYFILE EXEC PGM=IDCAMS

//SYSPRINT DD SPACE=(CYL,(150,150),RLSE),

//          DISP=(NEW,CATLG),

//          UNIT=SYSDA,

//          DSN=<data>.AV.QSAM.LOG

//SYSOUT     DD SYSOUT=*

//INDD       DD DISP=SHR,

              DSN=METAW00.QSAM.CUSTOMER

//OUTD       DD SPACE=(CYL,(150,150),RLSE),

//          DISP=(NEW,CATLG),

//          UNIT=SYSDA,

//          DCB=(LRECL=266,RECFM=FB),

//          DSN=<data>.AV.QSAM

//SYSIN      DD *

    REPRO INFILE(INDD) OUTFILE(OUTD)

/*

```

COBOL Transcoding Programs

Migration of z/OS Files to Db2/luw (udb) Tables

The COBOL transcoding programs are generated using the `-g` option of the `file.sh` command. They are then (using the `-i` option) installed in:

```
$HOME/trf/reload/file/<configuration name>/src
```

The programs are named: `RELTABLE-<logical file name>.sqb`

The programs should be compiled using the Microfocus COBOL compilation options and db2 Precompiler options documented in [Compiler Options](#).

The compilation of these programs requires the presence of a `CONVERTMW.cpy` copy file adapted to the project documented in [Codeset Conversion](#).

These files read a file on input and directly load an Db2/luw (udb) table using the SQL INSERT verb.

Listing 8-18 FILE CONTROL Section - for Transcoding Programs

```
SELECT MW-ENTREE

        ASSIGN TO "ENTREE"

        ORGANIZATION IS RECORD SEQUENTIAL

        ACCESS IS SEQUENTIAL

        FILE STATUS IS IO-STATUS.
```

For Db2/luw (udb) table with technical column, a SEQUENCE object is created:

```
CREATE SEQUENCE <table_name>_<type>_SEQ START WITH <num_rows>
```

A commit is made every 1000 records:

```
IF MW-NB-INSERT >= 1000

CALL "do_commit"
```

Note: The `do_commit` module is part of Oracle Tuxedo Application Runtime Batch.

A record count is written to the output file and is displayed at the end of processing via:

```

DISPLAY "RELOADING TERMINATED OK".

DISPLAY "Nb rows reloaded: " D-NB-RECS.

DISPLAY " ".

DISPLAY "NUMERIC MOVED WHEN USING CHAR FORMAT : "

DISPLAY "  NUMERIC-BCD : " MW-COUNT-NUMERIC-BCD-USE-X.

DISPLAY "  NUMERIC-DISP: " MW-COUNT-NUMERIC-DISP-USE-X.

```

The last two lines displayed signal the movement of data into fields where the COBOL description does not match the content of the input file (packed numeric fields containing non-numeric data and numeric DISPLAY fields containing non-numeric data). When such cases are encountered, each error is displayed.

Note: When migrating to a target platform using Intel hardware the message: “PROCESSOR UNIT IS INTEL” is output at the beginning of transcoding.

Reloading Korn Shell Scripts

The Reloading Korn shell scripts are generated using the `-g` option of the `file.sh` command. They are then (using the `-i` option) installed in:

```
$HOME/trf/reload/file/<configuration name>
```

Reloading Korn Shell Scripts for Migrating z/OS Files to Db2/luw (udb) Tables

The scripts are named: `loadtable-<logical file name>.ksh`

They contain a DDL creation phase, a transcoding (or loading) phase and a check phase. The different phases may be launched separately.

The execution of the scripts produces an execution log in `$MT_LOG/<logical file name>.log`

The following variables are set at the beginning of each script:

Listing 8-19 Reloading Table Script Variables

```

f="@ (c) Metaware:load-tables-ksh-db2luw.pgm. $Revision: 1.2.2.1 $null"

echo "reloading ODCSF0B into DB2 UNIX"

export DD_ENTREE=${DD_ENTREE:-${DATA_SOURCE}/ODCSF0B}

logfile=$MT_LOG/ODCSF0B.log

reportfile=${MT_LOG}/ODCSF0B.rpt

```

```
ddlfile=${DDL}/STFILEORA/ODCSF0B.sql
```

```
[...]
```

To change the file names, set the DD_ENTREE and DD_SORTIE variables before calling the script.

Various messages may be generated during the three execution phases of the scripts; explanations of these messages are listed in [Oracle Tuxedo Application Rehosting Workbench Messages](#).

On normal end, a return code of 0 is returned.

Creating Oracle DDL Phase

Db2/luw (udb) objects are created under db2 command using:

```
${DDL}/STFILEUDB/ODCSF0B.sql
```

```
db2 -ec -tvf ${ddlfile} >>$logfile 2>&1
```

On normal termination the following message is displayed:

```
echo "Table(s) created"
```

Note: Log file can contain errors on DROP statements: these errors are ignored by the script.

Transcoding and Loading Phases

These steps launch the execution of the COBOL transcoding program associated with the file processed:

```
runb RELTABLE-ODCSF0B >> $logfile 2>&1
```

On normal termination the following message is displayed:

```
echo "File ${DD_ENTREE} successfully transcoded and reloaded into  
DB2/LUW"
```

Check Phase

This step verifies after the reloading that the reloaded Db2/luw (udb) table contains the same number of records as the records transferred from ZOS on target platform. If the number of records is different, an error message is produced: . If the number of records is equal, this message is produced:

```
"Number of rows written in output file is equal to number calculated using  
the log file: OK"
```

Target DDL

The Db2/luw (udb) DDL is generated using the -g option of the file.sh command. They are then (using the -i option) installed in:

\$HOME/trf/SQL/file/<schema name>

They are named: <target file name>.ddl.

The format used is:

```
DROP TABLE <target_table_name>;

COMMIT;

CREATE TABLE <target_table_name> (
    <target_column_name> <column_data_type> <attribute>[, ...]
    CONSTRAINT <constraint_name> PRIMARY KEY (<target_column_name>)
    CONSTRAINT <fk_constraint_name> FOREIGN KEY (<target_column_name>)]
);
```

Where

<target_table_name>	Db2/luw (udb) table name.
<target_column_name>	Db2/luw (udb) column name.
<column_data_type>	Db2/luw (udb) data type (CHAR, VARCHAR or NUMERIC).
<attribute>	NOT NULL when primary key.
<constraint_name>	Constraint name of primary key (PK_<UDB table name>)
<fk_constraint_name>	Constraint name of foreign key (FK_<UDB table name>_<parent_table_name>)

Listing 8-20 DDL Generation sql Example

```
DROP TABLE CUSTOMER ;

COMMIT ;

CREATE TABLE CUSTOMER (
```

```

VS_CUSTIDENT          NUMERIC (6) NOT NULL,
VS_CUSTLNAME          VARCHAR (30),
VS_CUSTFNAME          CHAR      (20),
VS_CUSTADDRS          VARCHAR (30),
VS_CUSTCITY           CHAR      (20),
VS_CUSTSTATE          CHAR      (2),
VS_CUSTBDATE          NUMERIC (8),
VS_CUSTEMAIL          VARCHAR (40),
VS_CUSTPHONE          NUMERIC (10),
VS_FILLER             VARCHAR (100),
CONSTRAINT PKCUSTOMER PRIMARY KEY (
    VS_CUSTIDENT)) ;
COMMIT ;

```

Access Functions and Utility Programs

Access Functions

These access functions are generated using the `-g` option of `file.sh` and installed in `$HOME/trf/DML` using the `-i` and `-s` options..

Table 8-8 Access Functions

Access function	Role
RM_<logical file name>.sqb	Relational access module to Db2/luw (udb) table that replaces the specified logical file name.
DL_<logical file name>.cbl	Download module of the specified logical file (function used by Oracle Tuxedo Application Runtime Batch).
UL_<logical file name>.cbl	Upload module of the specified logical file (function used by Oracle Tuxedo Application Runtime Batch).

Table 8-8 Access Functions

Access function	Role
<code>ASG_<logical file name>.cbl</code>	Optional module generated when there are multiple assigns. See Mapper File Attributes
<code>getfileinfo.cbl</code>	This program checks if the <logical file name>.rdb associated with the assign-name given as an input argument exists. This function is called by <code>ASG_<logical file name>.cbl</code> .
<code>init_all_files.cbl</code>	Initialize all variables used by relational module and logical module (function used by Oracle Tuxedo Application Runtime Batch).
<code>init_all_files_<configuration name>.cbl</code>	Initialize all variables used by relational module and <code>ASG_<logical file name></code> module for the configuration name listed; (function used by Oracle Tuxedo Application Runtime Batch).
<code>dml_locking.cbl</code>	This program manages locking for all configuration files (function used by Oracle Tuxedo Application Runtime Batch).
<code>close_all_files_<configuration name>.cbl</code>	This program closes all cursors opened in tables for the configuration listed and closes all files opened with logical accessor (function used by Oracle Tuxedo Application Runtime Batch).
<code>close_all_files.cbl</code>	This program closes all cursors opened in tables (function used by Oracle Tuxedo Application Runtime Batch).

Access Function Call Arguments

The `RM_<logical file name>.sqb` and `ASG_<logical file name>.cbl` access functions use the following variables

Table 8-9 Access Call Implemented Variables

Variable	Description/origin
Function code	Indicates the type of operation to execute, for example OPEN, WRITE, etc. The code is passed using the FILE-CODE-F variable of the MWFITECH copy file.
File open mode	A file can be opened in different modes: INPUT, OUTPUT, I O, EXTEND. The mode is passed using the FILE-OPEN-MODE variable of the MWFITECH copy file.
IO-STATUS	The IO-STATUS variable is linked to each file providing the execution status of the last relational module operation.
Record to transmit or receive	The record to transmit has an access function for write operations or access by key; the record to receive has a read access function. These are described in the LINKAGE SECTION.
Name of secondary key to use	For indexed files with secondary keys, and only for files with this organization, an extra variable is required to identify the secondary key to use for a START operation. The name of the secondary key is passed using the FILE-ALT-KEY-NAME variable of the MWFITECH copy file. For files without secondary keys, this argument is unnecessary.
Relative Key	For a relative file, the value of the relative key is passed to or from the access module using the FILE-REL-KEY variable of the MWFITECH copy file.

Listing 8-21 LINKAGE SECTION Structure

LINKAGE SECTION.

```

01  IO-STATUS  PIC XX.

COPY MWFITECH.

*  *COBOL Record Description

01  VS-ODCSF0-RECORD.

06  X-VS-CUSTIDENT.
```

```

        07 VS-CUSTIDENT          PIC 9(006).
06 VS-CUSTLNAME                  PIC X(030).
06 VS-CUSTFNAME                  PIC X(020).
06 VS-CUSTADDRS                  PIC X(030).
06 VS-CUSTCITY                   PIC X(020).
06 VS-CUSTSTATE                  PIC X(002).
06 X-VS-CUSTBDATE.

        07 VS-CUSTBDATE          PIC 9(008).
06 VS-CUSTBDATE-G                REDEFINES VS-CUSTBDATE.
11 X-VS-CUSTBDATE-CC.
        12 VS-CUSTBDATE-CC      PIC 9(002).
11 X-VS-CUSTBDATE-YY.
        12 VS-CUSTBDATE-YY      PIC 9(002).
11 X-VS-CUSTBDATE-MM.
        12 VS-CUSTBDATE-MM      PIC 9(002).
11 X-VS-CUSTBDATE-DD.
        12 VS-CUSTBDATE-DD      PIC 9(002).
06 VS-CUSTEMAIL                  PIC X(040).
06 X-VS-CUSTPHONE.

        07 VS-CUSTPHONE          PIC 9(010).
06 VS-FILLER                     PIC X(100).

PROCEDURE DIVISION USING IO-STATUS

                                MW-FILE-TECH

                                VS-ODCSF0-RECORD.

```

Call Arguments Used

OPEN

For all OPEN operations, the `FILE-CODE-F` variable should contain the key-word OPEN.

The `FILE-OPEN-MODE` variable should contain the type of OPEN to perform as follows:.

Table 8-10 Call Argument File Open Modes

Source	Target
OPEN INPUT filename1	INPUT => FILE-OPEN-MODE
OPEN OUTPUT filename1	OUTPUT => FILE-OPEN-MODE
OPEN I-O filename1	I-O => FILE-OPEN-MODE
OPEN EXTEND filename1	EXTEND => FILE-OPEN-MODE

CLOSE

For CLOSE operations, the `FILE-CODE-F` variable should contain the key-word CLOSE.

CLOSE-LOCK

For CLOSE LOCK operations, the `FILE-CODE-F` variable should contain the key-word CLOSE-LOCK.

DELETE

Depending on the file access mode, the DELETE operation is either the current record or the one indicated by the file key.

The corresponding function code is indicated as follows:

Table 8-11 Call Argument Delete Modes

Access	Source	Target
Sequential	DELETE filename1	DELETE-CUR => FILE-CODE-F
Random or dynamic	DELETE filename1	DELETE-KEY => FILE-CODE-F

READ

The function code depends on the file access mode and the type of read required: sequential read, read primary key or read secondary key.

Table 8-12 Read Operation Values Depending on Arguments

Access	Source	Target
Sequential	READ filename1 [NEXT]	READ-NEXT => FILE-CODE-F
Random	READ filename1	READ-KEY => FILE-CODE-F
Dynamic	READ filename1 NEXT	READ-NEXT => FILE-CODE-F
	READ filename1	READ-KEY => FILE-CODE-F
	READ filename1 PREVIOUS	READ-PREV => FILE-CODE-F
If <i>DataName1</i> is a variable corresponding to the keyAltKey1	READ filename1 KEY DataName1	READ-ALT-KEY => FILE-CODE-F "AltKey1" => FILE-ALT-KEY-NAME
DataName1 represents the relative key	READ filename1 KEY DataName1	READ-REL-KEY => FILE-CODE-F "RelKeyVar" => FILE-REL-KEY

Note: If the INTO clause is found, a MOVE operation is added after the call in order to set the value of the indicated field.

REWRITE

The function code depends on the file access mode and the type of read required: sequential read, read primary key or read secondary key

Table 8-13 Rewrite Operation Values Depending on Arguments

Access	Source	Target
Sequential	REWRITE RecName1	REWRITE-CUR => FILE-CODE-F
Random or dynamic	REWRITE RecName1	REWRITE-KEY => FILE-CODE-F

Note: If the FROM clause is found, a MOVE operation is added before the call in order to set the value of the indicated field.

START

Whether the file is relative, indexed, with or without secondary key, the function code depends on the exact type of start.

Table 8-14 Rewrite Operation Values Depending on Arguments

When	Source	Target
	START file1	START-EQUAL => FILE-CODE-F
DataName1 represents the relative key or Primary Key of file1	START file1 KEY { EQUAL = EQUALS } DataName1	START-EQUAL => FILE-CODE-F
	START file1 KEY { EXCEEDS > GREATER } DataName1	START-SUP => FILE-CODE-F
	START file1 KEY { NOT LESS GREATER OR EQUAL NOT < >= } DataName1	START-SUPEQ => FILE-CODE-F
	START file1 KEY {< LESS } DataName1	START-INF => FILE-CODE-F
	START file1 KEY { NOT GREATER LESS OR EQUAL NOT > <= } DataName1	START-INFEQ => FILE-CODE-F

Table 8-14 Rewrite Operation Values Depending on Arguments

When	Source	Target
DataName1 is a variable corresponding to the AltKey1 key	START file1 KEY { EQUAL = EQUALS } DataName1	AltKey1 => FILE-ALT-KEY-NAME START-ALT-EQUAL => FILE-CODE-F
	START file1 KEY { EXCEEDS > GREATER } DataName1	AltKey1 => FILE-ALT-KEY-NAME START-ALT-SUP => FILE-CODE-F
	START file1 KEY { NOT LESS GREATER OR EQUAL NOT < >= } DataName1	AltKey1 => FILE-ALT-KEY-NAME START-ALT-SUPEQ => FILE-CODE-F
	START file1 KEY { < LESS } DataName1	AltKey1 => FILE-ALT-KEY-NAME START-ALT-INF => FILE-CODE-F
	START file1 KEY { NOT GREATER LESS OR EQUAL NOT > <= } DataName1	AltKey1 => FILE-ALT-KEY-NAME START-ALT-INFEQ => FILE-CODE-F

WRITE

The function code depends on the file access mode and the type of read required: sequential read, read primary key or read secondary key

Table 8-15 Write Operation Values Depending on Arguments

Access	Source	Target
Sequential	WRITE RecName1	WRITE-SEQ => FILE-CODE-F
Random or dynamic	WRITE RecName1	WRITE-KEY => FILE-CODE-F

Note: If the FROM clause is found, a MOVE operation is added before the call in order to set the value of the indicated field.

Copy Files to Be Implemented

The following copy files are used by certain access functions. They should be placed in the directory: `< installation platform>/fixed-copy/` during the installation of the Rehosting Workbench:

- `MW-PARAM-TRACE-VAR.cpy`
- `MW-PARAM-TRACE.cpy`
- `MW-PARAM-GETFILEINFO-VAR.cpy`
- `MW-PARAM-GETFILEINFO.cpy`
- `MW-PARAM-ERROR-VAR.cpy`
- `MW-PARAM-ERROR.cpy`
- `MW-PARAM-DML-LOCKING.cpy`
- `MWFITECH.cpy`
- `ERROR-SQLCODE-DB2LUW.cpy`

Korn Shell Utilities

These KSH scripts are generated using the `-g` option of `file.sh` and then installed in `$HOME/trf/SQL/file/<configuration name>` using the `-i` option. When necessary, they are used by Oracle Tuxedo Application Runtime Batch.

Table 8-16 Korn Shell Utilities

Korn shell script name	Role
<code>cleantable-<logical file name>.ksh</code>	Script file that CLEANs all tables associated with this file.
<code>createtable-<logical file name>.ksh</code>	Script file that CREATEs all table, constraint and indexes associated with this file.
<code>droptable-<logical file name>.ksh</code>	Script file that DROPs all tables associated with this file.

Table 8-16 Korn Shell Utilities

Korn shell script name	Role
<code>ifemptytable-<logical file name>.ksh</code>	Script file that checks if all tables are empty.
<code>ifexisttable-<logical file name>.ksh</code>	Script file that checks if all tables exist.

Oracle Tuxedo Application Runtime for CICS Configuration Files

The `desc.vsam` and `envfile_tux` files are generated in the `$HOME/trf/config/tux/` directory when VSAM files are migrated to Db2/luw (udb) tables. They are used by Oracle Tuxedo Application Runtime CICS.

COBOL and JCL Conversion Guide Files

These files are generated using the `-s` option of the `file.sh` command.

This file is used by the Rehosting Workbench [COBOL Converter](#) and JCL Converter to rename object names.

Table 8-17 Conversion file Names

<code>File-in-table-<configuration name></code>	Used by the Rehosting Workbench JCL converter
<code>File-in-table.txt</code>	Used by the Rehosting Workbench JCL converter
<code>Conv-ctrl.txt</code>	Used by the Rehosting Workbench COBOL converter
<code>Conv-ctrl-<configuration name></code>	Used by the Rehosting Workbench COBOL converter

.rdb Files

These files are created when VSAM files are converted to Db2/luw (udb) tables. They are used by Oracle Tuxedo Application Runtime Batch to bridge the technical differences between the z/OS file on the source platform and the corresponding Db2/luw (udb) table on the target platform.

The files are generated in: \$HOME/trf/data

They are named: <source platform physical file name>.rdb

The files contain two lines described in the next section.

Parameters and Syntax

