



Oracle® Documaker Connector
Installation, Administration and Customization Guide
Release 1.0.0.0.0 for Windows x86

July 2009

Oracle Documaker Connector Installation, Administration and Customization Guide, Release 1.0.0.0.0 for Windows x86

Copyright © 2009, Oracle and/or its affiliates. All rights reserved.

Primary Authors: Eugene Thompson, Joe Roberts, Steve Saunders

Contributing Authors: Lowell Von Egger, Phil Iorio

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States 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, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Contents

Preface	1
Audience	1
Documentation Accessibility	1
TTY Access to Oracle Support Services.....	1
Related Documents	1
Conventions	1
Introduction	3
Overview	3
Compatibility	3
Planning.....	4
Installation	5
Installation.....	5
Prerequisites.....	5
Running the Installer.....	5
Reviewing the Installation.....	7
Microsoft Windows Installation Files	7
Required Support Installers	8
Installer Support.....	8
Java Application and Support Files.....	8
Removing an Installation	9
Administration	11
Basic Configuration.....	11
Oracle Documaker Configuration	11
Making Index Data Available in Documaker GVM Variables.....	12
Generating Database Table Rows and Writing the Document Files	12
Documaker INI Setup for the Example Database Connection.....	15
Oracle UCM Configuration	16
Oracle Documaker Connector Configuration	18
The Logging Properties File.....	18
The Engine Properties File	20
The Documaker Source System Properties File.....	22
Running the Program.....	24
Modes of Operation.....	24
How Oracle DC Determines the Run-Mode.....	25
Stopping a Server-Mode Instance	25
Command-line Parameters Detail	25
Examples	25
Customization for Developers	27
Architecture.....	27
Class Structure	27
Source Implementation	28
The Connector Engine	28
Client Implementation.....	28
Functional Breakdown	28
Configuration	29
Initializing the Logging Framework	29
Parsing the Configuration File.....	29
Command-line Processing.....	29
Initializing the Connector.....	29

Processing.....	30
The BatchManager.....	30
The BatchProcessors.....	30
Shutdown.....	30
Core Engine Interfaces	31
The Source System.....	31
Overview	31
The Source Interface in Java	31
The Housekeeping Interface Overview.....	33
The Housekeeping Interface in Java	33
The Client System	33
The ClientAdministration Interface Overview.....	33
The ClientAdministration Interface in Java	33
The Client Interface Overview.....	34
The Client Interface in Java	34
Other Interfaces.....	35
The Documaker Source System Implementation.....	35
Source-Interface Implementation	35
configure Method:	35
getHouseKeeper Method:	35
getFileBatch Method:	35
ackProcess Method:.....	36
isValid Method:.....	36
setMaxBatchSize Method:	36
close Method:.....	36
getTableColumns Method:.....	36
Housekeeping-Interface Implementation	36
cleanUp Method:	36
close Method:.....	36
Appendix A – Windows Service Application	37
dm_ucm_connector.exe Service Application.....	37
The dmservice.dll	37
The dm_ucm_connector.properties File.....	37
Appendix B – Example Files Using TRN_FIELDS INI Options to Map Index Data in Documaker	39
XML Extract Input to Documaker.....	39
Example TRN_FIELDS INI Setup.....	42
Example TRNDFDFL.DFD Setup.....	43
Example RCBDFFDFL.DFD Setup.....	49
Appendix C – Example Documaker DAL Scripts.....	51
BatchBannerBeginScript = AOR_PREB	51
TransBannerBeginScript = AOR_PRET.....	52
TransBannerEndScript = AOR_POSTT	52
BatchBannerEndScript = AOR_POSTB.....	55
Internal Routine: AOR_NEWFILE.....	55
Internal Routine: AOR_NEWPATH.....	56
Internal Routine: AOR_EOB.....	56
Appendix D – Documaker Setup for DAL Output to a Database Table.....	59
DDL for DAL Output Records	59
Database Table DFD (Data Format Definition) File (AOR.DFD).....	60
The AOR.DFD File	60

Preface

Note: This document is accurate at the time of publication. Oracle will update the documentation periodically after the software release. You can access the latest information and additions to this document on the Oracle Technology Network (OTN) at:

<http://www.oracle.com/technology/documentation/>

This document contains information necessary for the installation and configuration of Oracle Documaker Connector (Oracle DC).

This Preface includes the following topics

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

This document is intended for users who want to install or administer Oracle DC. Experience installing Oracle Documaker and experience as a system administrator is necessary.

Documentation Accessibility

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents

For more information, refer to the following Oracle resources:

- The Oracle Documaker documentation set, specifically:
 - *Documaker Server Installation Guide*
 - *DAL Reference*
- The Oracle Universal Content Management (UCM) documentation set, specifically:
 - *Oracle Universal Content Management Product Overview*
 - *Getting Started with Content Server*
 - *Oracle Universal Content Management Content Server Installation Guide for (Platform)*
 - *Managing Repository Content*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.
#	Line contains operating system commands. Do not enter the leading #.
SQL>	Line contains SQL*Plus commands. Do not enter the leading SQL>.

Introduction

Oracle Documaker Connector (Oracle DC) is a Java application that works with Oracle Documaker scripting to archive Documaker output documents into Oracle Universal Content Management (UCM). This document describes for administrators the Oracle DC, its installation, configuration and operation. Once familiar with the material in this guide and other prerequisite background information, an administrator should be able to plan and execute an implementation of Oracle DC.

In addition to this guide, implementation of Oracle DC requires familiarity with Oracle Documaker configuration and Document Automation Language (DAL) scripting, as well as configuration of Oracle Universal Content Manager. This guide is not a substitute for understanding both of these products and their documentation. You must have the requisite skills in both Oracle Documaker and Oracle Universal Content Manager (UCM) to configure those products to work with this product which connects the two of them.

Overview

Oracle Documaker is a publishing engine which is driven by input **transactions** to produce output document sets based on the input transaction data, a repository of forms and a powerful rules engine which selects form information and composes custom documents based on the incoming transaction data. Each **transaction** is a set of variable data field values and other form-selection criteria. Oracle Documaker applies the variable data to fill in fields in the forms, optionally performing formatting and/or calculations on the data before using it as text on the forms. The data values can also affect which forms are used for the documents resulting from the input transaction. Because the output documents were originally designed and composed to be printed, the output documents are sometimes called **print streams**. Oracle Documaker DAL (Document Automation Language) scripting is used to place each output document in a file system directory and write a matching record containing the indexing information (metadata) into a database table. Oracle DC monitors the database table for new entries and then processes those entries by sending the document and indexing data into Oracle Universal Content Manager. Oracle DC may be run continuously, as a system service, or may be run periodically, for example after a batch run of Oracle Documaker.

The Oracle DC is configured with the connection information for the source database table and the connection information for UCM. The contents of the database table and the coincident configuration of the UCM metadata will vary with the document data particular to a specific customer implementation. It is this configuration which requires knowledge of Oracle Documaker on the one hand and UCM administration on the other. There are also performance and volume-related configuration parameters.

Connectors from Documaker to other repository systems may be created from the core components of the Oracle Documaker Connector: The client interface to UCM may be replaced with a new client interface to a different system. The Connector may also be used as a framework for creating new applications which draw documents from sources other than Documaker. That is, both *ends* of the Connector application are completely replaceable and therefore customizable.

Compatibility

Oracle DC is tested with Oracle Documaker version 11.3 and is compatible with Documaker versions 11.2 and newer. The UCM Client interface was implemented and tested with UCM release 10gR3, version 10.1.3. There are additional compatibility requirements for both Oracle Documaker and Oracle Universal Content Manager. The product documentation for those products, and in particular the specific versions you plan to be running, should be consulted as a part of your system planning.

Planning

Before implementing the Oracle Documaker Connector, there are a few planning exercises that need to be completed. You'll need these answers to configure Documaker, UCM, and the Documaker Connector to all work together to provide the service you desire.

First, you must decide which of your Documaker-produced documents you want to save in your archive. You may be publishing several versions of the same basic document (different recipient copies, for example) and archiving more than one of them would be duplicative and inefficient.

Next, for those documents, you must decide what output format you want to publish to the archive process. This should be a format which your end-users will be able to retrieve and view easily. A popular choice here is Adobe Portable Document Format (PDF).

Finally, you must decide how you expect your users to retrieve those documents. That is, what information will users have in hand to provide to the retrieval interface to search and find the specific documents they want? This is generally a subset of the variable data published on the documents themselves. Examples might include customer/account number or ID, customer name, date of issue, telephone number, postal code, and so on. This data must be saved into a database table which you customize. This is done by a Documaker Document Automation Language (DAL) script you provide. Each row in the table represents a document to be archived and the data by which it will be indexed in UCM.

To summarize, before you embark on an implementation, you need to have decided the following:

1. Which documents will be archived?
2. Which output format, such as PDF, will be sent to the archive?
3. What variable fields will be used to organize and retrieve the documents? These fields are called the **metadata** to be stored along with the documents.

Installation

Installation of the Oracle DC places the necessary program files on the target system at the requested location. Following the installation process, the product must be configured to work with your Documaker implementation, to process the desired documents with the correct indexing fields and to connect and import into the UCM system. Documaker itself also must be configured and customized to produce the documents to be archived along with their desired indexing data. This is a separate step that can be done without installation of the Oracle Documaker Connector. This step requires expertise in Documaker configuration, Document Automation Language (DAL) scripting for Documaker and some familiarity with SQL database table definition or modification.