```
${DATA}/<source platform physical file name> <max> <org> <form> UL_<logical  
file name> <asgn_in> DL_<logical file name> <asgn_out> RM_<logical file  
name> <target table name> ${DDL}/<configuration name>/cleantable-<target  
table name>.ksh ${DDL}/<configuration name>/droptable-<target table  
name>.ksh ${DDL}/<configuration name>/createtable-<target table name>.ksh  
${DDL}/<configuration name>/ifemptytable-<target table name>.ksh  
${DDL}/<configuration name>/ifexisttable-<target table name>.ksh
```

```
IDX_KEY <column name> <n m>
```

```
REL_KEY - <m>
```

Table 8-18 .rdb File Parameters

Parameter	Description
First Line:	
<source platform physical file name>	Physical file name
<max>	Maximum Record Size (in COBOL description).
<org>	File organization: <ul style="list-style-type: none">• IDX: indexed without alternate key• IDX_ALT: indexed with alternate key(s)• SEQ: sequential• REL: relative

Table 8-18 .rdb File Parameters

Parameter	Description
<form>	Record format: <ul style="list-style-type: none"> • FIX: fixed file • VAR:<min> variable file with minimal size. If <min> is missing, minimal size will be 1.
UL_<logical file name>	Uploading component name used by Oracle Tuxedo Application Runtime Batch.
<asgn_in>	Assign file name used by the uploading component.
DL_<logical file name>	Downloading component name used by Runtime.
<asgn_out>	Assign file name used by the downloading component.
RM_<logical file name>	Relational module name.
<target table name>	Name of the first table name (master table name or first table name for multi-record).
\${DDL}/<configuration name/cleantable-<target table name>.ksh	Name of the script file that CLEANs all tables associated with this file.
\${DDL}/<configuration name/droptable-<target table name>.ksh	Name of the script file that DROPs all tables associated with this file.
\${DDL}/<configuration name>/createtable-<target table name>.ksh	Name of the script file that CREATEs all tables associated with this file and their objects (constraints, indexes).
\${DDL}/<configuration name>/ifemptytable-<target table name>.ksh	Name of the script file that checks if all tables are empty.
\${DDL}/<configuration name>/ifexisttable-<target table name>.ksh	Name of the script file that checks if all tables exist.
Second Line for indexed file and indexed with alternate key file only:	
IDX_KEY	Constant.

Table 8-18 .rdb File Parameters

Parameter	Description
<VS-column name>	Indexed key name (group zone name or elementary field name as described in COBOL description).
<n m>	<ul style="list-style-type: none">n: offset of the indexed key (in COBOL description).m: length of the indexed key (in COBOL description).
Second Line for relative file:	
REL_KEY	Constant.
-	Constant.
<m>	<ul style="list-style-type: none">m: length of the relative key (in COBOL description).

Example of .rdb File

The following example is generated when migrating an indexed VSAM file to an Db2/luw (udb) table. On the source platform, the VSAM file is named: PJ01AAA.SS.VSAM.CUSTOMER

Listing 8-22 .rdb Indexed VSAM Example

```
${DATA}/PJ01AAA.SS.VSAM.CUSTOMER 266 IDX FIX UL_ODCSF0B ENTREE DL_ODCSF0B
SORTIE RM_ODCSF0B CUSTOMER ${DDL}/STFILEORA/cleantable-ODCSF0B.ksh
${DDL}/STFILEORA/droptable-ODCSF0B.ksh
${DDL}/STFILEORA/createtable-ODCSF0B.ksh
${DDL}/STFILEORA/ifemptytable-ODCSF0B.ksh
${DDL}/STFILEORA/ifexisttable-ODCSF0B.ksh

IDX_KEY VS-CUSTIDENT 1 6
```

Execution Reports

file.sh creates different execution reports depending on the options chosen. In the following examples the following command is used:

```
file.sh -gmi $HOME/trf STFILEUDB
```

Listing 8-23 Messages Produced when Using the Options -g with File.sh (step 1)

```
*-----
#####
Control of configuration STFILEUDB
#####
Control of templates
    OK: Use Default Templates list file
        File name is
/Qarefine/release/M3_L3_6/convert-data/default/file/file-templates-db2luw.
txt
#####
    Control of Mapper
#####
    COMPONENTS GENERATION

CMD : /Qarefine/release/M3_L3_6/scripts/launch file-converter -s
/home2/wkb4/param/system.desc -mf /home2/wkb4/tmp/mapper-STFILEUDB.re.tmp
-dmf /home2/wkb4/param/file/Datamap-STFILEUDB.re -td /home2/wkb4/tmp -tmps
/home2/wkb4/tmp/file-templates-STFILEUDB.tmp -target-sgbd udb9 -target-os
unix -varchar2 29 -print-ddl -print-dml -abort

MetaWorld starter

Loading lib: /Qarefine/release/M3_L3_6/Linux64/lib64/localext.so
(funcall LOAD-THE-SYS-AND-APPLY-DMAP-AND-MAPPER)

FILE-0092: *File-Converter*: We are in BATCH mode

FILE-0087: * Comand line arguments: begining of analyze

FILE-0088: * recognized argument -s value: /home2/wkb4/param/system.desc

FILE-0088: * recognized argument -mf value:
/home2/wkb4/tmp/mapper-STFILEUDB.re.tmp
```

```

FILE-0088: * recognized argument -dmf value:
/home2/wkb4/param/file/Datamap-STFILEUDB.re
FILE-0088: * recognized argument -td value: /home2/wkb4/tmp
FILE-0088: * recognized argument -tmps value:
/home2/wkb4/tmp/file-templates-STFILEUDB.tmp
FILE-0088: * recognized argument -target-sgbd value: udb9
FILE-0088: * recognized argument -target-os value: unix
FILE-0088: * recognized argument -varchar2 value: 29
FILE-0089: * recognized argument -print-ddl
FILE-0089: * recognized argument -print-dml
FILE-0089: * recognized argument -abort
FILE-0091: * End of Analyze
FILE-0094: * Parsing mapper file /home2/wkb4/tmp/mapper-STFILEUDB.re.tmp
...
FILE-0095: * Parsing data-map file
/home2/wkb4/param/file/Datamap-STFILEUDB.re ...
FILE-0096: * Parsing system description file /home2/wkb4/param/system.desc
...
Warning! OS clause is absent, assuming OS is IBM
Current OS is IBM-MF
Loading /home2/wkb4/source/symtab-STFILEUDB.pob at 14:49:41... done at
14:49:41
Build-Symtab-DL1 #1<a SYMTAB-DL1>
... Postanalyze-System-RPL...
sym=#2<a SYMTAB>
PostAnalyze-Common #2<a SYMTAB>
    0 classes
    0 classes
    0 classes

```

```

0 classes

1 classes

13 classes

Loading /home2/wkb4/source/BATCH/pob/RSSABB01.cbl.shrec...
Loading /home2/wkb4/source/COPY/pob/ODCSF0.cpy.cdm...
Loading /home2/wkb4/source/COPY/pob/ODCSF0B.cpy.cdm...
Loading /home2/wkb4/source/COPY/pob/ODCSFU.cpy.cdm...
FILE-0001: * Point 1 !!
FILE-0002: * Point 2 !!
FILE-0010: * Parsing file /home2/wkb4/source/COPY/ODCSF0.cpy ...
*Parsed 22 lines*
FILE-0010: * Parsing file /home2/wkb4/source/COPY/MW_SYSOUT.cpy ...
*Parsed 8 lines*
FILE-0010: * Parsing file /home2/wkb4/source/COPY/ODCSFU.cpy ...
*Parsed 24 lines*
FILE-0010: * Parsing file /home2/wkb4/source/COPY/ODCSF0B.cpy ...
*Parsed 22 lines*
FILE-0003: * Point 3 !!
FILE-0004: * Point 4 !!
FILE-0005: * Point 5 !!
FILE-0052: * loading pob file
/Qarefine/release/M3_L3_6/convert-data/templates/file/unloading/jcl-unload
-MVS-REPRO.pgm.pob
FILE-0085: * Expanding
/Qarefine/release/M3_L3_6/convert-data/templates/file/unloading/jcl-unload
-MVS-REPRO.pgm ...
FILE-0054: * Writing ODCSF0B.jclunload

[...]
```

```

FILE-0052: * loading pob file
/Qarefine/release/M3_L3_6/convert-data/templates/file/dml/generate-post-pr
ocess.pgm.pob

FILE-0085: * Expanding
/Qarefine/release/M3_L3_6/convert-data/templates/file/dml/generate-post-pr
ocess.pgm ...

FILE-0054: * Writing post-process-file.sh

FILE-0053: * Parsing template file
/Qarefine/release/M3_L3_6/convert-data/default/file/file-move-assignment-
db2luw.pgm

FILE-0085: * Expanding
/Qarefine/release/M3_L3_6/convert-data/default/file/file-move-assignment-
db2luw.pgm ...

FILE-0054: * Writing file-move-assignment.lst

Rest in peace, Refine...

*-----
Generated components are in /home2/wkb4/tmp/Template/STFILEUDB
(Optionaly in /home2/wkb4/tmp/SQL/STFILEUDB)
*-----

```

Listing 8-24 Messages Produced When Using the Options -m with File.sh (step 2)

```

#####
  FORMATTING COBOL LINES
#####
  CHANGE ATTRIBUTE TO KSH or SH scripts
*-----

```

Components are modified into /home2/wkb4/tmp directory

```
*-----  
#####  
  UDBINSTALL COMPONENTS INTO SPECIFIC DIRECTORY USING  
file-move-assignment.lst  
=====
```

==_PJ01AAA.SS.VSAM.CUSTOMER_==

Copied <Templates>:ODCSF0B.jclunload to
<td>/unload/file/STFILEUDB/ODCSF0B.jclunload

Copied <Templates>:loadtable-ODCSF0B.ksh to
<td>/reload/file/STFILEUDB/loadtable-ODCSF0B.ksh

Copied <Templates>:RETABLE-ODCSF0B.sqb to
<td>/reload/file/STFILEUDB/RETABLE-ODCSF0B.sqb

Copied <Templates>:ASG_ODCSF0B.cbl to <td>/DML/ASG_ODCSF0B.cbl

Copied <Templates>:RM_ODCSF0B.sqb to <td>/DML/RM_ODCSF0B.sqb

Copied <Templates>:DL_ODCSF0B.cbl to <td>/DML/DL_ODCSF0B.cbl

Copied <Templates>:UL_ODCSF0B.cbl to <td>/DML/UL_ODCSF0B.cbl

Copied <Templates>:PJ01AAA.SS.VSAM.CUSTOMER.rdb to
<td>/data/PJ01AAA.SS.VSAM.CUSTOMER.rdb

Copied <SQL>:ODCSF0B.sql to <td>/SQL/file/STFILEUDB/ODCSF0B.sql

Copied <Templates>:cleantable-ODCSF0B.ksh to
<td>/SQL/file/STFILEUDB/cleantable-ODCSF0B.ksh

Copied <Templates>:droptable-ODCSF0B.ksh to
<td>/SQL/file/STFILEUDB/droptable-ODCSF0B.ksh

Copied <Templates>:createtable-ODCSF0B.ksh to
<td>/SQL/file/STFILEUDB/createtable-ODCSF0B.ksh

Copied <Templates>:ifemptytable-ODCSF0B.ksh to
<td>/SQL/file/STFILEUDB/ifemptytable-ODCSF0B.ksh

Copied <Templates>:ifexisttable-ODCSF0B.ksh to
<td>/SQL/file/STFILEUDB/ifexisttable-ODCSF0B.ksh

```

=====

==_PJ01AAA.SS.QSAM.CUSTOMER.REPORT_==

  Copied    <Templates>:MW-SYSOUT.jclunload to
<td>/unload/file/STFILEUDB/MW-SYSOUT.jclunload

  Copied    <Templates>:loadfile-MW-SYSOUT.ksh to
<td>/reload/file/STFILEUDB/loadfile-MW-SYSOUT.ksh

  Copied    <Templates>:RELFILE-MW-SYSOUT.cbl to
<td>/reload/file/STFILEUDB/RELFILE-MW-SYSOUT.cbl

=====

==_PJ01AAA.SS.QSAM.CUSTOMER.UPDATE_==

  Copied    <Templates>:ODCSFU.jclunload to
<td>/unload/file/STFILEUDB/ODCSFU.jclunload

  Copied    <Templates>:loadfile-ODCSFU.ksh to
<td>/reload/file/STFILEUDB/loadfile-ODCSFU.ksh

  Copied    <Templates>:RELFILE-ODCSFU.cbl to
<td>/reload/file/STFILEUDB/RELFILE-ODCSFU.cbl

=====

==_PJ01AAA.SS.QSAM.CUSTOMER_==

  Copied    <Templates>:ODCSF0Q.jclunload to
<td>/unload/file/STFILEUDB/ODCSF0Q.jclunload

  Copied    <Templates>:loadfile-ODCSF0Q.ksh to
<td>/reload/file/STFILEUDB/loadfile-ODCSF0Q.ksh

  Copied    <Templates>:RELFILE-ODCSF0Q.cbl to
<td>/reload/file/STFILEUDB/RELFILE-ODCSF0Q.cbl

=====

  Copied    <Templates>:close_all_files_STFILEUDB.cbl to
<td>/DML/close_all_files_STFILEUDB.cbl

  Copied    <Templates>:init_all_files_STFILEUDB.cbl to
<td>/DML/init_all_files_STFILEUDB.cbl

  Copied    <Templates>:reload-files.txt to
<td>/reload/file/STFILEUDB/reload-files.txt

```

```

Copied    <fixed-components>:getfileinfo.cbl to <td>/DML/getfileinfo.cbl

Copied    <fixed-components>:CreateReportFromMVS.sh to
<td>/reload/bin/CreateReportFromMVS.sh

=====

Dynamic_configuration

Copied    <Templates>:File-in-table-STFILEUDB to
/home2/wkb4/param/dynamic-config/File-in-table-STFILEUDB

Copied    <Templates>:../../Conv-ctrl-STFILEUDB to
/home2/wkb4/param/dynamic-config/Conv-ctrl-STFILEUDB=====
=====

post-process

executed <Templates>:post-process-file.sh

/home2/wkb4/param/dynamic-config/Conv-ctrl-STFILEUDB treated

=====

Number of copied files:      30
Number of executed scripts: 1
Number of ignored files:     0

#####
*-----
Components are copied into /home2/wkb4/trf directory
*-----

```

Detailed Processing

This section describes the [Command-Line Syntax](#) used by the File-to-Db2/luw (udb) Converter, and the [Process Steps](#) summary.

The processes required on the source and target platforms concern:

- [Configuring the Environments and Installing the Components,](#)
- [Unloading Data,](#)
- [Transferring the Data,](#)
- [Reloading the Data,](#)
- [Checking the Transfers,](#)

Command-Line Syntax

file.sh

Name

`file.sh` - generate file migration components.

Synopsis

```
file.sh [ [-g] [-m] [-i <installation directory>] <configuration name> | -s
<installation directory> (<configuration name1>,<configuration name2>,...)
]
```

Description

`file.sh` generates the Rehosting Workbench components used to migrate z/OS files to Db2/luw (udb) databases.

Options

Generation Options

-g <configuration name>

Triggers the generation, for the configuration indicated, of the unloading and loading components in \$TMPPROJECT. This generation depends on the information found in the configuration files.

Modification Options

-m <configuration name>

Makes the generated SHELL scripts executable. COBOL programs are adapted to Micro Focus COBOL fixed format. When present, the shell script described in [File Modifying Generated Components](#) is executed.

Installation Option

-i <installation directory> <configuration name>

Places the components in the installation directory. This operation uses the information located in the [file-move-assignation-db2luw.pgm](#) file.

Final Option

-s <installation directory> (<configuration name 1>, <configuration name 2>, ...)

Enables the generation of the COBOL and JCL converter configuration files and DML utilities. These generated files take all of the unitary files of the project.

All configuration files are created in \$PARAM/dynamic-config and DML files in <trf>/DML directory.

Example

```
file.sh -gmi $HOME/trf FTFIL001
```

Unitary Usage Sequence

If the `file.sh` options are used one at a time, they should be used in the following order:

1. => -g
2. => -m
3. => -i
4. => -s (should be executed once steps 1 to 3 have been executed for all configurations).

Process Steps

Configuring the Environments and Installing the Components

This section describes the preparation work on the source and target platforms.

Installing the Unloading Components Under z/OS

The components used for the unloading (generated in `$HOME/trf/unload/file`) should be installed on the source z/OS platform (the generated JCL may need adapting to specific site constraints including JOB cards, library access paths and access paths to input and out put files).

Installing the Reloading Components on the Target Platform

The components used for the reloading (generated in `$HOME/trf/reload/file`) should be installed on the target platform.

The following environment variables should be set on the target platform.

Table 8-19 Target Platform Environment Variables

Variable	Value
DATA_SOURCE	The name of the directory containing the unloaded files transferred from z/OS to be reloaded into Db2/luw (udb) tables.
BIN	The location of the generic reload and control scripts (<code>\$HOME/trf/reload/bin</code>).
TMPPROJECT	The temporary directory.
MT_LOG	Directory to contain execution logs.
DATA_TRANSCODE	Temporary directory used by the file binary data transcoding script (contains temporary files in ASCII format).
DDL	The location of SQL scripts used to create Db2/luw (udb) objects: (<code>\$HOME/trf/SQL/file/<configuration name></code>).
PATH	This UNIX/Linux variable has to contain the directory of Oracle Tuxedo Application Runtime for Batch utilities

In addition, the following variable should be set according to the information in the Oracle Tuxedo Application Rehosting Workbench Installation Guide:

- MT_DB_DNAME
- MT_DB_USER
- MT_DB_PWD

Compiling COBOL Transcoding Programs

The COBOL transcoding programs should be compiled using the options specified in [Compiler Options](#).

Compiling these programs requires the presence of a copy of `CONVERTMW.cpy` adapted to the project.

Unloading Data

To unload each file, a JCL using the IBM IDCAMS REPRO utility is executed. The IDCAMS REPRO utility creates two files for each file:

- a data file,
- a log file,

These unloading JCLs are named `<logical filename>.jclunload`

A return code of 0 is sent on normal job end.

Transferring the Data

The unloaded data files should be transferred between the source z/OS platform and the target UNIX/Linux platform in binary format using the file transfer tools available at the site (CFT, FTP, ...).

The files transferred to the target UNIX/Linux platform should be stored in the `$DATA_SOURCE` directory.

Reloading the Data

The scripts enabling the transcoding and reloading of data are generated in the directory:

```
$HOME/trf/reload/file/<configuration name>/
```

Note: For a file-to-Db2/luw (udb) conversion, the format of the script names is:

```
loadtable-<logical file name>.ksh
```

Transcoding and Reloading Command for Tables

Name

`loadtable` transcode and reload data to table.

Synopsis

`loadtable-<logical file name>.ksh [-t] [-l] [-c: <method>]`

Options

-t

Transcode the file.

-l

Reload the file

-c ftp:<...>:<...>

Implement the verification of the transfer (see [Checking the Transfers](#)).

Checking the Transfers

This check uses the following option of the `loadtable-<logical file name>.ksh`

```
-c ftp:<name of transferred physical file>:<name of FTP log under UNIX>
```

This option verifies, after the reloading, that the physical file transferred from z/OS and the Db2/luw (udb) table reloaded on the target platform contains the same number of records. This check is performed using the FTP log and the execution report of the reloading program. If the number of records is different, an error message is produced.

JCL Translator

Overview

Oracle Tuxedo Rehosting platform is a packaged and comprehensive solution composed of tools (Oracle Tuxedo Application Rehosting Workbench) and runtime components (Oracle Tuxedo Application Runtime for CICS and Batch) which allow its users:

- To perform a replatforming project with minimum risk and cost.
- To run the replatformed applications in the Linux/UNIX standardized environment.

Oracle Tuxedo Application Rehosting Workbench is used only during the replatforming project itself, whereas the Runtime components are used throughout the whole life of the migrated system. Oracle Tuxedo Application Rehosting Workbench is composed of several tools, among which is the JCL Translator.

As its name suggests, the role of this tool is to translate JCLs running on the source platform (z/OS, IBM Job Control Language) into Korn-shell (KSH) shell scripts running on the target platform (Unix or Linux, Korn shell (KSH) dialect with invocations of Oracle Tuxedo Application Runtime for Batch functions) with the same behavior, in the context of other components translated or generated by the other Oracle Tuxedo Application Rehosting Workbench tools. The purpose of this chapter is to describe precisely all the features of the JCL translator.

JCL Translator Definitions

The following terms are used when describing the JCL Translator.

JCL: Job Control Language

A command language for the IBM operating system family. The components written in this language are called JCL jobs or simply JCLs. A source asset to be migrated with Oracle Tuxedo Application Runtime generally contains components of this type.

KSH or Korn Shell

A particular variant in the family of Unix Shell command languages. It is also the name of the shell interpreter itself. A shell script compatible with this language/interpreter is a KSH script. KSH scripts are the components in the target asset which correspond to source JCLs.

JES2

Job Entry Subsystem 2

SYMBOLS

Variables in JCLs, allowing them to be parameterized according to the environment. These variables are handled (substituted) by the JES2 reader.

Card Continuation

JCL cards (commands) of more than 72 characters need to be split over several lines. Lines after the first one are called continuation lines (or cards). The previous line must end before column 72 with an optional separator character (generally the comma character) and the continuation line starts with "/" followed by enough spaces to reach the option area. Comments may be embedded on continuation lines after the option area and must be separated from the latter by at least one space. Example:

```
//SYSIN DD DSN=LIB454R.COMMUN.SER,    this is a comment
//                                DISP=SHR
```

See also the [Concepts](#) described in the Introduction.

General Description and Operation

General Information

Oracle Tuxedo Application Rehosting Workbench JCL Translator handles every required translation in a single pass to:

- Create the structure of the target shell script (execution loop) and populate it with the original sequence of steps.
- Handle conditions and return codes.

- Handle file assignments (DD cards), including access modes and disposition as well as GDG files, spool files, etc.
- Invoke COBOL programs, including through launchers as well as passing parameters.
- Invoke most common utility programs such as file manipulation (IDCAMS, DFSORT, etc.) in a UNIX/Linux manner.
- Invoke other utility programs through a generic interface.
- More generally, implement all the features of the original JCL which are relevant to the target platform.

With the aid of the Oracle Tuxedo Application Runtime for Batch components, the resulting shell scripts can be compiled and run on the target platform with the same behavior as on the source platform, except in some cases detailed in the section [Behavior Coverage](#).

The JCL translator takes as input:

- The abstract syntax trees of the JCL scripts to translate (one or more), stored in the POB files produced by the [Cataloger](#).
- A number of configuration files:
 - (mandatory) the [System Description File](#), which describes where to locate the JCL files and sub-files on the migration platform file system.
 - (mandatory) the conversion configuration file, which specifies the general parameters of the conversion and gives the location of the specific configuration sub-files.
 - (optional) specific configuration sub-files such as the file-to-RDBMS conversion table; see [COBOL and JCL Conversion Guide Files](#) in [File-to-Oracle Converter](#) or [COBOL and JCL Conversion Guide Files](#) in [File-to-Db2/luw \(udb\) Converter](#).

It produces as output:

- An execution log;
- Translated KSH components in their textual representation: scripts (main files) and sub-files for procedures, INCLUDE files and possibly SYSIN files.

The JCL translation process acts individually and separately on each JCL. However, it is not suitable to run the JCL Translator concurrently on multiple JCLs at the same time because, if several of these JCL contain the same sub-file, then their translator instances may want to write the translation of this sub-file at the same time, with the risk of corrupting it. In practice, JCL translation is fast and incremental, so there is no real need to accelerate it using multiple processors.

Behavior Coverage

For a detailed description of which JCL cards, parameters, options and utility programs are supported by the JCL translator and the underlying Oracle Tuxedo Application Runtime for Batch components, please refer to the chapter [Z/OS JCL in the Oracle Tuxedo Application Runtime Batch Environment](#) in the book [Oracle Tuxedo Application Runtime for Batch Reference Guide](#).

Description of Input Components

The input components are all the JCL scripts (main files) in the asset, after they have been parsed by the cataloger. In fact, the JCL translator loads the POB files for the scripts, not their source files. In addition to the restrictions imposed by the cataloger (no multiple jobs per JCL, etc.; see the [Cataloger](#)), the following rules must be respected before attempting the JCL translation:

- All the anomalies reported by the cataloger must be fixed. Otherwise, there is a risk that the translation is incorrect, or even that the translator fails (crashes). In fact, the translator will refuse to translate any JCL which contains a FATAL anomaly. But even ERRORS or simple WARNINGS may cause trouble, so it is strongly advised to fix all anomalies.
- The data migration process must have been run before JCL translation is started, because the latter depends on the former, for instance to decide which files will be migrated into relational DB tables; see the Oracle Tuxedo Application Runtime Process Guide for more details. This dependency is concretized in the fact that the file migration tools generate some of the configuration files read by the JCL translator.

Description of the Configuration Files

The JCL translator is driven by two parameter files, the system description file and the JCL-translation configuration file.

The System Description File

The [System Description File](#) describes the location, type and possible dependencies of all the source files in the asset to process. As such, it is the key by which the cataloger, but also all of the Rehosting Workbench tools, including the JCL translator, can access these source files and the corresponding components.

The following component types are relevant to the JCL translator:

Table 9-1 JCL Translator Component Types

JCL	Main JCL files, defining one or more JCL jobs.
JCL-Lib	JCL sub-files, either defining procedures invoked by EXEC or containing statements invoked by INCLUDE.
JCL-Sysin	SYSIN files used by utility programs or program launchers in JCL scripts. Not all SYSIN files are required by the parser/cataloger.

The following (global and/or local) options are relevant to the JCL translator

Table 9-2 JCL Translator Global and Local Options

JCL-globals	List of pairs var-name = var-value separated by commas	var-name is a symbol (or string interpreted as a symbol) and var-value is a string. When parsing a JCL script, the parser simulates the JCL-variable substitution process performed by JES2. The name-value pairs given here are used to substitute global variables (as opposed to parameters, etc.). The parser reports an error when it cannot find a suitable value for a variable.
strict-jcl-libraries	None (Boolean flag)	The presence of this option influences how SYSIN files referenced by a JCL are searched in the whole system. See chapter Normal Sub-File Search operation in the Cataloger chapter.
jclz-launcher-spec[s]-file	string	Path of the JCL-launcher specification file to use for this system or directory; see the section JCL-Launcher Specification Files in the Cataloger chapter for more information on the contents and use of these files.

Listing 9-1 JCL Translator global and local options examples

```
options jcl-globals = VAR01 = "T23RT", PARM = "USERT".  
  
directory "Sysin/BPRO.PARMIMS" logical-name "BPRO.BXZ.PARMFIX" type  
JCL-SYSIN files "*.sysin".  
  
directory "JCL/BPRO" type JCL files "*.jcl" libraries "Sysin/BPRO.PARMIMS".
```

The JCL-Translation Configuration File

The contents of the JCL-translation configuration file is a list of "assignments" of the form parameter-name = parameter-value. Some of these assignments, those in which the parameter-value is a sequence of strings, must be terminated by a period; the strings are separated by commas. The following parameters are available:

Table 9-3 JCL-translation Configuration File Contents

root-skeleton	String	Path of the root directory in which the hierarchy of generated KSH scripts will be stored. This parameter is mandatory, no default value exists. The given path may be either absolute or relative to the location of the system description file.
target-proc	String	Path of the root directory in which the hierarchy of the sub-files extracted from generated KSH scripts (PROCs, INCLUDEs, possibly some SYSINs) will be stored. The default value is the root-skeleton. The given path may be either absolute or relative to the location of the system description file.
use-sort	Symbol	Target sort utility to use either <code>mf-sort</code> or <code>sync-sort</code> , or <code>cit-sort</code> (in release 11.1.1.2.2). Default is <code>mf-sort</code> .
var-dataroot	String	Root directory for target permanent data files. Default is "\${DATA}/", which allows to easily run the target shell scripts on different environments, such as test and production.
var-tmp	String	Root directory for target temporary data files. Default is "\${TMP}/".
var-spool	String	Root directory for target spool (print) files. Default is "\${SPOOL}/".
top-skeleton	String	String which will be inserted as header in all target KSH scripts. It is suggested that it contains only comments, not executable statements. It can be used as a standardized template for maintenance information. Default is no header (empty string).
bottom-skeleton	String	String which will be inserted as trailer in all target KSH scripts. It is suggested that it contains only comments, not executable statements. It can be used as a standardized template for maintenance information such as CVS log. Default is no trailer (empty string).

Table 9-3 JCL-translation Configuration File Contents

file-list-in-table	String	Path of the file containing a list of paths for files containing the DSNs of the (source) data files to be migrated into Oracle tables, if any. This file, and the files it refers to, are automatically generated by the data-migration tools (File-To-Oracle). The path of this file, and the paths it contains, may be given either in absolute form or relative to (the directory containing) the system description file.
suffix-skeleton	String	File extension (suffix) of the target KSH scripts. Default is ".ksh". Note that the extension for PROCs, INCLUDEs and SYSINs is not customizable.
set-no-delete-fsn	Sequence of FSNs	List of file assignments to keep in the target KSH scripts, when they would normally be removed. See set-delete-fsn and set-no-delete-fsn for more details.
set-delete-fsn	Sequence of FSNs	List of file assignments to remove from the target KSH scripts, when they would normally be kept. See set-delete-fsn and set-no-delete-fsn for more details.

set-delete-fsn and set-no-delete-fsn

For the last two parameter names, `set-delete-fsn` and `set-no-delete-fsn`, the syntax of the parameter value is as follows:

```
ddname ( program-name, program-name, ... ) , ddname (program-name ... ) , ... .
```

Note: The DD-names and program-names are written as symbols using the source syntax. The separators (parentheses, commas and trailing period) are mandatory.

The semantics of this parameter value is as follows: when a DD name is associated with an explicit list of program names, it stands for exactly these logical files (a logical file is some FD or SD in some program); when a DD-name is associated with no program name, it stands for all the logical files (FD or SD) which have these names in all programs in the asset. After expansion, we get two lists of logical files, the "to-keep" and the "to-remove". The "to-keep" list is subtracted from the "to-remove" list; then any file assignment (DD card) pertaining to some "to-remove" logical file is not translated into the target KSH script. This allows to "clean up" the target scripts from file assignments which are not directly useful to the invoked programs, such as temporary sort files, physical database files, etc.

Description of Output Files

Translated KSH Scripts and Sub-Files

KSH Version

The generated scripts are certified to be compatible only with ksh88 or pdksh (public-domain KSH). Note that the Oracle Tuxedo Application Runtime for Batch components invoked by the script may have more stringent requirements regarding the underlying KSH engine.

File Structure, Naming Scheme and Sub-File Handling

For each (main) JCL file (job), the JCL translator produces one (main) KSH file (script) with the same base name and extension as specified with the suffix-skeleton configuration parameter (see [JCL-translation Configuration File Contents](#)). This file is placed under the hierarchy rooted at the directory specified with the root-skeleton configuration parameter, in the same relative sub-directory as was the source file under the system root. If the target file existed before the translation, it is overwritten.

- For each sub-file which was included into the main file by the JCL parser (procedure, include file, SYSIN...), the JCL translator produces a corresponding translated sub-file with the same base name and fixed file extension (".proc" for a procedure, ".incl" for an include file, ...). This file is placed under the hierarchy rooted at the directory specified with the target-proc configuration parameter, in the same relative sub-directory as was the source file under the system root. If the target file existed before the translation, it is overwritten.

If a sub-file referenced by some main file is absent from the source asset when the main file is processed by the cataloger, the latter will complain in the log (with a message of the form `"*Warning*: ZZZ file XXXXXX not found in JCL file YYYYYY"`, where ZZZ is PROC or INCL or SYSIN) and an anomaly will be reported. In this case, the JCL translator will refuse to process the (incomplete) main file or produce a non-working, incomplete KSH script.

Procedures are invoked using the `m_ProcInclude` Oracle Tuxedo Application Runtime for Batch function. The target sub-file is not "executable" outside of this function and, more generally, an Oracle Tuxedo Application Runtime for Batch-generated KSH script. The procedure contents are inserted into the script structure in the "conversion" phase of job processing, see the Oracle Tuxedo Application Runtime for Batch User Guide.

Similarly, include files are invoked using the `m_ShellInclude` Oracle Tuxedo Application Runtime for Batch function and cannot be used outside this context. Most include files are

translated into separate sub-files; however, in some cases, when the include file contains an EXEC card but does not define a (sequence of) complete steps(s), the extraction cannot be performed and the contents of the original sub-file is translated as if it came from the main file.

SYSIN files and their contents are either directly translated into KSH statements (function invocations) or handled as standard file assignments (see below for more details).

In-stream procedures and SYSIN files are handled as their out-of-stream sisters, as far as translation is concerned, but of course they remain inside the main target file. In-stream procedures are moved to the end of the main file; in-stream SYSINs remain within the corresponding step.

Handling of JCL and KSH Variables

The z/OS variables used in the original JCL are translated into special symbols of the form `$_[PARM]`, which are "statically" substituted in the manner of JES2 in the conversion phase, just before the execution phase. See the Oracle Tuxedo Application Runtime for Batch User Guide for more details.

Besides this, the generated KSH script and the components of the Oracle Tuxedo Application Runtime for Batch use a small number of KSH and environment variables, for instance:

- environment variables for file-paths prefixes, such as `$_{DATA}`;
- shell variables for return codes, step labels at which to jump (`$_JUMP_LABEL`), etc.

These variables are substituted at run-time, in the execution phase.

Script Structure

The target KSH script is organized around an execution loop which allows to implement any kind of control flow, including "backward" jumps; it also makes it possible to (re)start the script at any step. The different steps in the original JCL are implemented in the same order in the loop body, and delimited by their label (the same as in the source JCL) and the case-element terminator. Technical support and bookkeeping operations are inserted at the beginning and end of each step, as well as at the beginning and end of the whole script. They allow for instance to handle the control flow (condition codes, jumps, etc.).

Script Layout

The following JCL cards or constructs are translated into invocations of appropriate Oracle Tuxedo Application Runtime for Batch functions: `JOB`, `JCLLIB`, `SET`, `DD`, `proc-overriding DD`,

EXEC (program), EXEC (procedure), INCLUDE, IF/THEN/ ELSE/ENDIF from the original JCL, IF from IDCAMS command files and the COND option.

Comments in the source JCL are translated into KSH comments, which the translator tries to place as close as possible to "the right place". However, comments in the continuation area (i.e., after the first space after the parameters) are ignored and not reproduced in the result script.

The // card which terminates the job is translated as a jump to the end of the script.

As mentioned above, an EXEC procedure card is translated into an invocation of the `m_ProcInclude` Oracle Tuxedo Application Runtime for Batch function, and an INCLUDE card is translated into an invocation of the `m_ShellInclude` function. SYSIN files and references (DD cards) are handled in two different ways:

- SYSIN for usual applicative programs: they are handled as any other data file, the DD card being translated into an invocation of the `m_FileAssign` function;
- SYSIN for utility programs: depending on which utility program is considered, its SYSIN can either be handled as any other data file-the interpretation of its contents being left to the program implementing the same operation as the utility program, possibly a component of the Oracle Tuxedo Application Runtime for Batch or a user-supplied program-or directly translated into invocation of appropriate Oracle Tuxedo Application Runtime for Batch functions-for instance, a REPRO command for the IDCAMS utility is generally translated into an invocation of the `m_FileLoad` function which performs the file copy.

In-stream data files (DD * cards and associated contents), which are generally text-like files, are also implemented inside the target script. The EBCDIC-to-ASCII conversion of their contents is performed at the same time as the source JCL file is converted, during the transfer to the migration platform.

Execution Logs

The execution log reports about the progress of the translation process. Its structure is as follows:

- At the beginning of the log file:
 - CMD: ..., the translation command with all its options.
 - Parsing system file: ..., a reference to the system description file.
 - MY_DIR = ..., location of the execution scripts.
 - Current OS is ..., source OS defined in the system description file (message to remove).
 - Parsing config file : ..., a reference to the configuration file.

- File list in table : ..., a reference to the file listing the data files migrated into RDBMS tables (if any).
- For each JCL to translate:
 - Creating target file, creation of the target KSH file (empty at this point, acts as a lock to synchronize concurrent processes).
 - Loading, the POB file associated with the JCL to translate is loaded in memory.
 - (In case of version mismatch between the cataloguer and the translator) Patching, adapts the loaded POB to the translator (if necessary).
 - Printing, after the target code is created in memory on top of the AST, it is printed into the target file (and sub-files).
- At the end of the log file:
 - Rest in peace, ART..., normal end of the translation process.
 - continue ..., in a controlled way, the translation process is terminated and then restarted, to free memory.
 - Error ..., abnormal termination of the translator.

Listing 9-2 Example Log file:

```

CMD : /refine/launch.bash jclz-unix -archi64 -s ../param/sys.desc -c
../param/config.desc
MetaWorld starter
(funcall MAKE-TRANSLATE)
*interactive-mode-string* *UNDEFINED*
Parsing system file : ../param/system.desc
MY_DIR = /refine
Current OS is IBM-MF
Parsing config file : ../param/config-trad-JCL.desc
File list in table :
/workspace/FTJCL01/source/../param/file-list-in-table.txt
Creating target file /workspace/FTJCL01/trf-jcl/JCL/BBSAJ001.ksh ...
At 12:02:08, Loading /workspace/FTJCL01/source/JCL/pob/BBSAJ001.jcl.pob...
```

```
Printing /workspace/FTJCL01/trf-jcl/JCL/BBSAJ001.ksh
done
Creating target file /workspace/FTJCL01/trf-jcl/JCL/BDBAJ001.ksh ...
At 12:02:08, Loading /workspace/FTJCL01/source/JCL/pob/BDBAJ001.jcl.pob...
Patching because current version ("9.9.1") and POB version ("0.8.6") are not
the same
Printing /workspace/FTJCL01/trf-jcl/JCL/BDBAJ001.ksh
done
Rest in peace, ART...
```

Detailed Operation

General Information

When the JCL translator starts, it reads and checks the various configuration files, starting with the main one. If any inconsistency is detected at this stage, one or more error messages are printed and the translator exits. Otherwise, the translator uses both command-line options and configuration-file options to set its internal parameters, including the list of (source) JCLs to process. Then it proceeds to handle each of these JCLs in turn; for each JCL:

1. According to the make-like, incremental behavior of the translator, it checks whether the target KSH script already exists and is up-to-date with reference to the POB file for its corresponding source JCL. If so, the translator skips to the next JCL. Otherwise, it continues with the next step.
2. The POB file for the JCL is loaded. The translator then checks whether it contains anomalies of severity FATAL. If so, it prints out a warning message and skips to the next JCL. Otherwise, it continues with the next step.
3. The AST of the JCL is then traversed, and the target code for each construct is created and assembled with that of surrounding nodes. Code to print "out-of-stream", in sub-files or at the end of the target main file, is marked in consequence.
4. The (text of the) target code is then printed out in the target KSH file and sub-files.

5. Lastly, if the post-translation file is specified in the configuration file, it is exercised by the post-translator on the main target KSH file and on all target sub-files.

The translator cannot be executed by several concurrent processes at the same time, because there is a risk that two different processes want to write the same sub-file at the same time.

Command-line Syntax

The Refine Launcher Interface

The JCL translator is designed to be run through the refine command, which is the generic Oracle Tuxedo Application Rehosting Workbench launcher. It handles various aspects of the operation of these tools, such as execution log management and incremental/repetitive operations. See [Command-line Syntax](#) in the Cataloger chapter.

The jclz-unix Command

Syntax

```
$REFINEDIR/refine jclz-unix [ launcher-options... ] \  
    ( -s | -system-desc-file ) system-desc-path \  
    ( -c | -config ) main-config-file-path \  
    ( source-file-path | ( -f | -file | -file-list-file ) file-of-files )...
```

The launcher options are described in [Command-line Syntax](#) in the Cataloger chapter. The mandatory options are:

(-s | -system-desc-file) system-desc-path

Specifies the location of the system description file. As usual for Unix/Linux commands, the given path can be absolute or relative to the current working directory. Note that many other paths used by many Oracle Tuxedo Application Rehosting Workbench tools are then derived from the location of this file, including that of the main configuration file (see next item); this makes it easy to run the same command from different working directories.

(-c | -config) main-config-file-path

Specifies the location of the main conversion configuration file. The given path can be either an absolute path or a relative path; in the latter case, it is relative to the directory containing the system description file, as usual for the Rehosting Workbench tools.

The generic options which define which source JCLs to process are:

source-file-path

Adds to the work-list the JCL source file designated by this path. The path must be given as relative to the root directory of the system, \$SYSROOT, even if the current working directory is different.

(-f | -file | -file-list-file) file-of-files

Adds to the work-list the JCL source files listed in the file designated by this path. The file-of-files itself may be located anywhere, and its path is either absolute or relative to the current working directory. The JCL source files listed in this file though, must be given relative to the root directory of the system.

You can give as many individual JCLs and/or files-of-files as you wish. The work-list is built when the command line is analyzed by the JCL translator, see the detailed description above.

Repetitive and Incremental Operation

Even with the powerful computing platforms easily available nowadays, processing a complete asset using the Rehosting Workbench remains a computing-intensive, long-running, memory-consuming task. Oracle Tuxedo Application Rehosting Workbench tools are hence designed to be easily stoppable and restartable and, thanks to a make-like mechanism, not repeat any piece of work which has already been done. This allows efficient operation in all phases of a migration project.

Initial Processing: Repetitive Operation

In the initial phase, starting with a completely fresh asset up to the end of the first conversion / translation / generation, with a stable asset, the make-like mechanism is used to allow repetitive operation, as follows:

1. When some tool-say, the JCL translator-starts, it begins with studying the current state of the asset (source files and target files such as the target JCL files) and determining what work remains to be done to reach a complete and consistent set of results. It then undertakes this work, producing more and more result files.
2. The Oracle Tuxedo Application Rehosting Workbench process consumes more and more memory. Regularly, it checks whether the available physical memory drops below the threshold set by the `minimum-free-ram-percent` option in the system description file.
3. If the work to do is complete before memory runs out, the process definitely stops.
4. Otherwise, the process stops but restarts immediately, after having freed memory. Going back to step 1 above, there is less work to do, so that the process eventually terminates.

This mode is particularly well suited for tools or commands which operate globally on the whole asset, such as the cataloger, but it is also useful for component-specific tools such as the JCL translator. This is the normal mode of operation for the Rehosting Workbench tools and there is no specific action required to activate it.

Changes in the Asset: Incremental Operation

The JCL translator knows the dependencies between the various components (main JCL files and sub-files) and associated result files (POB files, target KSH files). Using this information, it is able to react incrementally when some change occurs in the asset, i.e. when a JCL source file is added, modified or removed: the cataloger reparses the affected JCLs, and then the JCL translator re-translates only the reparsed JCLs. Again, this is the normal mode of operation for the Rehosting Workbench tools and there is no specific action required to activate it.

Concurrent Operation

As mentioned above, it is not advised to run the JCL translator concurrently with several processes, because different processes might wish to write the same target sub-file at the same time. However, JCL translation is fast and there is generally no need to make it faster using concurrent processing.

Use through make

to be completed

Frequently Asked Questions

When do I translate anew some JCL?

- When you have modified some options in the configuration file:
 - The root directory for the target components
 - The target sort-utility choice
 - The header and footer skeletons
 - Etc.
- When the JCL or one of its sub-files (procedure, include file or SYSIN) has just been added, modified or deleted. In principle, incremental operation of the cataloger and translator ensure that this situation is handled automatically: simply re-run both of them.

How do I force the (re)translation of a JCL?

Either:

- Delete the target KSH, or
- Change the contents of the JCL, or simply change its modification date (using the touch Unix command).

The former solution is recommended, because it is less "intrusive" on the source asset than the other.

I deleted a JCL. Why is the corresponding KSH still present?

You have to delete the latter by hand, together with its sub-files such as procedures and include files (if they are not used by any other KSH).

I run the translator but it produces no translation

- Check whether the translator produces any anomaly messages (see previous appendix); resolve reported anomalies or conflicts.
- Check that you indeed have components to translate, i.e. new or modified JCLs. Remember, the translator is incremental.

The procedures are not included in the JCLs, and hence in the KSH

In the system description file, check the following:

- You have defined a directory of type JCL-LIB containing these procedures.
- In the definition of the directory(ies) containing the JCLs, you have a libraries clause pointing to the procedure directory(ies).

Where do I find the translated procedures?

They are located in the directory defined by the target-proc (or possibly root-skeleton) option of the translator configuration file. To this path is appended the relative path of the procedures under the root directory of the source asset (as defined in the system description file). For instance, with the following directives:

- In the translator configuration file:

```
target-proc=/Workspace/Master-Proc
```

- In the system description file:

```
directory "PROC" type JCL-LIB files "*.proc".
```

```
directory "INCLUDE" type JCL-LIB files "*.incl".
```

```
directory "JCL" type JCL files "*.jcl" libraries "SYSIN", "INCLUDE",  
"PROC".
```

The procedures will be located in the directory `/Workspace/Master-Proc/PROC`.

Why are some FSNs lost during translation?

Check that the FSNs you want to keep are not in the list of FSNs to delete (set-delete-fsn option in the translator configuration file). If so, remove them from this list or, if you still want to delete some of them but not all ("wild card" in the set-delete-fsn list), add those you want to keep into the set-keep-fsn list.

COBOL Converter

The role of this tool is to convert COBOL programs running on the source platform (z/OS, IBM COBOL dialect) into COBOL programs running on the target platform (UNIX or Linux, Micro-Focus COBOL dialect or COBOL-IT) whilst maintaining the same behavior of the application. The conversion is performed in the context of other components translated or generated by the other Oracle Tuxedo Application Rehosting Workbench tools.

The purpose of this document is to describe precisely all the features of the COBOL Converter.

Overview of the COBOL Converter

Scope

The Refine COBOL converter handles the following transformations in a single pass:

- COBOL dialectal correction (from z/OS COBOL to the target COBOL (MicroFocus or COBOL-IT)).
- Adaptation to target platform UNIX interfaces (e.g., replacement of SEQUENTIAL files by more efficient LINE SEQUENTIAL files or handling of printer control characters).
- Embedded SQL conversion (from DB2 to Oracle DBMS), including the interface with the host program (SQLCODE, host variables...).
- Any code adaptation consecutive to supported reengineering options such as file to RDBMS conversion or component renaming.

- Normalization of the EXEC CICS statements and the programs to make them suitable for the run-time EXEC CICS preprocessor.

The resulting programs can be compiled and run on the target platform with the same behavior as on the source platform, except in some cases detailed in [Scope](#).

Inputs

The COBOL converter takes as input:

- The abstract syntax trees of the COBOL programs to convert (one or more), stored in the POB files produced by the Rehosting Workbench [Cataloger](#);
- A number of configuration files:
 - (mandatory) the system description file, which describes where to locate the COBOL programs on the migration platform file system.
 - (mandatory) the conversion configuration file, which specifies the general parameters of the conversion and gives the location of the specific configuration sub-files.
 - (optional) specific configuration sub-files such as the variable-renaming table, the component-renaming table, the file-to-RDBMS conversion table, etc.; see [Main Conversion Configuration File](#).

Outputs

It produces as output:

- An execution log.
- Converted COBOL components in their textual representation: programs and copy files.
- Additional files such as dependence files for make.

Conversion Phases

The COBOL conversion process is logically divided in two phases:

- Individual conversion: this phase acts "locally" and separately on each program. It converts the AST in its internal form, and then prints out the converted form of both the main source file (in the target directory) and the copy files (in a private subdirectory of the target directory). It also runs the post-translator on each of these files, if requested.
- Copy file reconciliation: the copy files have been converted separately for each program, but the objective is a single set of copy files for all programs. The reconciliation process

builds this set by "factorizing" all privately-converted copy files and storing them in a common file base, taking care to not mix different versions of the same copy file (these different versions may come from context-dependent conversions).

The individual conversion phase can run concurrently on several programs but, since the copy-reconciliation phase updates the global copy file base, it must run as a single process, possibly incrementally. This dictates the possible execution modes of the COBOL converter; see [Command-Line Syntax](#) for more details.

Restrictions and Limitations

By definition, the Rehosting Workbench COBOL Converter accepts only those programs accepted by the Rehosting Workbench COBOL [Cataloger](#), but imposes no further restriction on entry.

The resulting programs can be compiled and run on the target platform with the same behavior as on the source platform, except for the following potential pitfalls for which we take no responsibility:

- Some target-compiler options, such as the IBMCOMP option of MicroFocus, must be set as mandatory in this document (see [Compiler Options](#)).
- Problems with data files which would be incorrectly migrated because incorrect information was supplied to the data-migration tools.
- Differences of behavior in "incorrect" pieces of code, such as accessing an array with an out-of-bounds index value, or pieces of code which work "by chance", such as code assuming that two records which are adjacent in the Working-Storage Section are also adjacent in memory.
- Issues related to the semantic nature of some COBOL variables or requiring a deep, semantic understanding of the program. Some of these issues are described later in this chapter.

Use of COMP-5 Type on Linux Platforms

The Oracle Tuxedo Application Rehosting Workbench COBOL converter translates "portable" binary integer types (BINARY, COMP, COMP-4) to the native binary type COMP-5. This is in order to ensure compatibility with sub-programs written in C such as those in the transaction processing framework (see the CICS section of the Oracle Tuxedo Application Rehosting Workbench Reference Guide), and to improve execution performance. This may cause problems when the target platform does not have the same "endianness" as the source platform, in particular

on Linux and Intel platforms (the Intel processor line is little-endian whereas the zSeries processor is big-endian; most other processors, such as IBM pSeries and HP-RISC, are also big-endian). Indeed, in this case, the order of bytes in a binary variable is reversed with respect to the source platform. This can lead to different behavior when such a binary variable is redefined by a character (PIC X) variable and this redefinition is used to access the individual bytes in the binary variable. For Example:

Listing 10-1 Binary Field Manipulation Example

```
WORKING-STORAGE SECTION.  
  
01 FILLER.  
  
    02 BINVAR                      PIC S9(9) COMP.  
  
    02 CHARVAR REDEFINES BINVAR PIC X(4).  
  
PROCEDURE DIVISION.  
  
    ...  
  