Installation

Oracle DC is installed on all platforms with an installation application. This installer places the program and supporting files on the target system and performs initial configuration of the target system to run the Oracle DC. On Microsoft Windows systems, this includes installation of the Oracle DC as a Windows service and installation of a taskbar **notification icon** which controls the running of the service.

Prerequisites

1. Platform support of Java 5 (compatible JVM), including JDBC database connectivity.
2. Connections to a SQL database and file system directory which are both also accessible to Oracle Documaker.
3. Connections to Oracle Universal Content Manager and a file system directory also accessible to Oracle Universal Content Manager.

Running the Installer

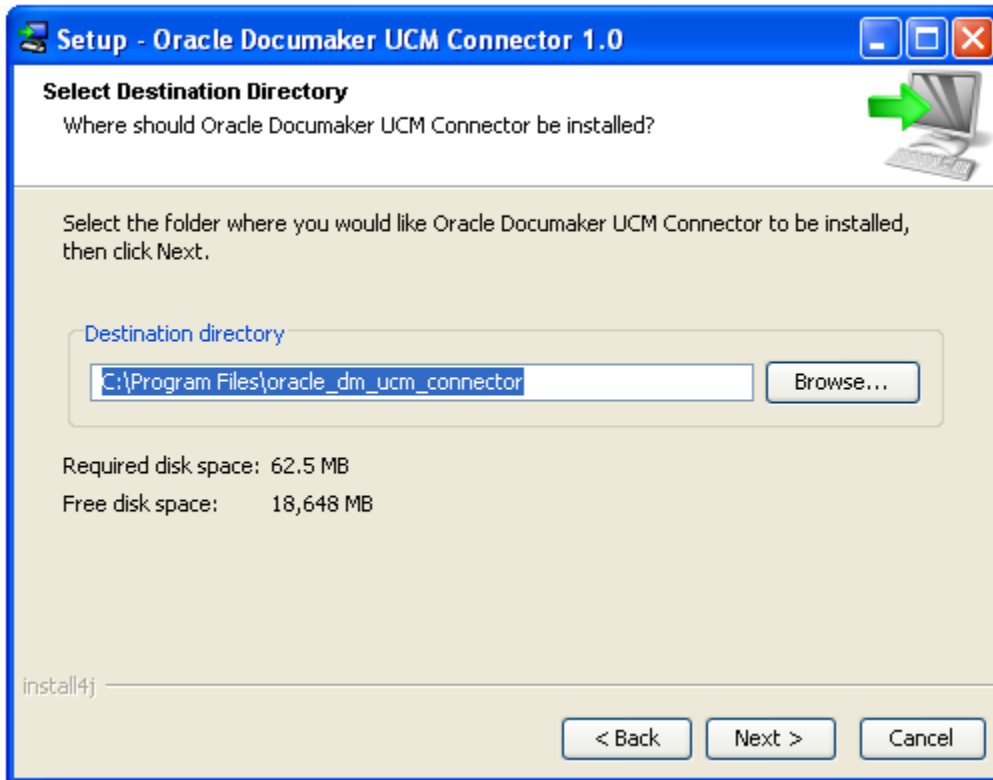
The process of running the installer in a graphical environment is similar across all supported platforms. These examples are from Microsoft Windows, but should be easily used as a guideline in other environments.



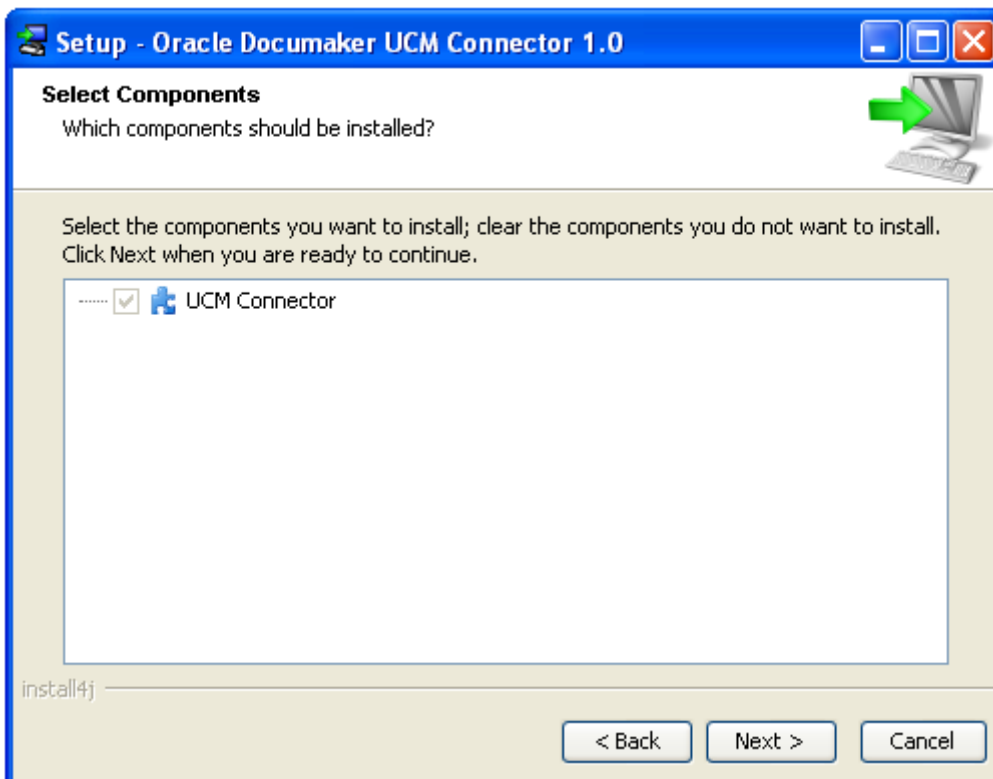
The installer is a self-contained application file. On Microsoft Windows, the installer is named `oracle_dm_ucm_connector_windows_1_0.exe`.

Double-click or otherwise execute the application file. The installer application launches and displays the welcome screen. Click Next to continue the installation.

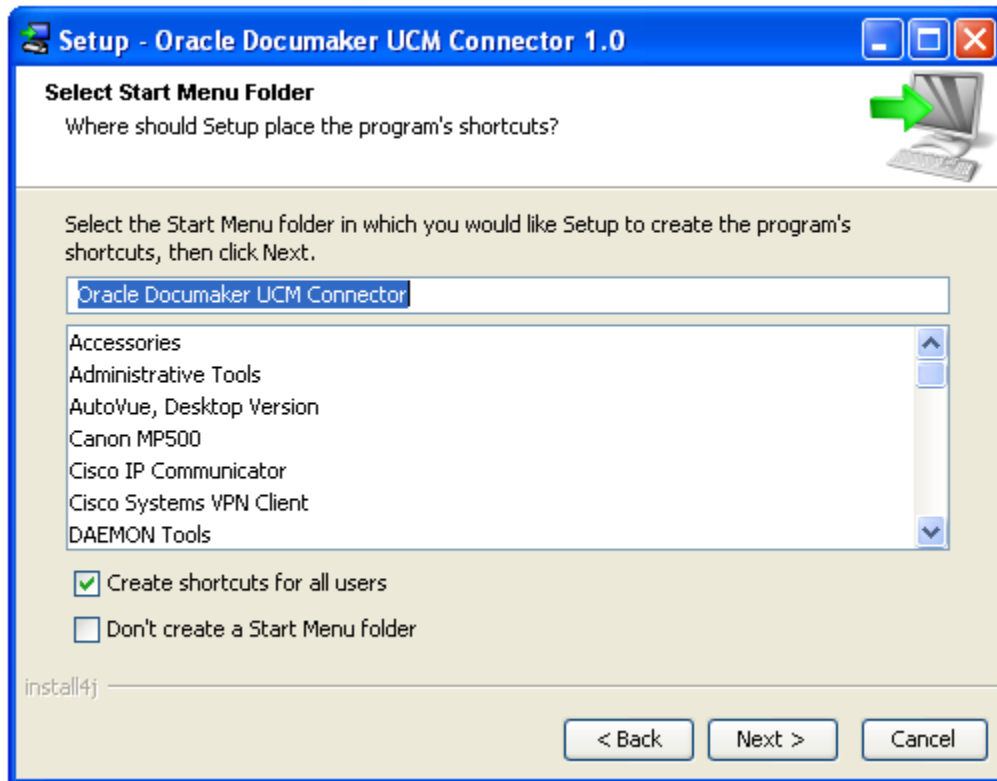
The next screen allows the choice of an installation target directory. On Windows, the default directory is under the normal Program Files directory and is shown in the figure below. The directory chosen on this screen is referred to throughout this document as the installation **Target Directory**.



Clicking Next > accepts the directory choice and moves to the Select Components screen. There is only one component in the product so nothing to see here. Move along by clicking Next >.



The next screen, `Select Start Menu Folder`, lets you setup a Start Menu item which contains a shortcut to the un-installation program. To skip creating any folder at all in the Start menu, check the box next to `Don't create a Start Menu folder`.



Checking `Create shortcuts for all users` causes the program to be installed so all users on the machine will have it in their Start menus as a choice. If this box is not checked, only the user running the installation will have the Start menu entry created.

Clicking `Next >` on this screen starts the installation process. The files are moved to the selected location and other required installations are then run. The installer puts a Java run-time environment (JRE) in a subdirectory of the target directory called `jre`. On Microsoft Windows, the installer runs the VC 2005 SP1 redistribution package from Microsoft. This installs the Visual C 2005 run-time if it is not already present on the machine.