    MOVE ... TO BINVAR  
  
    IF CHARVAR(1:1) = ... THEN ...
```

On a big-endian machine such as the z/OS hardware, CHARVAR(1:1) contains the most significant (higher-order) byte of BINVAR. However, on a little-endian machine, with the same code, CHARVAR(1:1) will contain the least significant (lower-order) byte of BINVAR; this is definitely a change of behavior and will probably lead to different observable results. However, the Rehosting Workbench COBOL Converter is unable to detect and fix all occurrences of this situation (the example above is "obvious", but there exists many much more complex cases); these must be handled manually.

Use of COMP-5 Type and the TRUNC Compiler Option

As mentioned in the previous paragraph, the Rehosting Workbench COBOL converter translates portable binary integer types (BINARY, COMP, COMP-4) to the native binary type COMP-5. In addition to endianness problems, this may cause another kind of difference of behavior for applications which were compiled with the (default) TRUNC(STD) option on the source platform – this option corresponds to the TRUNC option of Micro Focus COBOL or the

BINARY-TRUNCATE option of COBOL-IT. Indeed, both on the source and on the target platforms, the portable binary types obey this option whereas the native type does not. In our opinion, the probability of observing a real difference of behavior is very low, because in general, binary-integer variables are used to hold "control" values (loop counters, array indices, etc.) rather than applicative values. In any case, if differences of behavior are observed, it is up to the Rehosting Workbench user to deal with them, either by accepting them or by manually correcting them, for instance by returning a few selected variables to their original binary type.

EBCDIC-to-ASCII Conversion Issues

For reasons of efficiency and compatibility with native utility programs on the target platform—for instance, simply browsing through a data file on the terminal—one of the fundamental design choices for the migration performed by the Rehosting Workbench is to convert textual (alphabetic) data from the native character set of the source platform (EBCDIC, or one of its variants) into the native character set of the target platform (ASCII, or one of its variants). This common-sense decision however has important consequences on the migration process:

- The migration of the data itself must be handled with great care. In particular, the actual EBCDIC-to-ASCII conversion table used for your specific project must take into account the particular non-standard characters you use on your screens (e.g. accented letters, the Euro or pound sign, etc.), together with their encoding in the source character set, and make sure that they are appropriately transcoded into the corresponding characters in the target character set. See the Oracle Tuxedo Application Rehosting Workbench Process Guide and the Rehosting Workbench Data Migration Tools documentation in this guide and in the Oracle Tuxedo Application Rehosting Workbench Reference Guide.
- An important issue in the data migration process is that the EBCDIC-to-ASCII conversion must not apply to non-textual data such as binary, packed-BCD or floating-point data. This requires that each data structure (file, opaque SQL column, etc.) be described by a detailed and precise COBOL record exhibiting all these non-text fields. See the Rehosting Workbench Process Guide and the Rehosting Workbench Data Migration Tools documentation.
- The same EBCDIC-to-ASCII conversion is applied to the source file of the components, so that they appear visually correct on the target platform. This is important for their correct maintenance. This also means that it is applied to the contents of literal character strings in the programs or JCLs.

In most cases, if you comply correctly with these directives, the resulting application will run smoothly. There is one issue however, for which no efficient solution can be found: the collating sequences of the EBCDIC and ASCII character sets are not quite the same, and this may lead to

different behavior in sorting and string comparisons. In most cases, there is no problem, because you sort or compare "homogeneous" data such as names (alphabetic) or dates (numeric); only special characters such as accented letters may sort a bit differently but still satisfactorily. However, in cases when you sort or compare data which contains mixed letters and digits, you may find differences of behavior, because letters sort before digits in EBCDIC and after digits in ASCII. One typical example is when you compute a key for some type of data (account number, etc.) using both digits and letters. The COBOL converter cannot handle such issues because these are dynamic issues related to the contents of COBOL variables, not static issues related to their declarations.

Literal Constants: Characters or Numbers?

As mentioned above, string or character literals in COBOL programs, including hexadecimal string literals, are subject to EBCDIC-to-ASCII conversion. This is legitimate when these literals denote texts or pieces of text. Sometimes however, such constant values denote (numeric) codes such as file status codes, condition codes, CICS-related values, etc. In this case, it is generally not appropriate to apply EBCDIC-to-ASCII conversion to these values. However, the COBOL converter, like any automatic tool, cannot reliably "guess" the semantic nature of a COBOL variable or literal, so it cannot handle itself these exceptions; this will have to be done manually using post-translation, see [post-translation-file Clause](#)).

Note: CICS-related values and codes defined as character literals in standard copy files cause no trouble, because the Rehosting Workbench and Oracle Tuxedo Application Runtime for CICS come with pre-translated, validated versions of these copy files. Only user-defined constants may cause trouble.

Use of Floating-Point Variables

Source floating-point variables (`COMP-1` and `COMP-2`) types are "translated" to the same types on the target platform. Given this, the Micro Focus COBOL compiler and run-time system offer the possibility to use floating-point data (`COMP-1` and `COMP-2` variables) in either the IBM hexadecimal format or the native (IEEE 754) format. If the `NONATIVEFLOATINGPOINT` option is set at compile time (which is true by default), then the floating-point format is selected at run-time, depending on the `MAINFRAME_FLOATING_POINT` environment variable and/or the `mainframe_floating_point` tunable:

- `MAINFRAME_FLOATING_POINT` environment variable set, or `mainframe_floating_point` tunable set to true: the IBM format will be used.

- `MAINFRAME_FLOATING_POINT` environment variable unset, and `mainframe_floating_point` tunable unset or set to false: the native format will be used.

In the first case, the Micro Focus run-time system will ensure that you will observe no difference of behavior. However, this is at the expense of run-time efficiency, because the handling of this format is done entirely in software, whereas the native format is directly supported by the processor. Furthermore, this format is not directly compatible with the Oracle floating-point data types (`BINARY_FLOAT` and `BINARY_DOUBLE`) and cannot be converted to other numeric types by the Oracle engine; in fact, the only thing you can do with it is store it in opaque columns (`RAW(4)` and `RAW(8)`, respectively), which forbids using such values in SQL code.

In consequence, we recommend that, at migration time or later, you consider using the native IEEE754 floating-point format, more efficient, more portable (defined by an international standard) and more compatible, if only with Oracle. Of course, because:

1. The representation of single- and double-precision floating-point values are not the same in this format as in the source IBM format,
2. The source and target compilers may make different choices regarding arithmetic expressions using floating-point variables (order of computation, precision of intermediate variables, rounding mode, etc.),
3. The textual, printable representation of the same floating-point value may be different on both platforms (use of scientific notation, number of digits before and after the decimal point, etc.),

you will probably observe differences of behavior between the original and migrated applications. However, in our opinion, these differences of behavior are largely acceptable, if you keep in mind that floating-point arithmetic is only an approximation of the mathematical exactness: you will simply get a different approximation on the target machine than on the source one...

Note: To help you deal with this issue, we performed various experiments using varied floating point-values and computations, and we found out that:

- On the source platform, `COMP-1` and `COMP-2` types have the same representation range, from about 10^{-79} to 10^{76} , whereas on the target platforms (which all natively support the IEEE 754 format), the range for `COMP-1` is about from 10^{-45} to 10^{38} and the range for `COMP-2` is about from 10^{-323} to 10^{308} . So the tradeoff between range and precision is different on both platforms.

When the same computations are performed on ranges available on both the source and target platforms, the relative error between the observed results (as printed by `DISPLAY`) is always less

than 10^{-6} when using COMP-1 variables and less than 10^{-14} using COMP-2 variables. This is not a definitive proof that everything works fine, but it is at least an encouraging indication.

Given these results, it seems that one can always reproduce the same behavior on the target as on the source, up to insignificant approximations, possibly by replacing some COMP-1 variables by COMP-2 ones.

Note: If you decide to go with the native IEEE 754 format, we recommend that you set the NATIVEFLOATINGPOINT compiler option, which forces the use of this format at compile-time, regardless of run-time options and tunables. Thus, you will save the run-time format tests.

REWRITE Operations on LINE SEQUENTIAL Files

By default, data files which are SEQUENTIAL on the source platform are translated into LINE SEQUENTIAL files on the target platform, to be more "usable". In general, this is a good choice and such files are well supported by the target COBOL system. However, there is a catch: since such files are inherently of variable record size, a REWRITE operation may cause unpredictable results and differences of behavior (see the Micro Focus and COBOL-IT documentation). If you are not sure that REWRITE operations on a given SEQUENTIAL file would always succeed if that file is turned into a LINE SEQUENTIAL one, we advise to keep it purely SEQUENTIAL; this can be done by inserting its description in the configuration sub-file referenced by the pure-seq-map-file clause below.

To ease the handling of this problem, in a future version, the Rehosting Workbench cataloger will produce the list of SEQUENTIAL logical files which incur a REWRITE operation.

Pointer Manipulation

Pointer Size Changes: Beware of Redefinitions

On the source platform, a variable of type POINTER occupies 4 bytes in memory (32 bits); on all the supported target platforms, based on 64-bit Operating Systems, such a variable occupies 8 bytes. This may lead to various kinds of differences of behavior for which we take no responsibility:

- Technical redefinitions: if a POINTER variable is directly redefined by a PIC X(4) or PIC S9(9) COMP variable used to manipulate the representation of the pointer values, the redefining variable and the code dealing with it will have to be manually rewritten. However, we strongly discourage such machine-dependent "hacks".

- **Structure alignments:** if a `POINTER` variable is part of a structure containing variants (redefinitions), and if the different variants (sub-structures) are designed so that one particular field of one variant must be aligned with (have the same location as) some other field in some other variant, then this property must be maintained after the `POINTER` variable changes size: compensation fillers must be inserted, etc. Again, this must be handled manually. Note that such intended alignments must be maintained across redefinitions, but also across `MOVES` to other structures.
- **Structure size:** if a `POINTER` variable is part of a structure which is moved to some unstructured `PIC X(...)` variable which was big enough to hold the structure before the `POINTER` variable changes size, then you must make sure that it is still the case after the change.

Linkage-Section Arguments with NULL Address

On both the source and target platforms, a program parameter (defined in the Linkage Section and listed in the `USING` clause of the procedure division) which is not actually passed by the caller, either because of an explicit `OMITTED` item is passed instead or because the caller passes less arguments than the callee expects, appears to have a `NULL` address in the callee. So it is quite legal, and in fact recommended, to check whether the `ADDRESS` of some parameter is `NULL` before accessing the value of this parameter.

- However, when the callee fails to check the parameter address and the actual address is `NULL`, the source and target platforms may behave differently. For instance:
 - On `z/OS` and `AIX`, `NULL` is address 0 and this is considered as a legal address, so when the parameter is accessed, you get whatever is stored at that address (possibly with unpredictable results).
 - On `Linux` however, although `NULL` is also address 0, this is *not* considered as a legal address, so when the parameter is accessed, the program crashes.

It is not possible to automatically handle this situation and the associated differences of behavior, because even if the converter could insert address checks, what should it do when the test fails? Furthermore, the set of subprograms and parameters which are really affected by this problem is a very small minority of all subprograms and parameters and it would be ugly to insert such address checks for all of them. This will have to be handled manually, possibly using post-translation.

- There is one exception, though, which may alleviate the problem for a large majority of the offending cases: the Oracle Tuxedo Application Runtime for CICS will ensure that all programs called from it (first program in a transaction, `EXEC CICS XCTL`, `EXEC CICS`

LINK, etc.) will receive a valid COMMAREA: either the one passed from the caller or a dummy-but-legal one.

- See also the discussion of the STICKY-LINKAGE compiler option below.

Representation of the NULL Pointer Value

The representation of the NULL pointer value may vary from one platform to another, in particular between the source and target platforms – if only because they don't have the same size, like every other pointer value. In consequence, every program which assumes a specific representation for this value, for instance by "casting" it to or from some binary integer value, may have a different behavior from one platform to another. The COBOL converter cannot handle this issue by itself, automatically, and it will have to be handled manually. Anyway, we strongly discourage such machine-dependent "hacks".

Description of the Input Components, Prerequisites

The input components are all the COBOL programs in the asset, after they have been parsed by the cataloger. In fact, the COBOL Converter loads the POB files for the programs, not their source files. In addition to the restrictions imposed by the cataloger (no nested programs, etc.; see [Cataloger](#)), the following rules must be respected before attempting the COBOL conversion:

- All the anomalies reported by the cataloger must be fixed. Otherwise, there is a risk that the conversion is incorrect, or even that the Converter fails (crashes). In fact, the Converter will refuse to convert any program that contains a FATAL error. But even ERRORS or simple WARNINGS may cause trouble, so it is strongly advised to fix all anomalies. See [force-translation Clause](#), however.
- The source format for all COBOL source files (main programs and copy files) must be fixed format with a numbering area (columns 1-6) and a comment area C (columns 73-80) physically removed. This must be done before cataloging. Note that information thus removed can be re-attached to the converted COBOL files, using the post-processor `AddComment`.
- The data migration process must have been run before COBOL conversion is started, because the latter depends on the former, for instance to decide which files will be migrated into relational database tables; see the Process Guide for more details. This dependency is concretized by the fact that the file migration tools generate some of the configuration files read by the COBOL converter.

Description of the Configuration Files

System Description File

The system description file describes the location, type and possible dependencies of all the source files in the asset to process. As such, it is the key by which the cataloger, but also all of the Rehosting Workbench tools, including the COBOL Converter, can access these source files and the corresponding components.

Note: Because of the need to have COBOL source files with the numbering area and comment area C removed, option `Cobol-left-margin` must be set to 1 (one) and option `Cobol-right-margin` must be set to 66; these are the default values.

Main Conversion Configuration File

This file is given to the COBOL converter using the `-c` or `-config` mandatory command-line option. It defines various "scalar" parameters influencing the conversion and points to subordinate files containing "large" configuration data, such as renaming files.

Note: Many of the parameters configurable in this file can also be set on the command line; in this case, the command-line value overrides the configuration-file value.

Tip: Although not mandatory, it is advisable to store this file in the same parameter directory as the system description file.

General Syntax

The contents of the main conversion configuration file is a free-format, unordered list of clauses, each beginning with a keyword and ending with a period. Some clauses take one or more arguments, others are boolean clauses with no argument. The keywords are case-insensitive symbols; the arguments are integers, symbols or (case-sensitive) strings. Spaces, new lines, etc., are comments. Comments can be written in the configuration file in two ways:

- Start with a sharp sign ("`#`") and extend to the end of the same line.
- Start with the "`/*`" delimiter and extend to the matching "`*/`" delimiter; in this form, comments can be spread over several lines and be nested.

target-dir Clause

Syntax

```
Target-dir : dir-path .
```

This clause specifies the location of the directory that will contain the complete hierarchy of target files, for both programs and copy files. If there is a source program `A/B/name.ext` in the root directory of the asset (as specified in the system description file), then the corresponding target program will be located as `A/B/name.ext` in this target directory (possibly with a different file extension, see below). The same mechanism is used for copy files, except that the target path will be `Master-copy/A/B/name.ext` (or possibly a different file extension). The Master-copy directory is related to the copy reconciliation process, see [Command-Line Syntax](#).

- The `dir-path` is given as a string. It can be either an absolute path or a relative path; in the latter case, it is relative to the directory containing the system description file, as usual for the Rehosting Workbench tools.
- The actual target directory will be created automatically, if necessary, when the COBOL Converter is run.

Sql-rules Clause

Syntax

```
Sql-rules : target-sql-syntax.
```

This clause specifies the target SQL syntax. Its value can be `oracle` or `none`. If the value is `none`, the sql code in the source files isn't translated. It's transferred as it is to the target components. The default value of this clause is `oracle`. In the latter case, it's not necessary to set `sql-rules` to `oracle` in the configuration file.

keep-same-file-names, target-program-extension and target-copy-extension Clauses

Syntax

```
keep-same-file-names.
```

```
target-program-extension : extension .(or) tpe : extension .
```

```
target-copy-extension : extension .(or) tce : extension .
```

These clauses direct how the file extensions for the converted programs (main source files) and copy files are determined:

- If the `keep-same-file-names` clause is given, the converted programs and copy files will have the same file extensions as the original files in the source asset (as cataloged). The other clauses, if given, will be ignored.
- Otherwise:
 - If the `target-program-extension` clause is given, then the converted programs will have the given file extension,
 - If the `target-copy-extension` clause is given, then the converted copy files will have the given file extension.
- By default, the converted programs will have the file extension `cbl` and the converted copy files will have file extension `cpy`.

Verbosity-Level Clause

Syntax

```
verbosity-level : level .
```

This clause specifies the amount of detail which the COBOL converter writes to the execution log. The default value of 2 is fairly verbose, higher values are even more verbose, value 1 only displays important (error) messages.

deferred-copy-reconcil Clause

Syntax

```
deferred-copy-reconcil.
```

or

```
deferred-crp.
```

or

```
dcrp.
```

This clause specifies that the copy-reconciliation process `crp` is to be deferred until after the conversion is completed; this allows COBOL conversion to run in multiple concurrent processes. By default, in the absence of this clause, the copy-reconciliation process is executed incrementally immediately after each program is converted, which mandates single-process execution. See the copy-reconciliation process below for more details.

force-translation Clause

Syntax

```
force-translation.
```

This clause directs the COBOL converter to (try to) convert even those programs that contain FATAL errors although without any guarantees: the converter may produce incorrect results or even crash. By default, in this case, the converter refuses to work on this program and skips to the next one.

rename-copy-map-file Clause

Syntax

```
rename-copy-map-file : file-path .
```

This clause specifies the location of the subordinate configuration file containing information to rename copy files, see the [copy-renaming Configuration File](#) below. The file path is given as a string. It can be either an absolute path or a relative path; in the latter case, it is relative to the directory containing the system description file, as usual for the Rehosting Workbench tools.

rename-call-map-file Clause

Syntax

```
rename-call-map-file : file-path .
```

This clause specifies the location of the subordinate configuration file containing information to rename sub-programs and their calls, see the [Call-Renaming Configuration File](#). The file path is given as a string. It can be either an absolute path or a relative path; in the latter case, it is relative to the directory containing the system description file, as usual for the Rehosting Workbench tools.

post-translation-file Clause

Syntax

```
post-translation-file : file-path .
```

This clause specifies the location of the subordinate configuration file containing the description of manual transformations to apply after the Rehosting Workbench Converter, see the [Post-Translation Configuration File](#). The file path is given as a string. It can be either an absolute

path or a relative path; in the latter case, it is relative to the directory containing the system description file, as usual for the Rehosting Workbench tools.

on-size-error-call Clause

Syntax

```
on-size-error-call: proc-name .
```

This clause specifies the name, as a symbol, of the procedure to call to cause a definite termination of the program. This name is used to force termination in situations in which the IBM compiler would force termination but not the target compiler, such as size errors in arithmetic statements. The default name is `.ABORT`.

hexa-map-file Clause

Syntax

```
hexa-map-file : file-path .
```

This clause specifies the location of the subordinate configuration file containing the EBCDIC-to-ASCII transformation to apply to characters in hexadecimal form, see the [Hexadecimal Conversion Configuration File](#). The file path is given as a string. It can be either an absolute path or a relative path; in the latter case, it is relative to the directory containing the system description file, as usual for the Rehosting Workbench tools.

conv-ctrl-file Clause and alt-key-file Clause

These two clauses go together.

Syntax

```
conv-ctrl-file : file-path or conv-ctrl-list-file : file-path .  
alt-key-file : file-path
```

These clauses specify the location of the two subordinate configuration files containing information regarding file-to-Oracle conversion. These files are generated by the Rehosting Workbench File-to-Oracle conversion tool, as respectively the `Conv-ctrl-file` or the `Conv-ctrl-list-file` and the `Alt-key` file. See [File-to-RDBMS Configuration Files](#).

Only one of the first two clauses must be given: either the `conv-ctrl-file` clause or the `conv-ctrl-list-file` clause, but not both.

The file path is given as a string. It can be either an absolute path or a relative path; in the latter case, it is relative to the directory containing the system description file, as usual for the Rehosting Workbench tools.

RDBMS-conversion-file Clause

Syntax

```
RDBMS-conversion-file : file-path .
```

This clause specifies the location of the top-level subordinate configuration file containing information about relational DBMS conversion (from DB2 to Oracle). See the [RDBMS-conversion Configuration Files](#) for more details. The file path is given as a string. It can be either an absolute path or a relative path; in the latter case, it is relative to the directory containing the system description file, as usual for the Rehosting Workbench tools.

keywords-file Clause

Syntax

```
keywords-file : file-path .
```

This clause specifies the location of the subordinate configuration file containing information to rename COBOL identifiers which happen to be keywords or reserved words in the target COBOL dialect, see the [keywords File](#) for more details. The file path is given as a string. It can be either an absolute path or a relative path; in the latter case, it is relative to the directory containing the system description file, as usual for the Rehosting Workbench tools.

accept-date and accept-day Clauses

Syntax

```
accept-date: date-entry-name .
```

```
accept-day: day-entry-name .
```

These clauses specify sub-program names to replace ACCEPT ... FROM DATE and ACCEPT ... FROM DAY statements. For instance, the statement:

```
ACCEPT MY-DATE FROM DATE
```

would be replaced by:

```
CALL "DATE-ENTRY-NAME" USING MY-DATE BY VALUE LENGTH OF MY-DATE
```

This allows more control and more flexibility on how programs acquire their current date. For instance, during regression tests, it is necessary to run migrated programs with the same current date as when the source programs were run; these sub-programs (to be supplied by the Rehosting Workbench users, according to their requirements) will allow this.

If any of these clauses is not specified, the corresponding statements are not transformed. You can use the target COBOL parameters (for example `current_day`, `current_month`, and `current_year` parameters of the Microfocus COBOL run-time system) to control the date returned by the `ACCEPT` statements; see the MicroFocus/COBOL-IT documentation.

sql-stored-procedures-file Clause

Syntax

```
sql-stored-procedures-file: file-path .
```

This clause specifies the location of the subordinate configuration file containing the list of DB2 stored procedures called directly from COBOL, see the [stored-procedure File](#) for more details. The file path is given as a string. It can be either an absolute path or a relative path; in the latter case, it is relative to the directory containing the system description file, as usual for the Rehosting Workbench tools.

remove-sql-qualifier Clause

Syntax

```
remove-sql-qualifier .
```

This clause enables the transformation rule which removes the schema qualifier from every SQL identifier which has one. The resulting program will hence rely on implicit schema qualification.

Tip: This is generally not needed, and possibly even not desired, but it is useful if you want to run the program concurrently in multiple environments (connecting to multiple databases or schemas), for instance in multiple test corridors.

activate-cics-rules Clause

Syntax

```
activate-cics-rules .
```

This clause forces the COBOL converter to apply to any program processed in the current execution the rules which normalize the EXEC CICS statements and prepare the program for use with the Oracle Tuxedo Application Runtime for CICS environment, including the CICS preprocessor.

Notes:

- There exists a command-line option of the same name (see [cobol-convert Command](#)) which has the same effect as this clause, and which is more flexible to use. So we believe that the configuration-file clause will be seldom used, except perhaps in projects in which the TP and batch parts of the asset are well identified and strictly separated in the migration project.
- Whether this clause is given or not, the above rules will be applied anyway to every program which contains one or more EXEC CICS statement. So this clause (or the equivalent command-line argument) will be effective only for subprograms used in a CICS environment (implicit COMMAREA, etc). but do not perform CICS operations themselves.

pure-seq-map-file Clause

Syntax

```
pure-seq-map-file: file-path .
```

or

```
purely-sequential-map-file: file-path .
```

This clause specifies the location of the subordinate configuration file containing the list of SEQUENTIAL logical files which are to be kept (record) SEQUENTIAL rather than converted to LINE SEQUENTIAL. See [purely-sequential Configuration File](#) for more details. The file-path is given as a string. It can be either an absolute path or a relative path; in the latter case, it is relative to the directory containing the system description file, as usual for the Rehosting Workbench tools.

dont-print-what-string Clause

Syntax

```
dont-print-what-string .
```

When present, this clause specifies that the what-string containing conversion timestamp and converter version information, which the converter normally inserts at the beginning of every

converted file, is not to be printed out in this execution. This will be seldom used, unless you really want to hide the fact that your application is migrated using the Rehosting Workbench!

remove-empty-copies Clause

Syntax