The installer program sets the `service.path` in the `dm_ucm_connector.properties` file to the JRE that was installed.

On Windows, the installer runs the `dm_ucm_connector.exe` program with an `install` parameter to install the Windows service. (Running this program later with a parameter of `uninstall` will remove the Windows service.

Reviewing the Installation

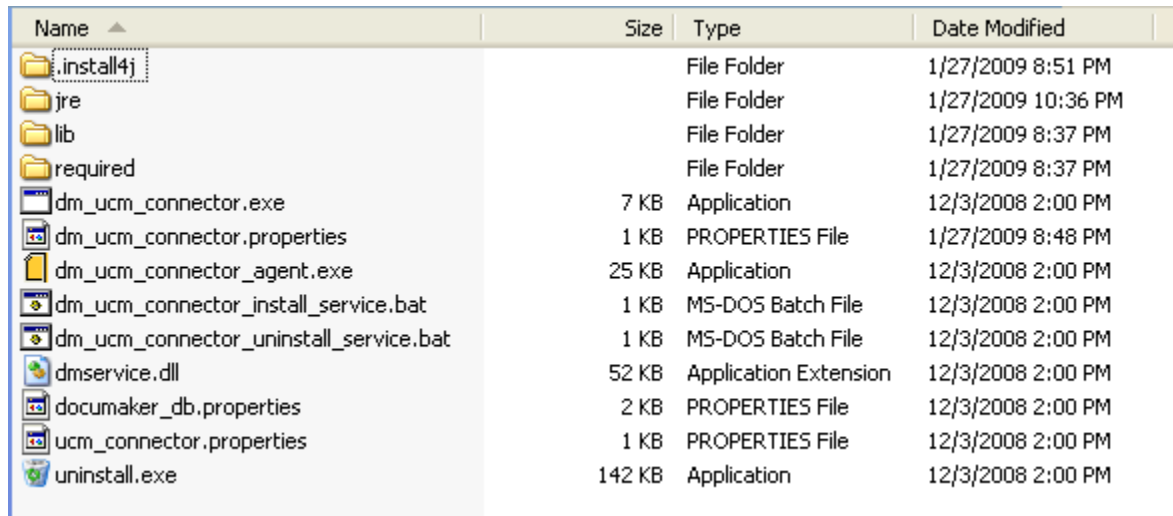
Microsoft Windows Installation Files

Although there are slight differences by platform, most of the installed files are the same regardless of the target system. The figure below is from a Windows-based installation.

Oracle DC as a Windows Service

The `dm_ucm_connector.exe` application runs Oracle DC as a Windows Service. Details on how it functions are in Appendix A. There are two batch file scripts which can be used to install and un-install the Windows service. (The installation runs the `install_service.bat` file.) These scripts run the `dm_ucm_connector.exe` application. Related files are `dm_service.dll`, `dm_ucm_connector.properties`, `dm_ucm_connector_install_service.bat`, `dm_ucm_connector_uninstall_service.bat`.

The small application, `dm_ucm_connector_agent.exe`, places an icon in the Windows task bar notification area. The icon shows the status of the Oracle DC Windows service and right-clicking the icon allows a user to open the log file and to start and stop the Windows service without opening the Services Control Panel.



Name	Size	Type	Date Modified
.install4j		File Folder	1/27/2009 8:51 PM
jre		File Folder	1/27/2009 10:36 PM
lib		File Folder	1/27/2009 8:37 PM
required		File Folder	1/27/2009 8:37 PM
dm_ucm_connector.exe	7 KB	Application	12/3/2008 2:00 PM
dm_ucm_connector.properties	1 KB	PROPERTIES File	1/27/2009 8:48 PM
dm_ucm_connector_agent.exe	25 KB	Application	12/3/2008 2:00 PM
dm_ucm_connector_install_service.bat	1 KB	MS-DOS Batch File	12/3/2008 2:00 PM
dm_ucm_connector_uninstall_service.bat	1 KB	MS-DOS Batch File	12/3/2008 2:00 PM
dm_service.dll	52 KB	Application Extension	12/3/2008 2:00 PM
documaker_db.properties	2 KB	PROPERTIES File	12/3/2008 2:00 PM
ucm_connector.properties	1 KB	PROPERTIES File	12/3/2008 2:00 PM
uninstall.exe	142 KB	Application	12/3/2008 2:00 PM

Required Support Installers

Any required support products which have their own installers are placed in the `required` subdirectory. On Windows, for example, this folder contains the installer for the Microsoft Visual Studio C 2005 Runtime libraries.

Installer Support

The installer places its own support files in the `.install4j` subdirectory. This directory should not be disturbed, as it contains information used for a subsequent un-installation. Un-installation is done by running the `uninstall.exe` application.

Java Application and Support Files

All the Java Jar files for the application and the required support code are installed in the `lib` subdirectory. These JAR files must be in the Java classpath for the Oracle DC to run (either from the command line or as a service). Most of these files are installed as part of the normal installation process, but the list is provided here for information useful in troubleshooting or customizing the install.

The default logging properties file `log4j.xml` is installed in the `lib` folder, with the `.jar` file listed below.

Name	URL
Core Connector Engine – Required Jar Files	
UCMImporter.jar	Provided with this application
UCMImporterLib.jar	Provided with this application
oracle-ridc-client-11g.jar	Provided with this application
commons-codec.jar	http://commons.apache.org/codec/
commons-httpclient-3.1.jar	http://hc.apache.org/httpclient-3.x/
commons-logging-1.1.1.jar	http://commons.apache.org/logging/
Core Connector Engine – Required Jar Files	
UCMImporterLib-javadoc.jar	Provided with this application
Documaker Source Component – Required Jar Files	
DBUtil.jar	Provided with this application
DocuCorpUtil.jar	Provided with this application
DocumakerSource.jar	Provided with this application
commons-beanutils.jar	
commons-collections-3.2.jar	
commons-dbcp-1.2.2.jar	
commons-lang-2.3.jar	
commons-logging-1.1.jar	
commons-pool-1.3.jar	
DdlUtils-1.0.jar	
jakarta-oro-2.0.8.jar	
log4j-1.2.15.jar	
Service Execution Utility	
DocucorpStartup.jar	Provided with this application

Table 1. Required Java Library Support Files (JARs)

Note: In addition to the files supplied with the application, you must assure that the appropriate jar files for your database provider are also included in the classpath. These files are **not** included in the installation of the Oracle Documaker Connector.

Some examples of the Jar files required for the JDBC connectivity for various database brands:

Oracle	ojdbc14.jar
DB2	db2jcc.jar
MySQL	mysql-connector-java-5.1.5-bin.jar
MSSQL	sqljdbc.jar

Table 2. JDBC Library Support Files (JARs)

Removing an Installation

Before removing the Oracle DC on a Windows system, be sure the application is not running as a Windows Service. Run the batch file script to remove the registration of the Oracle DC as a Windows Service, `dm_ucm_connector_uninstall_service.bat`.

Un-installation is accomplished by running the `uninstall.exe` application. On a Windows installation, a shortcut to the un-installation program is normally placed in the Start>All Programs>Oracle Documaker UCM Connector folder.

Basic Configuration

Archiving documents into UCM with the Oracle Documaker Connector is a cooperative process involving Documaker, the Oracle DC application and the UCM system. All three must be properly configured for the process to work reliably without problems. The most critical aspect of this configuration is the set of indexing metadata with which the documents are to be archived. As described in the earlier Planning section, you should already have decided which variable fields will be used as metadata. If you haven't completed that step, stop now and get that list together.

The indexing metadata must be supplied for each document by the Documaker process (generally as part of the incoming extract variable data for each document). Documaker must be configured to pass the desired data on to the Oracle DC so that it may, in turn, use it to archive each document. Configuring each step in the process so that they all agree on the indexing data from end to end requires care and planning.

The Oracle DC retrieves records from a database table which represent document copies produced by a Documaker batch process. You must configure Documaker to properly write these database records and the copies your documents. These document copies are intended to be imported into UCM. Oracle Documaker is usually configured using Batch Banner and Transaction Banner DAL scripting to control the output of the documents and to write the database records into the proper table. You must also configure the target UCM system with metadata fields appropriate to hold the desired indexing metadata. Lastly, you need to configure the Oracle DC so it may find the database table produced by the Documaker DAL scripting, access the documents referenced in the database table, connect to the desired UCM system and write the documents to that system to be archived.

In addition to these basic connectivity configuration requirements, the Oracle DC has configuration options for logging its operation and adjusting performance characteristics of the importing process which it manages. These settings allow the Oracle DC to be scaled from a small proof of concept to a large-scale production environment.

Oracle Documaker Configuration

Oracle DC reads a configured database table for rows containing processing columns, document metadata and an operating system path to each document file. It queries a database table using a select for update locking method for rows that have a status field indicating that the records have not been processed (STATUSCD column value of 0). It selects these rows with a maximum row count of as many records as it is configured to consider a maximum size **batch** and marks them with a status value (STATUSCD column) of 3 (in progress) and then processes each of those records.

Oracle Documaker must be configured by personnel with the necessary Documaker skill set to properly generate both the document files and the accompanying database records for the documents to be archived. The configuration of Documaker follows the configuration of UCM with the desired meta-data elements. Once this list of data elements is determined, Documaker can be configured to collect these data elements and store them in the associated Oracle DC table.

Oracle Documaker reads the variable transaction data for each document from a data extract input file. This file is supplied in either XML or CSV (Comma-Separated Value) format. To use the data elements in Documaker, the data extract (XML or CSV format) values are copied into Documaker **global variables** also known as GVM variables or GVMs using one of two methods: the TRN_FIELDS INI option may be

set or the EXT2GVM rule¹ may be used. Once the data are in GVM variables, you use DAL scripting² to control a recipient batch's output print stream name and map the meta-data in the GVM variables to database table columns. The Batch Banner and Transaction Banner processing DAL scripts insert the database rows.

Batch and Transaction Banner processing DAL is used since DAL scripts can be triggered for the different phases of output generation. The output print stream file name and location may be controlled as well as the post transaction processing step to map the GVM variables and any other static data to database table columns (using the DBPreVars DAL function) and finally insert database rows (using the DBAdd DAL function).

By default, UCM metadata maps to the intermediary database table by name but mapping of column names to UCM metadata can be done otherwise overriding the default name mapping method.

Making Index Data Available in Documaker GVM Variables

You can map Documaker extract input data which are in XML into GVM variables using the TRN_FIELDS INI option using XPATH notation. This requires setting up the mapping with XPATH declarations and modifying or creating the associated TRNDFDFL.DFD (Transaction) and RCBDFFDFL.DFD (Recipient Batch) Data Formation Definition files.

DFD files are used by the Documaker data storage abstraction interfaces. The data can be ASCII files, database, etc. Documaker pulls the data from extract files (XML or ASCII, etc) in batch processes and pushes them into field names defined in the DFD. This storage is then used as input to other batch processes down the process chain. Overall, these files control storage and propagation of the data through the batch system. See the "Documaker Server System Reference" documentation for configuration details. Full detailed examples of these files using a reference configuration may be found in Appendix B.

You can also use the EXT2GVM rule to map the data. See the Rules Reference for Oracle Documaker for more information on this rule.

Generating Database Table Rows and Writing the Document Files

You must create a database table and configure Documaker to both create the uniquely named recipient batch output print streams and to insert a row in the table with the name and location of the each output file. Each table row also contains the metadata fields needed for UCM ingestion. The Batch Banner and Transaction Banner processing DAL scripts are used to write the output documents and to insert the database table rows referencing these documents.

Batch Banner and Transaction Banner DAL processing each have a Begin and End phase. The phases are run in the following sequence with the actions indicated:

1. **BatchBannerBeginScript**
Configuration is loaded, a unique Job identifier is established, a related directory under the configuration-defined root location is created and the database connection is established.
2. **TransBannerBeginScript**
The batching folder unique name is defined and created and the first print stream name is defined.
3. **TransBannerEndScript**
The database table row is inserted with the desired metadata and a reference to the document.
4. **BatchBannerEndScript**
Clean-up is performed.

¹ See the "Documaker Server System Reference" documentation for configuration details.

² See the Documaker "DAL Reference" documentation.

Example INI Configuration

Enabling the Batch and Transaction Banner Scripts is done via the Oracle Documaker configuration INI file (for example, in FSISYS.INI or FSIUSER.INI). Typically the scripts are enabled for the chosen Recipient Batch output. In the example/reference implementation **Batch6** is the chosen output which is the FILE recipient's batch and is associated with Printer6. Here is an example configuration snippet with the changes and additions highlighted (**bold and yellow highlight**) to write PDF output for Batch6 and enable the DAL scripting calls:

```
< BatchingByRecip >
  Batch_Recip_Def      = TRUE; "BATCH1"; AGENT
  Batch_Recip_Def      = TRUE; "BATCH3"; INSURED
  Batch_Recip_Def      = TRUE; "BATCH2"; INSURED
  Batch_Recip_Def      = TRUE; "BATCH4"; CLAIMANT
  Batch_Recip_Def      = TRUE; "BATCH4"; OWNER
  Batch_Recip_Def      = TRUE; "BATCH4"; LIENHOLDER
  Batch_Recip_Def      = TRUE; "BATCH6"; FILE
  DefaultBatch         = ERROR

< Print_Batches >
  BATCH1               = data\agent.bch
  BATCH2               = data\insflat.bch
  BATCH3               = data\insured.bch
  BATCH4               = data\other.bch
  BATCH5               = data\lienholder.bch
  BATCH6               = data\file.bch

< Printer6 >
  Port                 = data\file.pdf
  PrtType              = PDF
  AORDebug             = No
  AORExt               = .pdf
  AORFilesPerBatch    = 1000
  AORPath              = c:\AOR\

; Enable a DAL library of scripts to be pre-loaded
< DalLibraries >
  LIB                  = aor

; Enable the Banner and Transaction DAL Scripting
< BATCH6 >
  EnableBatchBanner    = Yes
  EnableTransBanner    = Yes
  BatchBannerBeginScript = AOR_PREB
  BatchBannerEndScript  = AOR_POSTB
  TransBannerBeginScript = AOR_PRET
  TransBannerEndScript  = AOR_POSTT
  Printer              = Printer6
```

Example DAL Script

As indicated above in the < DalLibraries > section, **LIB** entry, Documaker is directed to look in an **aor.dal** file for the scripts listed under the < **Batch6** > section. **Aor.dal** is a library with the definitions of the BATCH6 DAL scripts shown in Appendix C:

This DAL scripting requires Oracle Documaker RP to be configured to connect to a database and for the referenced table to pre-exist in that database. The table schema must agree with what the DAL will insert into that table. This is done by first creating the database table in the target database, then setting up the Documaker RP INI configuration for the connection to that database with the table. Finally, the Documaker DFD (Data Format Definition) must be set up containing all these field values that are to be mapped from GVM variables to the database table.

Example Database Table Definition

Each row of the table must include a minimum set of columns required by both Documaker and the Oracle DC to manage the list of output documents and the document output directory. The table schema must also include additional columns of indexing data used to archive the documents with the desired set of keys and metadata. The table below indicates the minimum required columns of the table, with their *default* column names. These column names are customizable in the Documaker Source configuration file. DDL to create the table below can be found in Appendix D.

Column	Type	Description
JOBID	VARCHAR(50)	The Documaker globally-unique job identifier which identifies a grouping of one or more Documaker transactions for a single import run. Imports (XML, V2) can contain one or more transactions. This column's value is also used by default for the root directory folder of the output print stream.
TRANID	VARCHAR(50)	The Documaker transaction identifier for a transaction with one or more Documaker batches (recipient batches). This column's value is also in some implementations to map to the Document Title in UCM and for searching should identify the document type and purpose.
BATCHID	VARCHAR (50)	The Documaker batch identifier for a document, usually the same name as the recipient batch plus a counter. For example, BATCH6x2 where the counter is incremented as the specified maximum number of files per folder is reached. This columns value is also used by default to segment the transaction folder into sub-folders for each group of output files.
DOCID	VARCHAR (50)	The globally-unique document identifier.
NAME	VARCHAR (30)	The name of the document.
TYPE	VARCHAR (30)	The type of document.
TITLE	VARCHAR (255)	The title for the document.
AUTHOR	VARCHAR (50)	The author or owner of the document.
SECGROUP	VARCHAR (30)	The security group assigned to the document.
PFILE	VARCHAR (255)	A file URL or path to the document so that it can be imported into UCM.
STATUSCD	INTEGER DEFAULT 0	This column contains the status of a document. The following values are supported: 0 – Not yet processed by Oracle DC (“ new ”) 1 – Imported into UCM (“ success ”) 2 – Import failed (“ failure ”) 3 – In process by the Oracle DC (“ in progress ”)
STARTTIME	TIMESTAMP	A time stamp that indicates at which time the document import process started. This column is updated by the Oracle DC.
ENDTIME	TIMESTAMP	A time stamp that indicates at which time the document import process ended. This column is updated by the Oracle DC.
RESULTDESC	VARCHAR (2000)	A description of the outcome of the import process; updated by the Oracle DC at the time of import. This column will have a blank value if the document import process is successful. Otherwise, it will contain a description of the error.
RETENTION	TIMESTAMP	A time stamp that indicates when the document expires and can be removed from the table. This value is updated by the Oracle DC upon successful import based on the value of the <code>platform.retention.time</code> configuration property which indicates the number of days after import when the document expires.

Table 3. Minimum DAL Output Database Table

Documaker INI Setup for the Example Database Connection

Oracle Documaker is configured to access a database and a particular table in the Oracle Documaker configuration INI file (either `FSISYS.INI` or `FSIUSER.INI`). Documaker uses the Open DataBase

Connectivity (ODBC) interface API to the database. Examples of the INI file settings for this connection are shown below. In the example settings, Documaker is configured to connect via ODBC to a database server called OracleXE10g with an encrypted UserID and password. In this database, Documaker is configured below to use a table called AOR (as in “Archive of Record”.) Details on ODBC database access configuration are in the Documaker documentation.

```

; Database connection info
< DBHandler:ODBC >
    Class           = ODBC
    Server          = OracleXE10g
    ;SubClass       = ORA
    CreateTable     = No
    CreateIndex     = No
    ;Debug         = Yes
    UserID          = ~ENCRYPTED 1-S6rx_NR_wt2hsjXScy0
    PassWd         = ~ENCRYPTED 1-S6rx_NR_wt2hsjXScy0

; Database Table reference, this case a table named AOR
< DBTable:AOR >
    DBHandler       = ODBC

```

Oracle UCM Configuration

This section is not a substitute for the UCM documentation or familiarity with the UCM configuration requirements, procedures and user interfaces. The information here is primarily focused on explaining the minimum required UCM set up through once set of interfaces and to provide some example information to assure a knowledgeable UCM administrator has a few reference points to understand the requirements in this document.