```
remove-empty-copies .or rec.
```

When present, this clause specifies that COPY directives referencing copy files which no longer contain useful COBOL code after conversion are to be commented out; by default, these directives remain active. This applies for instance to copy files defining whole FD paragraphs for files which migrate into database tables.

sql-return-codes-file Clause

Syntax

```
sql-return-codes-file: file-path .
```

This clause specifies the location of the subordinate configuration file containing additional pairs of equivalent DB2 & Oracle SQLCODE values. See the [sql-return-codes Configuration File](#) for more details. The file-path is given as a string. It can be either an absolute path or a relative path; in the latter case, it is relative to the directory containing the system description file, as usual for the Rehosting Workbench tools.

copy-renaming Configuration File

This file is associated with the [rename-copy-map-file Clause](#). Its contents are in CSV format, with the semicolon character used as separator. Each line is in the form:

```
original-copy-name;original-library-name;new-copy-name.
```

All names are COBOL-like, case-insensitive symbols. The meaning of such a line is that, when the directive:

```
COPY ORIGINAL-COPY-NAME OF ORIGINAL-LIBRARY-NAME { REPLACING ... }
```

is encountered in a program, it is replaced by:

```
COPY NEW-COPY-NAME { REPLACING ... }
```

Note that library names are not used on the target platform because they are inconvenient; it is much better to use search paths, see the COBCPY environment variable). When the original-library-name field is empty, the rule is to replace unqualified directives of the form:

```
COPY ORIGINAL-COPY-NAME { REPLACING ... }
```

The same renaming applies to the copy file itself: when the Converter prints out, in the target private copy directory, the copy file referenced by this directive (see below for more information about copy reconciliation), it is printed with the new name.

When the [rename-copy-map-file Clause](#) is not present, or when this file is empty, no copy renaming takes place. It is an error when the file cannot be found or read, or when the same `original-copy-name;original-library-name` combination is associated with different `new-copy-names` in different lines of the file. In this case, the converter stops with an error message and does not convert any programs. Note however that it does not check whether two different copy files in the same directory are renamed to the same target file. In principle, this would be handled gracefully by the copy reconciliation process, but without guarantee.

Call-Renaming Configuration File

This file is associated with the [rename-call-map-file Clause](#) described above. Its contents are in CSV format, with the semicolon character as separator. Each line is in the form:

```
original-call-name;new-call-name.
```

All names are COBOL-like, case-insensitive symbols. The meaning of such a line is that, when the statement:

```
CALL "ORIGINAL-CALL-NAME" { USING ... }
```

is encountered in a program, it is replaced by:

```
CALL "NEW-CALL-NAME" { USING ... }
```

The converter also attempts to rename literal strings which are "associated" with variables used in dynamic calls using direct constructs (VALUE, MOVE, etc.). For obvious reasons, it cannot handle truly dynamic calls in which the callee name is "computed" using complex manipulations (STRING, etc.), transported thru opaque containers or obtained from outside the caller program (e.g., read from a file or passed as parameter); such situations must be handled manually.

The same renaming applies to called sub-programs and their entry points: when the converter prints out in the target directory, a program whose base name matches one of the original names listed in the renaming file, it is printed with the corresponding new name. Similarly, for ENTRY statements whose argument is a string matching one of the original names, the string is transformed into the new name.

When the [rename-call-map-file Clause](#) clause is not present, or when this file is empty, no call renaming takes place. It is an error when the file cannot be found or read, or when the same `original-call-name` is associated with different `new-call-names` in different lines of the file. In this

case, the converter stops with an error message and does not convert any programs. Note however that it does not check whether two different subprograms in the same directory are renamed to the same target file.

Post-Translation Configuration File

This file is associated with the [post-translation-file Clause](#). Its contents are a sequence of rules with the following syntax:

```
rule rule_name
filter [
(+|-)program_name_regexp
]
transform [
    source_lines_block
]
into [
    target_lines_block
]
```

The semantics of such a rule are simple: if, in a program, the (base) name of which matches any of the "positive" `program_name_regexp`'s but none of the "negative" ones, a block of lines matching `source_lines_block`¹ is encountered, it is replaced by `target_lines_block`. `rule_name` is used in the comment associated with the application of the transformation. See appendix the post-translator below for more details.

The post-translation file may contain as many rules as desired, in any order (although the behavior of the post-translator is not defined when two `source_lines_blocks` overlap, or when a `source_lines_block` and a `target_lines_block` overlap).

Tip: In the syntax above, it is very important that the square brackets closing the filter, transform, and into clauses, are in column 1, at the very beginning of the line; otherwise, they will be interpreted as part of the block.

1. In this context, matching simply means that the two blocks of lines must be identical when you reduce each sequence of spaces in both of them to a single space. This is basically "identical" with a little flexibility added.

Comments start with a sharp sign ("#") and extend to the end of the line; you can insert them anywhere between the rules, between the four clauses in a rule and after the rule name; if you insert such a comment inside a square-bracketed filter or transform or into block, it will be considered as part of the block contents rather than as a comment.

Hexadecimal Conversion Configuration File

This file is associated with the [hexa-map-file Clause](#) above. Its contents are an EBCDIC-to-ASCII conversion table to apply to characters in hexadecimal form (characters in textual form are supposed to be converted at the same time as the source file itself). The syntax is simply a CSV file with a semicolon as separator. Each line is in the form:

```
source-hexa-code;target-hexa-code,
```

Each hexa-code being written as usual, with two hexadecimal characters (0-9, A-F). The semantics of this conversion table are that if some hexadecimal literal in the source file does not match any source code in this table, it is left as is, unconverted. Such conversion works also on embedded-SQL code. Note that the converter makes no attempt to check the intrinsic consistency of the conversion table (e.g. the fact that no source-hexa-code or no target-hexa-code appears twice), nor the fact that it really describes some EBCDIC-to-ASCII conversion.

Tip: It is strongly suggested that this table be derived from the global, project-specific conversion table used to convert data and source files, see Oracle Tuxedo Application Runtime Process Guide for more details. Failure to do so may lead to differences of behavior on the target platform, for which we take no responsibility.

How to Generate the hexa-map File

Oracle Tuxedo Application Rehosting Workbench makes available a script which generates the hexa-map file based on the CONVERTMW copy file, see Using the COBOL CONVERTMW.cpy file in [Codeset Conversion](#) chapter.

The script generating the hexa-map file is located in the directory:

```
REFINEDIR/scripts/
```

The script names is:

```
convert-hexa-copy-to-map.sh
```

Syntax

```
REFINEDIR/scripts/convert-hexa-copy-to-map.sh convertmw_copy_file
```

`convertmw_copy_file`: location of the `CONVERTMW.cpy` file

The script automatically generates the `tr-hexa.map` file inside the current directory (`PARAM` directory is a better choice). This generated file name has to be used as `file-path` value with the `hexa-map-file` attribute.

On normal end, a return code of 0 is returned. Otherwise, see displayed messages and content of `tr-hexa.map` file.

Error Messages

WBART-1001:

Example: COPY file <filename> not found. Check argument 1.

Explanation: Argument 1 must contain the CONVERTMW COBOL copy file name.

WBART-1003:

Example: bad status returned by awk

Explanation: see messages written into `tr-hexa.map` file

Messages could be :

too many FILLER in TRANSCODE-[SOURCE | CIBLE]

The filler number *id* does not contains enough hexa element: *num* instead of 64

not enough FILLER in TRANSCODE-SOURCE and/or TRANSCODE-CIBLE

File-to-RDBMS Configuration Files

These files are associated with the [conv-ctrl-file Clause](#) and [alt-key-file Clause](#). They contain information about file-to-RDBMS conversion, e.g. to define which logical files (FDs) are converted into RDBMS tables (actually, because the physical files they are associated with are converted to these DB tables). Since these files are automatically generated by the Rehosting Workbench File-to-Oracle conversion tool and should not be modified by hand, their contents are not further specified here.

RDBMS-conversion Configuration Files

These files are associated with the [RDBMS-conversion-file Clause](#) above. The information they contain is accessed in a two-level way:

- The top-level file is named in the [RDBMS-conversion-file Clause](#) proper. Its contents is a CSV table, with each line in the form:

```
schema-name;file-path.
```

File-path is the path to the file containing RDBMS-conversion information pertaining to SQL schema schema-name. As usual, it can be either an absolute path or a relative path; in the latter case, it is relative to the directory containing the system description file. This file must be created by the Rehosting Workbench user.

- For each schema in the application, the file containing RDBMS-conversion information pertaining to this schema (default date and time format, renaming map, etc.) is an XML file generated by the File-to-Oracle tool, which should not be modified by hand. In consequence, its content is not further specified here.

keywords File

This file is associated with the [keywords-file Clause](#). Its contents are a CSV table using the semicolon as separator, each line being in the form:

```
old-name;new-name.
```

The effect of such a line is to rename every COBOL identifier (variable name, paragraph name, etc.) named old-name in every program into new-name. This is required for names which happen to be keywords or reserved words in the target COBOL dialect, such as TEST, but it may also be useful to rename plain identifiers for reengineering purposes.

stored-procedure File

This file is associated with the [sql-stored-procedures-file Clause](#). Its contents are a list of subprogram names, one per line. When one of these names appears in a COBOL CALL statement, the latter is replaced by an SQL CALL statement. In addition, declarations of the parameters of the CALL, if any, are adapted so that they can be used in SQL statements.

purely-sequential Configuration File

This file is associated with the [pure-seq-map-file Clause](#). Its contents is a CSV table using the semi-colon as separator, each line being in the form

```
program-name;FD-name
```

with both names being symbols. The effect of such a line is to prevent this particular logical file (the given FD in the given program), assumed to be (record) SEQUENTIAL on the source platform, to be converted to LINE SEQUENTIAL on the target platform; rather, it is kept unchanged as a

record `SEQUENTIAL` file. This makes it much less amenable to manipulation using standard target-platform utilities, but on the other hand, it will support unrestricted `REWRITE` operations (see section `REWRITE` operations on `LINE SEQUENTIAL` files above). This might also be useful for files exchanged with a z/OS platform in binary form.

sql-return-codes Configuration File

This file is associated with the [sql-return-codes-file Clause](#). Its contents is a CSV table using the semicolon as separator, each line being in the form:

```
DB2-sqlcode-value;Oracle-sqlcode-value
```

with both values being positive or negative integers. The effect of such a line is to add the pair of values to the translation table used to map "remarkable" DB2 `SQLCODE` values to their equivalent Oracle `SQLCODE` values. This translation table is initialized as if read from the following file:

Listing 10-2 DB2 to Oracle SQL Return Code Mapping

```
+100;+1403
-810;-1422
-803;-1
-530;-2291
-516;-1002
-501;-1001
-407;-1451
-305;-1405
-180;-1820
-181;-1821
-811;-2112
-204;-942
```

Of course, value 0 (zero) is mapped to itself.

Note: The COBOL converter does not currently check the consistency of this translation table; for instance, it does not complain if the same DB2 value is mapped to more than one distinct Oracle value.

Description of Output Files

Converted Programs and Copy Files

Naming Scheme

As mentioned above, the main purpose of the Rehosting Workbench COBOL Converter is to produce the converted COBOL components, in the form of their source files. There is a direct, one-to-one correspondence between the hierarchy of main program files inside the source root directory and the hierarchy of main program files inside the target root directory; the only possible differences, as far as file names are concerned, come from the CALL-renaming map and the choice of the target program-file extension, see [rename-call-map-file Clause](#) and [keep-same-file-names, target-program-extension and target-copy-extension Clauses](#). The same comments apply for the target copy files, with the following observations:

- The hierarchy of target copy files is located in the `Master-copy` sub-directory of the target root directory.
- The names of the target copy files may differ from those of the source files because of the `COPY-renaming` map and the choice of the target copy-file extension.
- The correspondence between source and target copy files may not be strictly one-to-one. It may be one-to-many when the transformations applied by the converter on the contents of some copy file depend on the context in which this file is included. This is handled by the copy reconciliation process, see below for more details.

If file `ORIGCOPY(.s-ext)` is translated into multiple versions, these versions are named `ORIGCOPY(.t-ext)`, `ORIGCOPY_V1(.t-ext)`, `ORIGCOPY_V2(.t-ext)`, etc.

Transformation Comments

In principle, COBOL conversion is a "light" process, because COBOL on the target platform is not that different from COBOL on the source platform. This is why this process is called conversion rather than translation. Indeed, a converted file, either main file or copy file, generally differs from its corresponding source file only in very few places; the bulk of the contents is not affected in any way and is reproduced in the target file exactly as it is in the source file. The differing places, however, are identified by specific transformation comments.

Modified Code

In places at which some transformation actually took place, the converter inserts transformation comments describing the effects of the transformation. The affected code is composed of:

- A header line giving the transformation-rule name and version; the header line starts with a recognizable prefix, namely "`*{`", in which the opening curly bracket symbolizes the start of the transformation.
- The original code, commented out.
- An intermediate separator, namely a line composed of "`*--`".
- The new code which replaces the original one.
- A terminating line, namely "`*}`", in which the closing curly bracket symbolizes the end of the transformation.

Listing 10-3 Transformation Comment Example

```
*{ tr-binary-to-comp-5 1.2
* 77 MY-VAR PIC S9(9) COMP.
*--
    77 MY-VAR PIC S9(9) COMP-5.
*}
```

Added Code

Some rules not only transform existing code but also insert some completely new code in some remote places, for instance, the declaration of an intermediate variable in the Working-Storage Section. In this case, the affected area in the program is composed of:

- A header line giving the transformation-rule name and version; the header line starts with the prefix "`*+{`", in which the opening curly bracket symbolizes the start of the transformation and the plus sign indicates that this is an insertion rather than a transformation.
- The inserted code.

- The terminating line `"*+}"`.

Deleted Code

When a rule simply deletes some code rather than transforming it, the affected area in the program has the same organization as for modified code, except that the "new code" area is empty:

Moved Code

Some rules move code from one place in the program to another, for instance, when a file is migrated into a relational DB table, the corresponding FD is deleted and the data records it contains are moved to the Working-Storage Section. In this case, the code at the original location is shown as deleted and the code at the new location is shown as inserted.

Other Comment Rules

- It is possible that some transformation acts on the code produced by a previous transformation. In this case, the transformation comments are properly nested.
- The format of the transformation comments is designed both:
 - To be informative and help the program maintenance team to understand the transformations which were applied during migration by the COBOL Converter; studying these transformations is a quick way for the developer to understand the differences between the source and target COBOL dialects, and become proficient with the latter.
 - To be automatically deletable when they cease to be useful and become a nuisance.

Layout

When the COBOL Converter applies a transformation rule to a piece of code, it attempts to keep the same layout for the new code, by minimizing how elements of the code which exist in both the original and new versions are moved around. In addition, when the converter inserts a new element, for instance a statement or a variable declaration, it tries to align the new element with similar ones before or after it. When, by following these guidelines, a transformed or new line of code becomes too long for the fixed format, the converter cuts the line at the right-most "nice" cutting point (preferably between two words) and wraps the rest on the next line, flush with the right margin, to indicate that these wrapped elements are logically part of the previous line.

- We recognize however that the layout issue is a subjective matter and that the result of a transformation might not appear exactly as you would have made it yourself. Consider though that:

- The only obligation of the Converter is to produce code which is correct; aesthetic considerations are not part of the contract.
- The converter only applies rules which have to be deterministic; it does not have the power to estimate the aesthetic value of this or that layout.
- Some other people might actually be completely satisfied with the resulting layout!

Miscellaneous Issues

When the Converter prints out a copy file invoked in the main program file with a REPLACING clause, it takes great care to undo the effect of the replacements. However, when the transformations performed by the converter apply to pieces of text generated by some replacement, it may be very hard for the converter to compute the "inverse" of the transformation and create the transformed replacement clause – for instance, when the COPY clause replaces "something" by "nothing", the converter may have a hard time finding the "nothing" to replace back by "something" when the area is affected by a transformation. In such cases, some manual correction might be necessary; use the post-translation feature to apply it in a repeatable way.

Compiler Options

To guarantee identical behavior between the source programs and the target ones produced by the COBOL converter, up to the limitations described above, the target programs must be compiled with a certain set of compiler options in effect. Indeed, some of the MicroFocus COBOL compiler options do change the behavior of the executed code. The transformations applied by the Rehosting Workbench COBOL converter are hence tailored to the option set described below. No support is provided for programs compiled with a different, or at least conflicting, option set. For more information, please see the Micro Focus COBOL documentation, in particular the Compiler Directives book and the COBOL-IT Compiler Suite Enterprise Edition - Reference Manual.

MicroFocus

Mandatory Options

The mandatory compiler options are listed below. For each of them, we indicate whether it is set by default, we give a brief description and we justify why we require it.

DIALECT"MF" (default)

Sets the most “native” and efficient mode of operation. Since the aim of the Rehosting Workbench is not simply to emulate the source mainframe, but to forget about it and lead you towards the benefits of the target platform, this is the best choice. It will enable you

to use the modern features of MF COBOL such as Unicode support and object-oriented programming.

CHARSET"ASCII" (default)

Sets the default character set and collating sequence to those supported natively on the target platform. This is an obvious choice, too.

SOURCEFORMAT"FIXED" (default)

Directs the compiler to stick with the “old” standard, fixed-format, column-based format. This may appear contradictory with our aim of modernity, but unfortunately the Oracle Pro*COBOL compiler, even the latest 11g version, is still not quite compatible with the MicroFocus free format, and we have to require fixed format to guarantee correct behavior.

ALIGN"8" (default)

Defines the alignment for top-level structures (01 and 77-level). Required to make sure that structures retain the same layout as on the source platform, and yields the best performance, at a slight expense in memory size.

COMP5-BYTE-ORDER"NATIVE" (default)

Uses native byte ordering for COMP-5 variables. Necessary for compatibility with the C language and the Oracle Tuxedo Application Runtime.

P64 (to set explicitly, except on MicroFocus installations set up to compile to 64-bit mode by default)

Compiles for 64-bit platforms. All the target platforms supported by the Rehosting Workbench are 64-bit.

SIGN"EBCDIC" (to set explicitly)

Uses the EBCDIC convention rather than the ASCII convention for representing overpunched sign on DISPLAY numeric values. This is the same convention as on the source platform and hence provides for less differences of behavior when the last digit of such a numeric value is redefined by a character variable.

DEFAULTBYTE"00" (to set explicitly, except if the previous option is given)

Specifies the value with which to initialize all otherwise-undefined variables in the Working-Storage Section. Not strictly necessary, since on the source platform, the Working-Storage Section is officially not implicitly initialized, but we found that it leads to less differences of behavior when a numeric variable is redefined by a character variable.

RWHARDPAGE (to set explicitly)

Causes the Report Writer control module to execute a form feed after the last item has been printed on a page, instead of the usual multiple blank lines. All Unix printer systems correctly handle Form Feed characters.

INDD or INDD"SYSIN80L" (to set explicitly)

Causes “default” ACCEPT statements to read from the specified logical file instead of from the Unix standard input file. This is the same behavior as on the source platform and is appropriate with ART-translated KSH scripts, which treats SYSIN as any other file for COBOL programs.

OUTDD or OUTDD"SYSOUT80L" (to set explicitly)

Causes “default” DISPLAY statements to write to the specified logical file instead of to the Unix standard output file. This is the same behavior as on the source platform and is appropriate with the Rehosting Workbench-translated KSH scripts, which treat SYSOUT as any other file for COBOL programs.

**HOSTARITHMETIC, HOST-NUMMOVE"2", HOST-NUMCOMPARE"2",
ARITHMETIC"ENTCOBOL", CHECKDIV"ENTCOBOL",
FP-ROUNDING"ENTCOBOL", REMAINDER"2" (to set explicitly)**

All these options control various aspects of the treatment of numeric variables and expressions. Their settings maximizes the compatibility with the source platform.

IBMCOMP (to set explicitly)

Turns on word-storage mode for the layout of the structures, the same mode as on the source platform. It also enables the SYNC[HRONIZED] clause to have an effect for “machine-native” types (binary integers, binary floats, pointers, etc.), so as to yield the most efficient performance, at a slight increase in memory consumption.

ODOSLIDE (to set explicitly)

Moves data items that follow a variable-length table in the same record as the table's length changes. This affects data items that appear after a variable-length table in the same record; that is, after an item with an OCCURS DEPENDING clause, but not subordinate to it. With ODOSLIDE, these items always immediately follow the table, whatever its current size; this means their addresses change as the table's size changes. With NOODOSLIDE, these items have fixed addresses, and begin after the end of the space allocated for the table at its maximum length. Setting ODOSLIDE leads to the same behavior as on the source platform.

PERFORM-TYPE"ENTCOBOL" (to set explicitly)

Enables the same behavior as on the source platform regarding nested PERFORM statements with overlapping ranges. The default option PERFORM-TYPE"MF" is semantically cleaner and allows more efficient execution, but may lead to differences of

behavior which are hard to detect and diagnose; hence, unless you know that your `PERFORM` ranges are “well-behaved” and never overlap, we can’t recommend the default setting.

RDW (to set explicitly)

Enables you to find out the length of a record that has just been read from a variable-length sequential file, by providing a “hidden” length variable just before the first record of the `FD` (see more details in the MicroFocus documentation). This “trick” is available on the source platform, although not explicitly advertised, and this option allows to reproduce the same behavior.

RECMODE"ENTCOBOL" (to set explicitly)

Directs the MicroFocus compiler to use the same algorithm as the source compiler to determine whether a file has fixed-length or variable-length format, depending on the length of the various records in the file definition.

ASSIGN"EXTERNAL" (to set explicitly)

Directs the MicroFocus compiler to use, by default, the `EXTERNAL` file-assignment method. In this method, the `SELECT` names are used as keys to search the actual file names in environment variables of the form `DD_NAME`. This is the mode chosen for the Rehosting Workbench, because it allows the file assignments to be specified outside the programs, namely in the calling KSH scripts. Not only is this closer in philosophy to the source behavior, but in our opinion this is the most flexible method.

SYSPUNCH"80" (to set explicitly)

Defines the record length for the `SYSPUNCH` logical file. Default setting (132) is not the same as on the source platform.

ZEROLENGTHFALSE (to set explicitly)

When `ZEROLENGTHFALSE` is set, all comparisons between zero-length group items, and between zero-length items and figurative constants, return false; when it is not set, they all return true. To reproduce the same behavior as on the source platform, it must be set.

NOADV (default)

Do not use special printer-control characters on text files. Target-platform printing utilities will simply print a file with the same layout as it appears on the screen.

NOTRUNCCALLNAME (default)

Does not truncate names of subprograms referenced in `CALL` statements. This is necessary for the Rehosting Workbench-migrated assets, because data access routines generated by the Rehosting Workbench have long names. In addition, in future evolutions of the asset, you will want to get rid of the short-names limitations imposed on the [source platform](#).

NOTRUNCCOPY (default)

Does not truncate names of copy files referenced in COPY directives. This is necessary for the Rehosting Workbench-migrated assets, because copy files generated or inserted by the Rehosting Workbench have long names. In addition, in future evolutions of the asset, you will want to get rid of the short-names limitations imposed on the source platform.

NOCOPYLBR (default)

Treat copy-file names as plain paths, not library archives (.lbr files). This is necessary for the Rehosting Workbench-migrated assets, because copy files converted or generated by the Rehosting Workbench are not grouped in archives.

NOSPZERO / NOSIGN-FIXUP (default)

NOSIGN-FIXUP provides emulation of the mainframe compiler option NUMPROC(PFD) when using the HOST-NUMCOMPARE and HOST-NUMMOVE directives. This option gives the best performance, given that it is not possible to provide a complete emulation of NUMPROC(NOPFD) behavior.

REPORT-LINE"256" (default)

Specifies the maximum length of a Report Writer line.

COPYEXT"cpy,cbl" (to set explicitly)

Specifies the filename extension of the copyfile that the compiler is to look for if a filename in a COPY statement is specified without an extension. This non-default setting is appropriate for AST-migrated asset, because copy files generated by the Rehosting Workbench always have the .cpy extension and copy files converted by the Rehosting Workbench generally have the same extension. However, if you configure the COBOL converter for another extension, you will have to adapt the setting of this option appropriately.

After taking into account default and explicit options, and dependencies, the required option list must start with the following:

Listing 10-4 Validated COBOL Compiler Option List

```
P64 SIGN"EBCDIC" RWHARDPAGE INDD OUTDD HOSTARITHMETIC HOST-NUMMOVE"2"
HOST-NUMCOMPARE"2" ARITHMETIC"ENTCOBOL" CHECKDIV"ENTCOBOL"
FP-ROUNDING"ENTCOBOL" IBMCOMP ODOSLIDE PERFORM-TYPE"ENTCOBOL" RDW
RECMODE"ENTCOBOL" REMAINDER"2" ASSIGN"EXTERNAL" SYSPUNCH"80"
ZEROLENGTHFALSE COPYEXT"cpy,cbl"
```

No guarantee will be given for programs compiled with an option list which contradicts the above one. The current version of Oracle Tuxedo Application Rehosting Workbench and Oracle Tuxedo Application Runtime for CICS, have been validated with this option list.

Note: Mandatory Oracle SQL compilation options for the Pro*COBOL preprocessor are listed in the *ART Workbench RDBMS Converter Reference Guide*.

Note: The P64 option is not necessary on Micro Focus installations set up to compile to 64-bit mode by default.

Installation-dependent Options

These options are not strictly necessary but may help you handle assets in which programs contain a mixture of upper-case and lower-case letters:

FOLDCALLNAME"UPPER" (to set explicitly)

Directs the compiler to map subprogram names in `CALL` statements to upper case.

FOLDCOPYNAME"UPPER" (to set explicitly)

Directs the compiler to map copy file names in `COPY` directives to upper case.

MAPNAME (to set explicitly)

Makes the Compiler alter program-names and entry-point names to make them compatible with the source platform.

By experimenting with these settings, you may find the combination which is appropriate for your particular asset. For instance, `FOLDCALLNAME"UPPER"` and `MAPNAME` taken together provide a good enough emulation of the `PGMNAME (COMPAT)` source-compiler option, but there is no sure way to emulate the other values of this option.

1.1.1.3 Options Depending on Customer Choice

The following options influence the behavior of the target asset, but may be set more or less at will by the user of the ART system.

BOUND and SSRANGE

checks that each index is between the correct bounds when accessing an array or in reference modifiers. This is similar to the `SSRANGE` option of the IBM compiler. We strongly recommend that both of these options be set, at least during migration tests and in the first few months of operation (note that setting `SSRANGE` also sets `BOUND`). This choice is a bit controversial because it can break some programs which apparently run correctly on the source platform (without the `SSRANGE` option). However, in our experience, the only programs which break are incorrect programs which just happen to work by chance on the source platform and would not work in the same way, or at all, on

the target platform. The sooner we detect these programs and fix them, the better. In the same way, you could consider setting the `CHECK` option, which enables various (other) kinds of run-time checks and allows to detect other kinds of seemingly-correct programs.

TRUNC

specifies whether truncation to the given `PIC` size occurs when assigning a value to a `BINARY` variable (or `COMP`, or `COMP-4`). This is similar to the `TRUNC(STD)` option of the IBM compiler. However, with the present specification of the ART COBOL converter, all such variables are transformed into `COMP-5` variables, which do not obey the `TRUNC` option. See the discussion in *Use of COMP-5 type and the TRUNC compiler option* above.

APOST and QUOTE

allow to choose which character, single or double quote, the `QUOTE` symbolic constant will represent. This is similar to the IBM options of the same name. Use the same setting as on the source platform.

NOALTER

forbids the presence of `ALTER` statements in the COBOL programs. Since `ALTER` statements are a thing of the past, and a very bad thing if any, we recommend that you take the opportunity of migrating your asset with the ART Workbench to chase out any remaining `ALTER`. Then, set this option to prevent their reappearance and to make compiled code more efficient and safe.

AREACHECK

Causes the Compiler to treat any token which starts in area A in the Procedure Division as a paragraph or section label, regardless of the preceding tokens. If `AREACHECK` is not specified, only tokens which follow a period are treated as possible labels. This directive provides closer compatibility with IBM error handling, where omitting a period before the label produces a less serious message. We recommend that such erroneous source code is corrected.

NOBYTEMODEMOVE

Controls behavior for alphanumeric moves between overlapping data items. If `BYTE-MODE-MOVE` is specified, data is moved one byte at a time from the source to the target. If `NOBYTE-MODE-MOVE` is specified, the data is moved in granules of two, four or more bytes at a time (depending on environment) from the source to the target. Consequently, if the overlap is less than the size of the granule, each granule moved overwrites part of the next granule to be moved. `NO-BYTE-MODE-MOVE` gives better performance, but may yield incorrect code on some very rare programs which work correctly on the source platform; we suggest that you start with the “more compatible” setting (`BYTE-MODE-MOVE`), perform complete regression tests until satisfaction, then choose the other option and re-test.

DYNAM

Specifies that `CANCEL` statements are not to be ignored. This is similar to the IBM option of the same name (but not quite the same, see the MicroFocus documentation). We strongly recommend that you set this option, because the Tuxedo servers in the ART TP Run-time system, which execute the applicative CICS programs, use `CANCEL` statements to free the memory used to load and run those programs. If `NODYNAM` is in effect, the amount of memory use by these servers would grow as they execute more and more different programs.

NOFDCLEAR, NOHOSTFD

The “positive” settings of these options reproduce the restrictions on `FD` usage imposed by the IBM compiler (`FD` records allocated only at `OPEN` time, record contents lost after `WRITE`, etc.). We feel that these restrictions are silly and hence recommend that you don’t use these options.

NATIVEFLOATINGPOINT

see the discussion in *Use of floating-point variables* above.

NOSEG

turns off segmentation and ignores all segment numbers. The resulting program is a single piece with no overlay. Who still uses segmentation, anyway?

STICKY-LINKAGE"2" / NOSTICKY-LINKAGE

this option controls how a program parameter (Linkage Section item) which has been linked to some actual data item in a previous invocation of the program may be re-linked with the same item if the current invocation specifies no new linking (no actual argument supplied). The `STICKY-LINKAGE"2"` setting is “more compatible” with the behavior of the source platform, especially for CICS programs, but it is certainly non-standard and error-prone. It may also be incompatible with certain features of the ART TP run-time system, in particular the possibility to distribute TP transactions over several servers running in a cluster with no shared memory. So we strongly suggest to use the default `NOSTICKY-LINKAGE` setting from the beginning and fix any sticky-linkage-related bug discovered during regression testing. See also the discussion in Linkage-section arguments with `NULL` address above.

1.1.1.4 Options Influencing Compile-Time Operation

The following options influence only the production of the compilation listing and may be chosen at will:

LIST

Specifies the location and format of the compilation listing.

SETTINGS

Specifies whether to include the complete list of compiler options in the compilation listing.

TRACE

Specifies whether tracing statements (`READY TRACE` and `RESET TRACE`) are obeyed.

WARNING

Specifies the verbosity of error messages printed in the compilation listing.

FLAG “*dialect*”

Specifies whether the compiler must produce language-level certification flags when it finds syntax that is not part of the specified dialect of COBOL.

Mandatory Options

The mandatory compiler options are listed below. For each of them, we indicate whether it is set by default, we give a brief description and we justify why we require it.

- `DIALECT"MF"` (default): sets the most "native" and efficient mode of operation. Since the aim of Oracle ART is not simply to emulate the source mainframe, but to forget about it and lead you towards the benefits of the target platform, this is the best choice. It will enable you to use the modern features of MF Cobol such as Unicode support and object-oriented programming.
- `CHARSET"ASCII"` (default): sets the default character set and collating sequence to those supported natively on the target platform. This is an obvious choice, too.
- `SOURCEFORMAT"FIXED"` (default): directs the compiler to stick with the "old" standard, fixed-format, column-based format. This may appear contradictory with our aim at modernity, but unfortunately the Oracle Pro*Cobol compiler, even the latest 11g version, is still not quite compatible with the MicroFocus free format, and we have to require fixed format to guarantee correct behavior.
- `ALIGN"8"` (default): defines the alignment for top-level structures (01 and 77-level). Required to make sure that structures retain the same layout as on the source platform, and yields the best performance, at a slight expense in memory size.
- `COMP5-BYTE-ORDER"NATIVE"` (default): uses native byte ordering for COMP-5 variables. Necessary for compatibility with the C language and the ART TP run-time system.
- `P64` (to set explicitly): compiles for 64-bit platforms. All the target platforms supported by ART are 64-bit.

- `SIGN"EBCDIC"` (to set explicitly): uses the EBCDIC convention rather than the ASCII convention for representing overpunched sign on `DISPLAY` numeric values. This is the same convention as on the source platform and hence provides for less differences of behavior when the last digit of such a numeric value is redefined by a character variable.
- `DEFAULTBYTE"00"` (to set explicitly, except if the previous option is given): specifies the value with which to initialize all otherwise-undefined variables in the Working-Storage Section. Not strictly necessary, since on the source platform, the Working-Storage Section is officially not implicitly initialized, but we found that it leads to less differences of behavior when a numeric variable is redefined by a character variable.
- `RWHARDPAGE` (to set explicitly): Causes the Report Writer control module to execute a form feed after the last item has been printed on a page, instead of the usual multiple blank lines. All Unix printer systems correctly handle Form Feed characters.
- `INDD` or `INDD"SYSIN80L"` (to set explicitly): causes "default" `ACCEPT` statements to read from the specified logical file instead of from the UNIX standard input file. This is the same behavior as on the source platform and is appropriate with ART-translated KSH scripts, which treats `SYSIN` as any other file for Cobol programs.
- `OUTDD` or `OUTDD"SYSOUT80L"` (to set explicitly): causes "default" `DISPLAY` statements to write to the specified logical file instead of to the UNIX standard output file. This is the same behavior as on the source platform and is appropriate with ART-translated KSH scripts, which treat `SYSOUT` as any other file for Cobol programs.
- `HOSTARITHMETIC`, `HOST-NUMMOVE "2"`, `HOST-NUMCOMPARE "2"`, `ARITHMETIC"ENTCOBOL"`, `CHECKDIV"ENTCOBOL"`, `FP-ROUNDING"ENTCOBOL"`, `REMAINDER "2"` (to set explicitly): all these options control various aspects of the treatment of numeric variables and expressions. Their settings maximizes the compatibility with the source platform.
- `IBMCOMP` (to set explicitly): turns on word-storage mode for the layout of the structures, the same mode as on the source platform and also the one yielding the most efficient performance, at a slight increase in memory consumption.
- `ODOSLIDE` (to set explicitly): Moves data items that follow a variable-length table in the same record as the table's length changes. This affects data items that appear after a variable-length table in the same record; that is, after an item with an `OCCURS DEPENDING` clause, but not subordinate to it. With `ODOSLIDE`, these items always immediately follow the table, whatever its current size; this means their addresses change as the table's size changes. With `NOODOSLIDE`, these items have fixed addresses, and begin after the end of the space allocated for the table at its maximum length. Setting `ODOSLIDE` leads to the same behavior as on the source platform.

- `PERFORM-TYPE"ENTCOBOL"` (to set explicitly): enables the same behavior as on the source platform regarding nested `PERFORM` statements with overlapping ranges. The default option `PERFORM-TYPE"MF"` is semantically cleaner and allows more efficient execution, but may lead to differences of behavior which are hard to detect and diagnose; hence, unless you know that your `PERFORM` ranges are "well-behaved" and never overlap, we can't recommend the default setting.
- `RDW` (to set explicitly): Enables you to find out the length of a record that has just been read from a variable-length sequential file, by providing a "hidden" length variable just before the first record of the FD (see more details in the Micro Focus documentation). This "trick" is available on the source platform, although not explicitly advertised, and this option allows to reproduce the same behavior.
- `RECMODE"ENTCOBOL"` (to set explicitly): directs the Micro Focus compiler to use the same algorithm as the source compiler to determine whether a file has fixed-length or variable-length for-mat, depending on the length of the various records in the file definition.
- `ASSIGN"EXTERNAL"` (to set explicitly): directs the MicroFocus compiler to use, by default, the `EXTERNAL` file-assignment method. In this method, the `SELECT` names are used as keys to search the actual file names in environment variables of the form `DD_NAME`. This is the mode chosen for ART, because it allows the file assignments to be specified outside the programs, namely in the calling KSH scripts. Not only is this closer in philosophy to the source behavior, but in our opinion this is the most flexible method.
- `SYSPUNCH"80"` (to set explicitly): defines the record length for the `SYSPUNCH` logical file. Default setting (132) is not the same as on the source platform.
- `ZEROLENGTHFALSE` (to set explicitly): When `ZEROLENGTHFALSE` is set, all comparisons between zero-length group items, and between zero-length items and figurative constants, return false; when it is not set, they all return true. To reproduce the same behavior as on the source platform, it must be set.
- `NOADV` (default): don't use special printer-control characters on text files. Target-platform print-ing utilities will simply print a file with the same layout as it appears on the screen.
- `NOTRUNCALLNAME` (default): does not truncate names of subprograms referenced in `CALL` state-ments. This is necessary for ART-migrated assets, because data access routines generated by ART have long names. In addition, in future evolutions of the asset, you will want to get rid of the short-names limitations imposed on the source platform.
- `NOTRUNCOPY` (default): does not truncate names of copy files referenced in `COPY` directives. This is necessary for ART-migrated assets, because copy files generated or inserted by ART have long names. In addition, in future evolutions of the asset, you will want to get rid of the short-names limitations imposed on the source platform.

- `NOCOPYLBR` (default): treat copy-file names as plain paths, not library archives (.lbr files). This is necessary for ART-migrated assets, because copy files converted or generated by ART are not grouped in archives.
- `REPORT-LINE "256"` (default): Specifies the maximum length of a Report Writer line.
- `COPYEXT "cpy, cbl"` (to set explicitly): Specifies the filename extension of the copyfile that the compiler is to look for if a filename in a `COPY` statement is specified without an extension. This non-default setting is appropriate for AST-migrated asset, because copy files generated by ART always have the .cpy extension and copy files converted by ART generally have the same extension. However, if you configure the Cobol converter for another extension, you will have to adapt the setting of this option appropriately.

After taking into account default and explicit options, and dependencies, the required option list must start with the following:

Listing 10-5 Required Compiler Option List