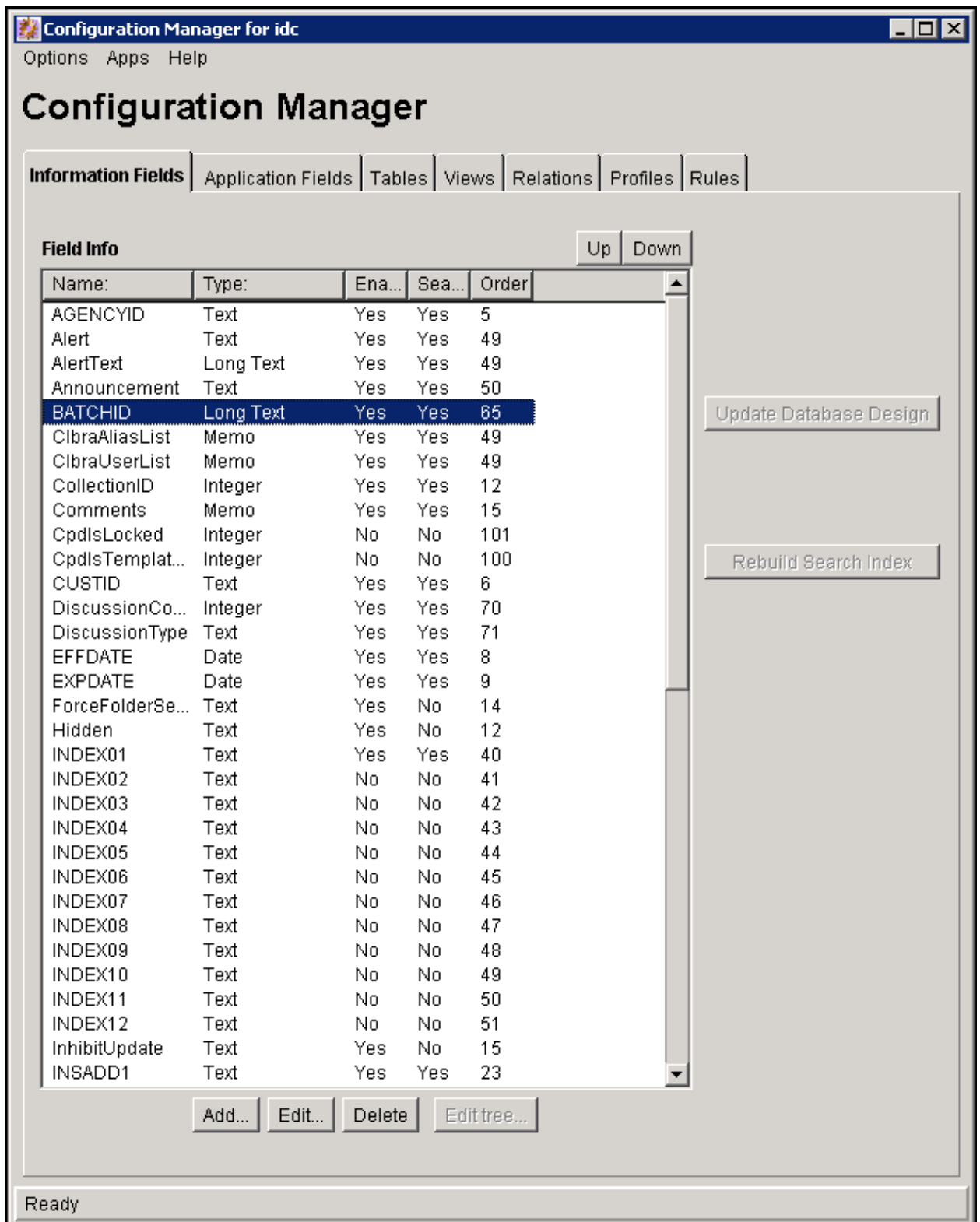
A minimal UCM system for use with the Oracle DC starts with installation of the Oracle Content Server. Development of the Oracle DC used Oracle Content Server 10gR3.

The metadata and documents you capture in Documaker and pass through the Oracle DC must have homes in your UCM system. You must set up your UCM system with the fields needed to hold your incoming metadata. You must also make sure there is plenty of space to store the documents you are sending in via Oracle DC. UCM allows metadata fields to be either required or optional. The Oracle DC pre-flights the fields provided in the database records to assure at least the fields required in your UCM configuration are present. If the table is missing a required field definition, the Oracle DC will log this information and refuse to finish starting up.

By default, the names of the columns in the incoming table from Documaker are used to match the data to UCM property (field) names. Not all the table column names need be used in UCM. Many of the table columns are used internally by Documaker and the Documaker Source modules to manage the table itself, remove old records which have been processed and to keep track of incoming documents before they are processed into UCM.

Within UCM, use the Configuration Manager interface to establish the UCM **Information Fields** for the incoming table columns you want to capture. In the example below, Information Fields have been established for these example fields coming in from Documaker: AGENCYID, BATCHID, CUSTID, EFFDATE, EXPDATE, INDEX01, INDEX02, ..., INDEX12, INSADD1, INSADD2, INSCITY, INSDOB, INSFNAME, INSLNAME, INSPHONE, INSSSTATE, INSZIP, JOBID, KEY1, KEY2, KEYID, POLNUM, RUNDATE, TRANCODE, TRANID

Of the columns shown above in Table 3, these columns are mapped by their default names to the indicated UCM metadata fields: JOBID, TRANID, and BATCHID. These fields are shown in the figure below.



Oracle Documaker Connector Configuration

Oracle DC is configured using the command line parameters and these configuration files:

- The logging properties file
- The engine properties file
- The source system properties file

You can enter the names of these properties files on the command line.

The Logging Properties File

This properties file is specified by the `-loginfo` command line parameter. If the parameter is not specified on the command line, the log messages are written to the console (standard out) in the following default format:

```
<number of milliseconds since program start> [ <current thread name> ] <message priority>  
<message category> <any nested diagnostic context> - <the log message>
```

The Connector uses the Log4J logging framework and will accept a properties-file containing configuration values appropriate to Log4J. This section provides information on configuration values defined by the Oracle Documaker Connector. However, it is not an exhaustive reference on the Log4J capability or configuration options. Please see the Log4J documentation for more details.

At startup, the code queries the `-Dlog4j.configuration` property which should point to a valid log4j XML configuration file. If this parameter does not exist, then the default configuration is applied (as defined by the BasicConfigurator).

Here is an example of a command line property:

```
-Dlog4j.configuration=log4j.xml
```

Here is an example of a configuration file, such as the file referenced above, `log4j.xml`:

```
<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">  
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/" debug="false">  
  
  <appender name="roll-stderr" class="org.apache.log4j.RollingFileAppender">  
    <param name="File" value="ucmimporter-stderr.txt"/>  
    <param name="Encoding" value="ISO-8859-1"/>  
    <param name="maxFileSize" value="100KB" />  
    <param name="maxBackupIndex" value="5" />  
    <layout class="org.apache.log4j.PatternLayout">  
      <param name="ConversionPattern" value="rse %d{ISO8601} %-5p [%t] - %c-%m\r\n"/>  
    </layout>  
  </appender>  
  
  <appender name="roll-stdout" class="org.apache.log4j.RollingFileAppender">  
    <param name="File" value="ucmimporter-stdout.txt"/>  
    <param name="Encoding" value="ISO-8859-1"/>  
    <param name="maxFileSize" value="100KB" />  
    <param name="maxBackupIndex" value="5" />  
    <layout class="org.apache.log4j.PatternLayout">  
      <param name="ConversionPattern" value="rso %d{ISO8601} %-5p [%t] - %c-%m\r\n"/>  
    </layout>  
  </appender>
```

```

<category name="com.oracle.documaker.ucmimporterlib.source.DocumakerSource"
  additivity="false">
  <priority value="error"/>
  <appender-ref ref="roll-stderr"/>
</category>

<category name="com.oracle.documaker.ucmimporterlib.db.DBUtil" additivity="false">
  <priority value="error"/>
  <appender-ref ref="roll-stderr"/>
</category>

<category name="org.apache.ddlutils.util" additivity="false">
  <priority value="error"/>
  <appender-ref ref="roll-stderr"/>
</category>

<category name="com.oracle.documaker.ucmimporterlib.housekeeping.HouseKeeperSingleton"
  additivity="false">
  <priority value="error"/>
  <appender-ref ref="roll-stderr"/>
</category>

<category name="com.oracle.documaker.ucmimporter.config" additivity="false">
  <priority value="error"/>
  <appender-ref ref="roll-stderr"/>
</category>

<category name="com.oracle.documaker.ucmimporter" additivity="false">
  <priority value="error"/>
  <appender-ref ref="roll-stderr"/>
</category>

<category name="oracle.stellent.ridc.protocol.intradoc" additivity="false">
  <priority value="info"/>
  <appender-ref ref="roll-stderr"/>
</category>

<root>
  <priority value="debug"/>
  <appender-ref ref="roll-stderr"/>
</root>

</log4j:configuration>

```

Examining this example file, we see that it contains seven categories. Categories are used to enable and/or disable logging at the class (or component) level. For example, the `com.oracle.documaker.ucmimporterlib.source.DocumakerSource` category is used to enable or disable logging for the `DocumakerSource` class. Changing the `priority` value to `debug` instead of `error` will enable logging of debug and error statements, while leaving the value set to `error` will only log errors.