```
P64 SIGN"EBCDIC" RWHARDPAGE INDD OUTDD HOSTARITHMETIC HOST-NUMMOVE"2"
HOST-NUMCOMPARE"2" ARITHMETIC"ENTCOBOL" CHECKDIV"ENTCOBOL"
FP-ROUNDING"ENTCOBOL" IBMCOMP ODOSLIDE PERFORM-TYPE"ENTCOBOL" RDW
RECMODE"ENTCOBOL" REMAINDER"2" ASSIGN"EXTERNAL" SYSPUNCH"80"
```

ORACLEDBTUPCALLS COPYEXT="cpy, cbl"

Note: Mandatory Oracle SQL compilation options for the Pro*Cobol preprocessor are listed in the ART Workbench RDBMS Converter Reference Guide.

Installation-dependent options

These options are not strictly necessary but may help you handle assets in which programs contain a mixture of upper-case and lower-case letters:

- `FOLDCALLNAME "UPPER"` (to set explicitly): directs the compiler to map subprogram names in `CALL` statements to upper case.
- `FOLDCOPYNAME "UPPER"` (to set explicitly): directs the compiler to map copy file names in `COPY` directives to upper case.
- `MAPNAME` (to set explicitly): Makes the Compiler alter program-names and entry-point names to make them compatible with the source platform.

By experimenting with these settings, you may find the combination which is appropriate for your particular asset. For instance, `FOLDCALLNAME"UPPER"` and `MAPNAME` taken together provide a good enough emulation of the `PGMNAME(COMPAT)` source-compiler option, but there is no sure way to emulate the other values of this option...

Options depending on customer choice

The following options influence the behavior of the target asset, but may be set more or less at will by the user of the ART system.

- `BOUND` and `SSRANGE` check that each index is between the correct bounds when accessing an array or in reference modifiers. This is similar to the `SSRANGE` option of the IBM compiler. We strongly recommend that both of these options be set, at least during migration tests and in the first few months of operation (note that setting `SSRANGE` also sets `BOUND`). This choice is a bit controversial because it can break some programs which apparently run correctly on the source platform (without the `SSRANGE` option). However, in our experience, the only programs which break are incorrect programs which just happen to work by chance on the source platform and would not work in the same way, or at all, on the target platform. The sooner we detect these programs and fix them, the better. In the same way, you could consider setting the `CHECK` option, which enables various (other) kinds of run-time checks and allows to detect other kinds of seem-ingly-correct programs.
- `TRUNC`: specifies whether truncation to the given PIC size occurs when assigning a value to a `BINARY` variable (or `COMP`, or `COMP-4`). This is similar to the `TRUNC(STD)` option of the IBM compiler. However, with the present specification of the ART Cobol converter, all such variables are transformed into `COMP-5` variables, which do not obey the `TRUNC` option. See the discussion in `Use of COMP-5 Type` and the `TRUNC Compiler Option`.
- `APOST` and `QUOTE`: allow to choose which character, single or double quote, the `QUOTE` symbolic constant will represent. This is similar to the IBM options of the same name. Use the same setting as on the source platform.
- `NOALTER`: forbids the presence of `ALTER` statements in the Cobol programs. Since `ALTER` statements are a thing of the past, and a very bad thing at that, we recommend that you take the opportunity of migrating your asset with the ART Workbench to chase out any remaining `ALTER` clauses. Then, set this option to prevent their reappearance and to make compiled code more efficient and safe.
- `AREACHECK`: Causes the Compiler to treat any token which starts in area A in the Procedure Division as a paragraph or section label, regardless of the preceding tokens. If `AREACHECK` is not specified, only tokens which follow a period are treated as possible labels. This

directive provides closer compatibility with IBM error handling, where omitting a period before the label produces a less serious message. We recommend that such erroneous source code is corrected.

- **NOBYTEMODEMOVE**: Controls behavior for alphanumeric moves between overlapping data items. If **BYTE-MODE-MOVE** is specified, data is moved one byte at a time from the source to the target. If **NOBYTE-MODE-MOVE** is specified, the data is moved in granules of two, four or more bytes at a time (depending on environment) from the source to the target. Consequently, if the overlap is less than the size of the granule, each granule moved overwrites part of the next granule to be moved. **NO-BYTE-MODE-MOVE** gives better performance, but may yield incorrect code on some very rare programs which work correctly on the source platform; we suggest that you start with the "more compatible" setting (**BYTE-MODE-MOVE**), perform complete regression tests until satisfaction, then choose the other option and re-test.
- **DYNAM**: Specifies that **CANCEL** statements are not to be ignored. This is similar to the IBM option of the same name (but not quite the same, see the Micro Focus documentation). We strongly recommend that you set this option, because the Tuxedo servers in the ART TP Runtime system, which execute the applicative CICS programs, use **CANCEL** statements to free the memory used to load and run those programs. If **NODYNAM** is in effect, the amount of memory used by these servers would grow as they execute more and more different programs.
- **NOFDCLEAR**, **NOHOSTFD**: The "positive" settings of these options reproduce the restrictions on **FD** usage imposed by the IBM compiler (**FD** records allocated only at **OPEN** time, record contents lost after **WRITE**, etc.). We feel that these restrictions are silly and hence recommend that you don't use these options.
- **NATIVEFLOATINGPOINT**: see the discussion in Use of Floating-point Variables.
- **NOSEG**: turns off segmentation and ignores all segment numbers. The resulting program is a single piece with no overlay. Who still uses segmentation, anyway?
- **STICKY-LINKAGE" 2 "** / **NOSTICKY-LINKAGE**: this option controls how a program parameter (Linkage Section item) which has been linked to some actual data item in a previous invocation of the program may be re-linked with the same item if the current invocation specifies no new linking (no actual argument supplied). The **STICKY-LINKAGE" 2 "** setting is "more compatible" with the behavior of the source platform, especially for CICS programs, but it is certainly non-standard and error-prone. It may also be incompatible with certain features of the ART TP runtime system, in particular the possibility to distribute TP transactions over several servers running in a cluster with no shared memory. So we strongly suggest to use the default **NOSTICKY-LINKAGE** setting

from the beginning and fix any sticky-linkage-related bugs discovered during regression testing. See also the discussion in Linkage-Section Arguments with NULL Address.

Options influencing compile-time operation

The following options influence only the production of the compilation listing and may be chosen at will:

- **LIST**: specifies the location and format of the compilation listing.
- **SETTINGS**: specifies whether to include the complete list of compiler options in the compilation listing.
- **TRACE**: specifies whether tracing statements (**READY TRACE** and **RESET TRACE**) are obeyed.
- **WARNING**: specifies the verbosity of error messages printed in the compilation listing.
- **FLAG"dialect"**: specifies whether the compiler must produce language-level certification flags when it finds syntax that is not part of the specified dialect of Cobol.

To guarantee identical behavior between the source programs and the target ones produced by the Cobol Converter, in light of the limitations previously described, the target programs must be compiled with a certain set of compiler options in effect. Indeed, some of the MicroFocus Cobol compiler options do change the behavior of the executed code. The transformations applied by the the Rehosting Workbench Cobol Converter are hence tailored to the option set described below. No support will be provided for programs compiled with a different, or at least conflicting, option sets. For more information, please see the MicroFocus Cobol documentation, in particular the Compiler Directives book.

The main behavior-influencing option to set mandatory is **DIALECT"ENTCOBOL"**. Indeed, this dialect option sets a number of sub-options, such as **PERFORM-TYPE"ENTCOBOL"**, which make the target program behave as closely as possible to the original source program compiled by the IBM Enterprise Cobol Compiler.

However, the Refine Cobol converter departs from the Enterprise Cobol basic choices by using the native character set of the target platform, namely ASCII. This mandates to set the option **CHARSET"ASCII"**. Conversely, it sticks to the IBM convention for representing overpunched sign on **DISPLAY** numeric values, so the option **SIGN"EBCDIC"** must be set.

The following minor options must also be set to guarantee the correct behavior of target programs:

- NOADV: do not use special printer-control characters on text files. Target-platform printing utilities will simply print a file with the same layout as it appears on the screen.
- ALIGN"8": 01- and 77-level data items are aligned at the "most universal" memory boundary.
- BOUND: check that each index is between the correct bounds when accessing an array. This choice is a bit controversial because it can break some programs which apparently run correctly on the source platform. However, in our experience, the only programs which break are incorrect programs which just happen to work by chance on the source platform and would not work in the same way, if at all, on the target platform. The sooner we detect these programs and fix them, the better.
- COMP-5"2": use native byte ordering for COMP-5 variables. This is necessary for compatibility with the the Rehosting Workbench CICS Runtime routines, among others.
- NOCOPYLBR: copy files are just plain files, not .lbr library files.
- HOSTARITHMETIC: try to comply with IBM behavior on size error conditions in arithmetic computations.
- INTLEVEL"4": allow numeric variables up to 38 digits, and more generally use improved arithmetic behavior.
- REPORT-LINE"256": specifies maximum line size for Report Writer.
- RWHARDPAGE: use "hard" Form Feed (FF) characters to jump to a new page in Report Writer, instead of using multiple Line Feeds. FF is recognized as jumping to a new page by all target-platform printing utilities.
- NOTRUNCALLNAME: do not truncate the names of CALLED subprograms to 8 characters, as the ENTCOBOL dialect would normally do, because the Oracle Tuxedo Application Rehosting Workbench Cobol Converter uses longer names (and this is better for future evolution anyway).
- NOTRUNCOPY: same thing for names of COPY files.
- The following options do not influence run-time behavior and can be set according to site convention or requirements:
- COPYEXT: specifies the file extensions used for copy files. To set according to the choices you made during migration (see the target-copy-extension configuration clause).
- LIST: specifies the location and format of the compilation listing.

- **SETTINGS:** specifies whether to include the complete list of compiler options in the compilation listing.
- **TRACE:** specifies whether tracing statements (READY TRACE and RESET TRACE) are obeyed.
- **WARNING:** specifies the verbosity of error messages printed in the compilation listing.

COBOL-IT

To reproduce the source COBOL compiler behavior, COBOL-IT offers an IBM compatible configuration file (ibm.conf). This configuration file will be used to compile the target COBOL asset.

In addition to the compiler options set in the configuration file ibm.conf, at least the following options must be added to improve compatibility between the source and the target COBOL environments.

External-mapping

If set to yes, all the file names of the file declared as EXTERNAL are resolved at runtime using environment variables. It must be set to yes.

Binary-truncate

Binary-truncate is a boolean operator that governs the behavior of the runtime when binary data is truncated. It must be set to no. This corresponds to the behavior of the MicroFocus compiler directive NOTRUNC.

Spzero

If set to yes, Space character moved to NUMERIC USAGE field are converted in '0'. It must be set to no.

Depending on the customer needs, other compiler options can be set. To learn about the COBOL-IT compiler options, please refer to the COBOL-IT Compiler Suite Enterprise Edition - Reference Manual.

Detailed Processing

Overview

When the COBOL Converter starts, it reads and checks the various configuration files, starting with the main one. If any inconsistency is detected at this stage, one or more error messages are printed and the converter exits. Otherwise, the converter uses both command-line options and

configuration-file options to set its internal parameters, including the list of (source) programs to process. Then it proceeds to process each of these programs in turn; for each of them:

1. According to the make-like, incremental behavior of the Converter, it checks whether the target program already exists and is up-to-date with reference to the POB file for its corresponding source program. If not, the converter skips to the next program. Otherwise, it continues with the next step.
2. The POB file for the program is loaded. The converter then checks whether it contains FATAL errors. If so, and unless the force-translation flag is set on the command-line or in the configuration file, it prints out a warning message and skips to the next program. Otherwise, it continues with the next step.
3. The various transformation rules are then applied on the program AST, in several passes. Each transformation modifies the AST and then updates the program layout accordingly (textual appearance).
4. The (text of the) resulting AST is then printed out in the target program file. When the beginning of a copy file is encountered, the COPY clause is written to the caller file and the sequel of the output is diverted to a new output file for this copy file, in a private directory; if a file with this name already exists in the directory, it probably is because the same copy file is included more than once in the program, and the new file carries a new version number (the existing version is not overwritten). If the copy file was invoked with a REPLACING clause, the effects of the replacements are undone before the file is printed out (see the caveat [Miscellaneous Issues](#) regarding interferences between transformations and replacements). When the end of the copy file is reached, output is reverted to the caller file. This allows to correctly handle nested copy files.
5. If the post-translation file is specified in the configuration file, it is exercised by the post-translator on the main target program file and on all target copy files in the private directory.
6. Lastly, if the [deferred-copy-reconcil Clause](#) is not given, either on the command-line or in the configuration file, the copy reconciliation process is applied to the target copy files in the private directory.

The converter can be executed by several concurrent processes at the same time, provided that the [deferred-copy-reconcil Clause](#) is given either on the command-line or in the configuration file; otherwise, the copy-reconciliation phase of these concurrent processes may run into access conflicts over the "data-base" of final, reconciled copy files, which could lead to corrupted results.

Command-Line Syntax

Refine Launcher Interface

The COBOL Converter is designed to be run through the refine command, which is the generic Oracle Tuxedo Application Rehosting Workbench launcher and is also used to launch all "big" Oracle Tuxedo Application Rehosting Workbench tools. This launcher handles various aspects of the operation of these tools, such as execution log management and incremental/repetitive operation.

cobol-convert Command

Synopsis

```
$REFINEDIR/refine cobol-convert [ launcher-options... ] \  
    ( -s | -system-desc-file ) system-desc-path \  
    ( -c | -config ) main-config-file-path \  
    [ other-specific-flags... ] \  
    ( source-file-path | ( -f | -file | -file-list-file ) file-of-files  
)...
```

Options

The mandatory options are:

- (-s | -system-desc-file) system-desc-path
Specifies the location of the [System Description File](#). As usual for Unix/Linux commands, the given path can be absolute or relative to the current working directory. Note that many other paths used by many of the Rehosting Workbench tools are then derived from the location of this file, including that of the main configuration file (see next option); this makes it easy to run the same command from different working directories.
- (-c | -config) main-config-file-path
Specifies the location of the [Main Conversion Configuration File](#). The given path can be either an absolute path or a relative path; in the latter case, it is relative to the directory containing the system description file, as usual for the Rehosting Workbench tools.

The generic options which define which source programs to process are:

`source-file-path`

Adds to the work-list the program source file designated by this path. The path must be given as relative to the root directory of the system, `$SYSROOT`, even if the current working directory is different.

`(-f | -file | -file-list-file) file-of-files`

Adds to the work-list the program source files listed in the file designated by this path. The file-of-files itself may be located anywhere, and its path is either absolute or relative to the current working directory. The program source files listed in this file, though, must be given relative to the root directory of the system.

You can give as many individual programs and/or files-of-files as you wish. The work-list is built when the command line is analyzed by the COBOL Converter, see the detailed description above.

The optional specific flags or options are:

`-dcrp` or `-deferred-copy-reconcil`

Has the same effect as the [deferred-copy-reconcil Clause](#) of the configuration file, namely to not run the copy reconciliation process incrementally after converting each program. Only with this clause or flag can the COBOL converter run in multiple concurrent processes.

`-tpe extension` or `-target-program-extension extension`

This option has the same effect as the configuration-file clause of the same name, and overrides it when given.

`-tce extension` or `-target-copy-extension extension`

This option has the same effect as the configuration-file clause of the same name, and overrides it when given.

`-keep` or `-keep-same-file-names`

This option has the same effect as the configuration-file clause of the same name, and overrides it when given.

`-force` or `-force-translation`

This option has the same effect as the configuration-file clause of the same name, and overrides it when given.

`-cics` or `-activate-cics-rules`

This option has the same effect as the configuration-file clause of the same name, and overrides it when given.

Repetitive and Incremental Operation

Even with the powerful computing platforms easily available nowadays, processing a complete asset using the Rehosting Workbench remains a computing-intensive, long-running, memory-consuming task. The Work-bench tools are hence designed to be easily stopped and restarted and, thanks to a make-like mechanism, not repeat any piece of work which has already been done. This allows efficient operation in all phases of a migration project.

Initial Processing: Repetitive Operation

In the initial phase, when starting with a completely fresh asset and up to the end of the first conversion-translation-generation cycle of a stable asset, the make-like mechanism is used to allow repetitive operation, as follows:

1. When the COBOL Converter starts, it begins by studying the current state of the asset (source files and target files such as the target program files) and determining what work remains to do to reach a complete and consistent set of results. It then undertakes this work, producing more and more result files.
2. As the volume of processed files grows, the Rehosting Workbench process consumes more and more memory.
3. Regularly, the tool checks whether the available physical memory drops below the threshold set by the minimum-free-ram-percent option in the system description file.
 - If the work to be performed is complete before running out of memory, the process definitely stops.
 - Otherwise, the process stops but restarts immediately, after memory is freed. Going back to step 1 above, there is less work to do, so that the process eventually terminates.

This mode is particularly well suited for tools or commands which operate globally on the whole asset such as the Cataloger, but it is also useful for component-wise tools such as the COBOL Converter. This is the normal mode of operation for the Rehosting Workbench tools and there is nothing specific to choose it.

Changes in the Asset: Incremental Operation

The COBOL converter knows the dependencies between the various components (main program files) and associated result files (POB files, target program files). Using this information, it is able to react incrementally when some change occurs in the asset. For example, when a COBOL source file is added, modified or removed: the cataloguer re-parses the affected programs, and

then the COBOL converter re-converts only those. Again, this is the normal mode of operation for the Rehosting Workbench tools and there is nothing specific to choose it.

Common Information

COBOL Reloading Programs Reserved Words List

MW-ENTREE

MW-SORTIE

MW-BINARY-DATA

MW-REL-KEY

MW-SEQ-KEY

MW-NB-INSERT

Future use: MW-IND-OCC-<copy-field_name>

MW-COUNT-NUMERIC-BCD-USE-X

MW-COUNT-NUMERIC-DISP-USE-X

MW-DISCRIM-TO-TRANSCODE

MW-CHECK-ONE-CHAR

MW-CHECK-ONE-CHAR-SPECIAL-SIGN

DISCRIM-RULE-RESULT

IO-STATUS

D-NB-RECS

D-NB-RECS-MAX

TR-<copy-field_name>

TRANSCODE-STR

TRANSCODE-STR-LENGTH

TRANSCODE-LENGTH

TRANSCODE-SOURCE

TRANSCODE-CIBLE

MW-FILE-END-OF-LINE