There are several priority values available in log4j, sorted in the table below from most restricted (least logging overhead) to least restrictive (most logging overhead):

Logging Priority Value	Description
Fatal	Logs Fatal statements.
Error	Logs Error and Fatal statements.
Warn	Logs Warn, Error and Fatal statements.
Info	Logs Info, Warn, Error and Fatal statements.
Debug	Logs Debug, Info, Warn, Error and Fatal statements.

Table 4. log4j Logging Priorities

Category definitions also reference one or more **appenders**. These appenders are defined earlier in the property file. In this example file, two appenders are defined: `roll-stderr` and `roll-stdout`. Category `com.oracle.documaker.ucmimporterlib.source.DocumakerSource`, for example, references the appender (`appender-ref`) named `roll-stderr`.

Each appender defines a component that will append a new log record to the end of a given log destination, in a particular way. For example, the first appender element named `roll-stderr` will direct the log record output to a file named `ucmimporter-stderr.txt`. The other parameters for the appender define exactly how the appender translates and formats the log data before writing it to the destination. In the case of this file, a parameter (`maxFileSize`) also places a maximum size on the destination file beyond which it will be closed, renamed and a new file of the same name started.

You can use these categories to debug the `DocumakerSource` implementation:

- `com.oracle.documaker.ucmimporterlib.source.DocumakerSource`
- `com.oracle.documaker.ucmimporterlib.db.DBUtil`
- `org.apache.ddlutils.util`

The following categories can be used to debug the `HouseKeeperSingleton` implementation:

- `com.oracle.documaker.ucmimporterlib.housekeeping.HouseKeeperSingleton`
- `com.oracle.documaker.ucmimporterlib.db.DBUtil`
- `org.apache.ddlutils.util`

The following categories can be used to debug the `Connector`:

- `com.oracle.documaker.ucmimporter.config`
- `com.oracle.documaker.ucmimporter`
- `oracle.stellent.ridc.protocol.intradoc`

Log4j is very configurable and flexible. Please refer to the Apache documentation for the Log4J project to learn more.

The Engine Properties File

This properties file is specified by the `-config` command line parameter and contains the configuration properties for the Connector's engine. Of special interest, this properties file also contains the credentials

used to connect to UCM. This file can contain any configuration data except the configuration file path (the path to itself) and the log configuration file path.

All of these properties can also be overridden by command line parameters of the same name (simply prepend a dash-character (-) to the name in the table below). There is therefore an order to the sources of these parameters: default values (if any) are overridden by the contents of the properties file. If the property is also specified on the command line, that value takes the highest precedence, overriding both the default and file contents values for the property.

Also in this file are source and client class names, various timeouts, and a number of other flags and/or commands.

<i>UCM Credential & Connection (Repository) Properties</i>		
Property Name	Description	Default
ucmpassword	The password for connections to the content management system.	
ucmuser	The user name for connections to the content management system (UCM in this case).	
connectionstring_#	Possible connection strings used by the Client implementation. Each of these has a number appended to it signifying its priority ('0' being the highest).	
<i>General Configuration Properties</i>		
Property Name	Description	Default
batchmode	This flag tells the Connector to run in batch mode (see "Modes of Operation").	false
config	Name of the core Connector application configuration file. This is the file in which these properties (in this table) may be used.	null
deletefiles	If true, Oracle DC will delete content files after they are successfully imported.	false
loginfo	Name of the log4j properties file with the logging configuration.	null
sourceconfig	Name of the properties file holding the configuration of the Documaker Source.	null
<i>Performance Tuning Properties</i>		
Property Name	Description	Default
housekeepingwait	The wait time between calls to the source system's cleanup functionality.	60
maxrecords	Maximum number of records that should be retrieved from the document source for processing before processing them.	10
sourcecount	Specifies the number of concurrent connections to the document source system.	1
sourcewait	Number of seconds a processing thread pauses when no new documents are available from the source system.	2
threadcount	Number of processing threads within the engine that will import documents into the repository.	
<i>Server / Command Mode Configuration Properties</i>		
Property Name	Description	Default
hostName	When running in client mode (see "Modes of Operation"), this property specifies the Connector instance to which commands are sent.	null
port	The command channel port number which a server instance of the Connector opens for listening and to which a client instance will connect and send commands.	23232
password	Guards access to the command channel in a server instance. The	

	password specified to the server instance must be provided by a client during each command request.	
shutdown	If present, the Oracle DC instance runs in <i>command</i> mode and sends a shutdown command to another Oracle DC instance running in server mode.	
<i>Product Development Customization Properties (Should usually not be changed)</i>		
Property Name	Description	Default
clientadminname	The Java class name of the <i>ClientAdministration</i> implementation the Connector will use to communicate with the document repository for non-import specific functions. Oracle DC installs and uses the UCM client admin interface.	
clientname	The Java class name of the <i>ClientInterface</i> implementation the Connector will use to communicate with the document repository. Oracle DC installs and uses the UCM client interface.	
source	The name of the <i>Source</i> interface implementation that the Connector will use to communicate with the document provider. Oracle DC installs and uses the Documaker Source.	

Table 5. Engine Properties

The Documaker Source System Properties File

This properties file is specified by the `-sourceconfig` command line parameter. Each of these properties is passed to the Documaker Source component. The configuration properties listed in the table below are supported. Every property name begins with the prefix `platform.`. For the sake of brevity, this prefix is not repeated in the table below.

This property file is shared between the two principal programming interfaces. This table is referenced in the later section for developers and the distinction is important there. Not all the properties apply to both interfaces, but many of them do. In the table below, the properties that apply to both interfaces are not colored. Those that apply only to the Source implementation are in **green** and those applying only to the Housekeeping interface are in **blue**.

Property Name (without 'platform.' prefix)	Description
<i>Basic Configuration Properties (Required)</i>	
check.length	Set value: true or false . The default value is false . Configures whether or not the Oracle DC should validate the length of the data retrieved for each column before passing that data back on to UCM. When enabled via a value of true or yes, the length of the data retrieved is checked against the length of the UCM metadata for that column and if the length of the data exceeds the length of the UCM metadata, a <code>LengthCheckException</code> is thrown.
retention.time	Indicates how long records should remain in the table once their STATUSCD is set to the <i>imported/success</i> value of 1. The default value is 7 (days).
security.group	Configures the default security group for a UCM <code>FileData</code> object.
<i>Database Connection Properties (Required)</i>	
table	Sets the JDBC database table name.
timeout.seconds	Configures the timeout in seconds for JDBC connections. The default

	value is 15 (seconds).
fetch.size	Configures how many records are read and processed at one time. The default is 10 .
url	The JDBC database URL.
driver	The JDBC driver name to use.
principal	The JDBC account name used for authentication to the database.
credentials	The JDBC password used for authentication to the database.
schema	The table schema. (Oracle databases only.)
version	The Database version. (Oracle databases only.) Acceptable values are Oracle8 , Oracle9 , Oracle10 , or Oracle11 .
catalog	The table catalog. (Oracle databases only.)
Database Table Customization Properties (Optional)	
system.fields	Maps the default system field names to other table column names. The default system field names are: BATCHID, DOCID, NAME, TYPE, TITLE, AUTHOR, SECGROUP, PFILE, STARTTIME, ENDTIME, RESULTDESC, STATUSCD, and RETENTION. The format of the value for this property is a comma-separated list of name-value pairs: systemfieldname=tablecolumnname,.... Here is an example: DOCID=FID, NAME=DOCNAME, BATCHID=TRANID Some of these fields may also be mapped to new names with other specific properties described in this table.
filter	Maps the STATUSCD system field to a different table column name.
filter.value	Maps the filter value of the STATUSCD system field to a value other than the default value of 0 .
filter.update.value	Maps the filter update value of the STATUSCD system field to a value other than the default value of 3 .
retention.filter	Maps the RETENTION system field to a different table column name.
Database Connection Maintenance Properties (Should usually not be changed)	
max.active.connections	Maximum active connections in the connection pool The default is 50 .
max.idle.connections	Maximum idle connections in the connection pool. The default is 25 .
min.idle.connections	Minimum idle connections in the connection pool. The default is 15 .
max.wait.time.seconds	Maximum number of seconds to wait for a connection from the connection pool. The default is 30 .
pool.initial.size	Initial connection pool size. The default is 25 .
max.open.prepared.statements	Maximum number of opened prepared statements to keep for the connection pool. The default is 15 .
test.while.idle	Whether to test connection objects while idle. The default is true .
test.on.borrow	Whether to test connection objects when obtaining a connection. The default is true .
test.query.string	The query string to use for testing connections. The default is 'select 1 from tableName' . (See table above.)
min.eviction.idle.time.millis	How long connections should be idle before they are cleaned up. The default is 600000 or 10 minutes.
time.between.eviction.runs.millis	How often should cleanup of idle connections be performed? The default is 300000 or 5 minutes.

<code>tests.per.eviction.run</code>	How many connections to test each run. The default is 25.
<i>Database Connection Maintenance Properties (Should usually not be changed)</i>	
<code>housekeeping.root</code>	A comma-separated list of top-level file system directories that should be cleaned. All directories within each of these directories will be scanned by the housekeeping process.
<code>housekeeping.transact.file</code>	The name of a file which if present in a directory indicates the processing of the directory contents is complete and it is therefore ready for clean-up. The default name is <code>transact.dat</code> .

Table 6. Documaker Source Properties

Running the Program

The Oracle DC can be run from the command line or as a Windows service. Command line execution is generally done with a script, since the Java command line is so complex. The basic command line format is:

```
java -cp classpath mainclass parameters
```

The *classpath* is installation dependent. Generally, it will be something like

```
log4j-1.2.8.jar:/Users/gene/Development/Java/Tools/oracle-ridc-client-11g.jar:/Users/gene/Development/Java/Tools/commons-codec/commons-codec.jar:/Users/gene/Development/Java/Tools/commons-httpclient-3.1/commons-httpclient-3.1.jar:/Users/gene/Development/Java/Tools/commons-logging-1.1.1/commons-logging-1.1.1.jar:target/UCMImporter.jar:../ucmimporterlib/target/UCMImporterLib.jar
```

mainclass is always `com.oracle.documaker.ucmimporter.Driver`

The command-line *parameters* might be something like this (see below for detail):

```
-logininfo log4j.properties -config ucmimporter.properties -sourceconfig dmkrsrc.properties
```

Together, the overall command-line to run the program might look like this:

```
java -cp log4j-1.2.8.jar:/Users/gene/Development/Java/Tools/oracle-ridc-client-11g.jar:/Users/gene/Development/Java/Tools/commons-codec/commons-codec.jar:/Users/gene/Development/Java/Tools/commons-httpclient-3.1/commons-httpclient-3.1.jar:/Users/gene/Development/Java/Tools/commons-logging-1.1.1/commons-logging-1.1.1.jar:target/UCMImporter.jar:../ucmimporterlib/target/UCMImporterLib.jar com.oracle.documaker.ucmimporter.Driver -logininfo log4j.properties -config ucmimporter.properties -sourceconfig filesource.properties
```

Running the Oracle DC as a Windows service is accomplished via the Windows Service Control Manager.

Modes of Operation

The Oracle DC has three modes of operation. The mode is selected based on the command-line and configuration file option settings:

1. **Server or Normal continuous mode** – Oracle DC runs until it receives a shutdown command, polling the Source instances for documents to process. To receive control commands, Oracle DC opens a TCP/IP socket and listens for incoming messages. The port number may be controlled by a configuration parameter and a *password* may be established which must be supplied along with any commands.

2. **Batch or One-Shot mode** – In this mode, Oracle DC creates the configured number of Documaker Source instances, calls each one once to fetch and process a single batch of transactions (if any are available) and then terminates. No socket is opened for commands and there is no subsequent polling of the Documaker Source instances. In the case of the Oracle DC, this would allow a single pass through any records in the database table for each configured Source instance.

This is expected to be used primarily for a non-persistent, static document source, rather than with the Oracle Documaker Connector. An example of such a source would be one which read a named input text file. Such a source has no means to continually receive new documents, so once the configured source of documents (such as the single named input file) is exhausted, there is no purpose in having the Connector application continue to run. This would function similarly to the UCM BatchLoader application.

3. **Command mode** – Oracle DC may also be run solely to send a command to another copy of Oracle DC running in *server* mode (above). If a command is specified as a configuration option (normally on the command line, but could be in the configuration file), then Oracle DC runs in this mode, sends the command to the designated hostname and port with the supplied password, if any, and immediately terminates.

How Oracle DC Determines the Run-Mode

When first run, Oracle DC processes the configuration file and the command line and then examines the `batchmode` parameter value. If set **true**, Oracle DC operates for the current run in *batch* mode. If `batchmode` is not specified or **false**, Oracle DC looks for a command keyword in the parameters. The only valid command keyword is **shutdown**. If present, this run will be in *command* mode and will send a shutdown command to the copy of Oracle DC defined by the parameters:

Stopping a Server-Mode Instance

If an Oracle DC is running in Server mode on the local machine with the default port number and a configured password of `boogie`, then it may be stopped by running the Oracle DC again with this command line:

```
java -cp log4j-1.2.8.jar:/Users/gene/Development/Java/Tools/oracle-ridc-client-11g.jar:/Users/gene/Development/Java/Tools/commons-codec/commons-codec.jar:/Users/gene/Development/Java/Tools/commons-httpclient-3.1/commons-httpclient-3.1.jar:/Users/gene/Development/Java/Tools/commons-logging-1.1.1/commons-logging-1.1.1.jar:target/UCMImporter.jar:../ucmimporterlib/target/UCMImporterLib.jar com.oracle.documaker.ucmimporter.Driver -loginfo log4j.properties -shutdown -password boogie
```

Command-line Parameters Detail

The Oracle DC command-line *parameters* are given in the following format:

```
-property [value] -property [value]...
```

That is, properties to be set on the command line are listed separated by spaces and each property name is prefaced with a dash (or *hyphen* character.) If the property requires a value to be set, the value is placed immediately after the property name and a space. The only property not requiring a value is the command property, `shutdown`.

Examples

Some examples of command-line *parameters* portions:

```
-loginfo log4j.properties -config odc.properties -sourceconfig odm.properties
```

This example would be used to run Oracle DC in *normal* mode, with log settings coming from the file `log4j.properties`, the Connector settings coming from the file `odc.properties` and the Documaker Source configuration coming from the file `odm.properties`. The program would open a socket on the default port and accept any shutdown command set to it.

```
-loginfo log4j.properties -password boogie
```

This example would be used to run Oracle DC in *normal* mode, with log settings coming from the file `log4j.properties`. The program would open a socket on the default port, but would only accept shutdown commands sent to it with the proper password of `boogie`.

```
-loginfo log4j.properties -password boogie -shutdown
```

This example would run Oracle DC in *command* mode, with log settings coming from the file `log4j.properties`. The program would connect to another Oracle DC instance running on the same computer via a socket on the default port and would then send the shutdown command with the password `boogie`.

Customization for Developers

The Oracle Documaker Connector is built on a generic framework for building document archiving or transmission services. The core of the Connector is a multi-threaded Java application that is highly configurable and customizable (via new implementations of the source and client interfaces). This section is written for programmers who wish to utilize the Connector framework and create a new application which can either draw documents from a source other than Oracle Documaker and/or send those documents to a repository interface other than UCM. This section, therefore gets under the covers of the Oracle DC, describes how it is implemented and discusses the Oracle DC as a tool framework.

Architecture

When the core Connector Java application is run, it dynamically loads Java classes which provide the interfaces to both the Source of incoming documents or files as well as the Client code which submits these files for forwarding or storage into a repository system such as Oracle UCM.

The main application thread runs in the core Connector application and that code drives both the dynamically-loaded classes by calling out to them for services. The application polls the Source for incoming documents (on threads which may block until a document is available) and then pushes each document to the Client side when it is available. Status is relayed by the Connector between Client-side and Source-side so that appropriate chain-of-custody tracking can be implemented.

The Source class and the Client class must each be configured by name to the core Connector application. There can be only one of each (named) class, so if multiple, different sources are to be polled for incoming documents, multiple instances of the Connector application must be configured and run separately. However, within each Connector application instance, many instances of each class may be instantiated by the Connector application and run on separate threads to provide scalability and better performance.

The Connector application makes available both configuration file and logging services to the source and client class implementations, however these classes (and their supporting classes) are free to do what they like with respect to configuration and logging.

One of the core Connector configurations is the number of concurrent processing threads to use so the burden of starting and managing multiple processing threads which call both the Source and Client code is shouldered by the core Connector application. The configured Source and Client should not start additional threads for this purpose.

Class Structure

The Connector is made up of three parts: the **source implementation**, the **engine**, and the **client adapter implementation**. The Connector defines the **engine** functionality and the interfaces to both the **source** and **client** implementation classes. Consequently, the Connector provides the translation layer between independent implementations of document or file sources and client system adapters.

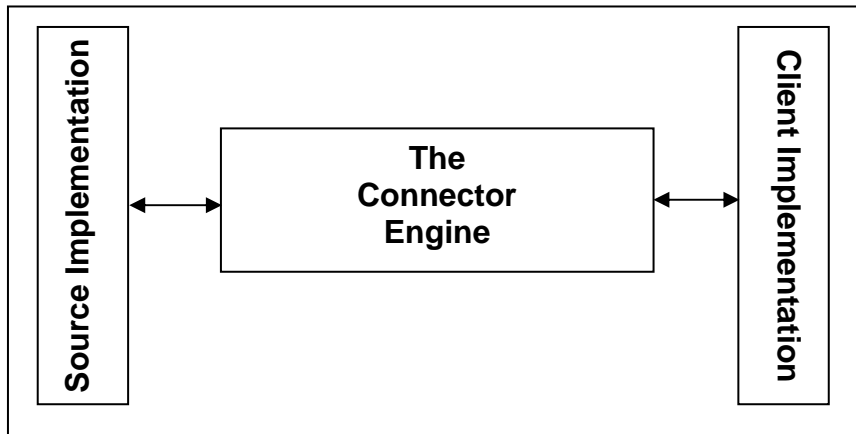


Figure 1 – The Connector Structure

Source Implementation

This component provides the specific functionality for interacting with the document source by implementing the Source interface. It is responsible for all communication between the Connector and the document source, for validating the document meta-data information, and for any housekeeping functionality needed to manage the document source.

The Connector Engine calls the Source for two primary purposes. First, the Source provides a *housekeeping* class that is called with a periodic *heartbeat* thread. This processing should not block for long periods of time. Second, the Source is called for document processing on a configurable number of threads, each of which may block while waiting for incoming documents.

The Connector Engine

The Connector Engine is the primary controller for the service, contains the main application thread and starts all the additional housekeeping and payload work threads. It creates instances of the Source and Client implementations (based on configuration data) and drives the Source-to-Client process.

Client Implementation

This component provides the specific functionality for interacting with the document repository (or other configured service). This includes managing connections, sending requests and processing results/errors.

The Client is called, as is the Source, for both periodic housekeeping functions and on a variable number of *payload* threads that are actually performing document transfer operations.

Functional Breakdown

The following sections describe the three principal areas of application functionality as implemented by the Connector:

1. configuration
2. processing
3. shutdown

Configuration

This functionality is executed both when the application begins and when a change is detected in the specified configuration file (by monitoring its last modified date). It consists of:

1. initializing the logging framework,
2. parsing the configuration file,
3. processing any additional command-line arguments, and
4. (re-)initializing the connector.

Initializing the Logging Framework

The Connector uses the Log4j logging library. At startup, it queries the command-line for the `-loginfo` parameter that specifies the library's configuration properties. If this parameter does not exist, then the default configuration is applied (as defined by the BasicConfigurator).

Parsing the Configuration File

The order of precedence (highest to lowest) for configuration data is command-line, configuration file, and default values. The configuration file (as specified by the `-config` command-line parameter) is parsed and the values found are stored for later usage. This file can contain any configuration data except the configuration file path and the log configuration file path.

Command-line Processing

The last step of determining the configuration data is to process the remaining elements of the command-line. These values overwrite any defaults or those found in the configuration file.

Initializing the Connector

The actions that occur, in order, during the initialization of the Connector:

- The configuration watch thread is created. Every 15 seconds, this thread looks for a change in the configuration file. If it finds one, it sends a reset message to the Connector's engine. This message doesn't interrupt any ongoing processing. It is held until the Connector is otherwise idle. Processing this message sends the Connector back through this initialization process.
- The command channel and thread is initialized. This thread monitors the command socket for shutdown requests that may or may not be password protected.
- The client service is checked for availability. A client implementation is instantiated and a *ping* request is issued every second until a valid response is returned. Until this is so, no further initialization will occur, as a valid document repository is necessary for the Connector to function properly.
- The **source pool** is created and initialized. This pool contains the configured count of the configured source objects.
- A single housekeeping thread is created with a housekeeping object acquired from a source object. This thread calls the housekeeping object's `cleanUp` method once every configurable number of seconds to allow the source system to be correctly maintained. This housekeeping functionality is source system dependent. The client-side object(s) have no analogous functionality, as those systems should maintain their own states. Note also that this object is optional; a source that returns no housekeeping object will cause this initialization step to be skipped.

- The batch management thread is started. This process along with its child threads and functionality are the engine of the Connector. This thread receives its termination notice from the main application thread.
- Finally, the main application thread begins waiting for the command channel thread to terminate signaling either the reset or termination of the application.

At this point, the Connector simply waits for notification from the command channel that a shutdown request has been received.

Processing

The engine component provides the Documaker Connector processing functionality. This component is centered on the BatchManager and BatchProcessor objects.

The BatchManager

The BatchManager controls the lifecycles of the configured number of BatchProcessor threads. On startup, it creates and initializes each of these threads. Then, while waiting for the shutdown command, every 10 seconds it directs the source pool to revalidate any failed source objects. Upon receiving a shutdown command, the BatchManager messages each of its BatchProcessor threads with a shutdown command. When all of these child threads have terminated, the BatchManager thread itself terminates allowing the main application thread to continue its cleanup.

The BatchProcessors

These threads/objects are responsible for the actual importing of documents into the document repository (UCM). The steps they follow to achieve this are:

- Acquire a source object from the source pool. This will be maintained until the end of the import attempt.
- Acquire a file data batch from the source object. This is a two-step process. First, a source object is acquired from the source pool. The BatchProcessor requests a source object and, if one is available, uses that for step two. If one is not available, the BatchProcessor waits 250 milliseconds and asks again. In step two, the BatchProcessor queries the source object for a file data batch. It is up to the implementation of the source object as to whether it blocks or not, but the recommended action is to either return an empty list or null if no file data objects are available. In this case, the BatchProcessor will wait the configured number of seconds, and ask again. During both step one and two, if a shutdown message is received, the BatchProcessor terminates without making additional requests.
- For each file data object in the batch, call the client implementation's `importFile` method, and (upon successful completion) if the `deletefiles` configuration value is set, delete the source file.
- Call the `ackProcess` function on the source object so that it may update the source system with the results of the import attempts. This method processes the entire batch at once and need not be called for each document.
- Return the source to the pool.

These steps are repeated until the BatchProcessor receives a shutdown message.

Shutdown

This is the simplest part of the application. The Connector sends shutdown messages to all the worker threads and waits for them to terminate before exiting.

Core Engine Interfaces

The Connector Engine manages two systems: the Source system and the Client system. Each of these defines the interfaces necessary to administrate and use the particular system.

The Source System

The two main interfaces of the Source System are the Source interface that allows the engine to communicate with the document source (like Documaker) and the Housekeeping interface for the management/maintenance of the document source.

Overview

Implementations of this interface provide access to the document source. The interface's methods are broken up into three types: configuration and maintenance, main functionality, and interface health.

Since the engine cannot know a priori the specific configuration and management functions needed for each possible source, this interface provides entry points for an implementation's version of each of these. The `configure` and `getHouseKeeper` functions are the first to be called (`getHouseKeeper` is only called on one instance) supplying any source specific setup data needed and to retrieve the maintenance interface. Additionally, the `setMaxBatchSize` method is called to limit the number of records returned from the source. This helps balance the document load over the available processing threads.

The functional entry points are `getFileBatch`, which returns a list of `FileData` objects describing the documents to be imported, and `ackProcess` which accepts a list of `FileData` objects that contain the results of their import attempts.

The `isValid` entry point allows the engine to monitor the health of a particular instance. If a Source instance generates an error during normal processing, it is marked invalid and set aside. Once every 10 seconds, the Connector calls the `isValid` method on these instanced giving them a chance to recover from their previous error state. If they return true, they are re-added to the available pool and may be used by subsequent batch processor requests.

The Source Interface in Java

```
package com.oracle.documaker.ucmimporterlib;

import java.util.List;
import java.util.Properties;

import com.oracle.documaker.ucmimporterlib.exceptions.SourceException;

/**
 * Source is the interface that must be implemented by any class provided in the -
 * input parameter of the ucmimporter application.
 *
 * @since 1.0
 */
public interface Source {
    /**
     * Configures the file data source with the provided properties. These will be
     * specific to the source type and will have no pre-processing done before
     * this call is made.
     *
     * @param configData      The configuration data needed by this source type
     * @param customProperties The list of custom properties that may be associated
```

```

with each file
 * @param securityGroups The list of possible security groups that may be
assigned to each file upon import
 * @throws SourceException If a configuration error occurs during the
initialization of this file data source
 * @see MetadataDefinition
 */
public void configure(Properties configData, List<MetadataDefinition>
customProperties, List<String> securityGroups) throws SourceException;

/**
 * Returns the Housekeeping interface to the object that is to maintain the file
data source.
 *
 * @return The Housekeeping implementation or null if no housekeeping is necessary.
 */
public Housekeeping getHousekeeper();

/**
 * Returns a list of FileData objects that will be processed by the ucmimporter.
Each FileData object contains the necessary data for import
 * into the Oracle Universal Content Manager.
 *
 * @return A List of FileData objects or null if none or available
 * @throws SourceException If the file data source cannot generate a list (i.e. it
has been closed)
 * @see FileData
 */
public List<FileData> getFileBatch() throws SourceException;

/**
 * Processes the FileData list results. This may update/delete rows in a database
table and/or retry failures.
 *
 * @param fileData A list of FileData objects that has been processed by the
ucmimporter application
 * @throws SourceException If an error occurs during processing that invalidated
this instance.
 * @see FileData
 */
public void ackProcess(List<FileData> fileData) throws SourceException;

/**
 * Determines if the current source object is valid and able to process requests
 *
 * @return True if the current source can process requests, false otherwise
 */
public boolean isValid();

/**
 * Sets the maximum size of the list FileData objects that can be returned by the
getFileBatch method. Unless specified by the ucmimporter
 * application, this value is implementation specific. Note that this field can
change during runtime (possibly due to performance issues).
 *
 * @param maxBatchSize the maximum number of FileData objects returnable from
getFileBatch
 */
public void setMaxBatchSize(int maxBatchSize);

/**
 * Closes the current Source object and cancels any getFileBatch or ackProcess
calls in process.

```

```
*/  
public void close();  
}
```

The Housekeeping Interface Overview

The Housekeeping interface has just two entry points: `cleanup`, called periodically to give the document generation service a chance to do any maintenance, and `close`, called when the Connector is shutting down.

The Housekeeping Interface in Java

```
package com.oracle.documaker.ucmimporterlib;  
  
import com.oracle.documaker.ucmimporterlib.exceptions.HousekeepingException;  
  
/**  
 * Housekeeping is the interface that must be implemented by any class that will  
 * provide Source housekeeping functionality. This class will  
 * be returned by the getHousekeeper method of the Source interface.  
 *  
 * @since 1.0  
 */  
public interface Housekeeping {  
    /**  
     * Executes any cleanup processing necessary for the maintenance of the Source  
     * implementation.  
     *  
     * @throws HousekeepingException If a fatal error occurs during the housekeeping  
     * process  
     */  
    public void cleanUp() throws HousekeepingException;  
  
    /**  
     * Closes the current Housekeeping object and cancels any cleanUp calls in  
     * process.  
     */  
    public void close();  
}
```

The Client System

The two main interfaces of the Client System are the `ClientAdministration` interface that handles non-import specific client functions, and the `Client` interface that allows the engine to communicate with the document repository, such as UCM.

The ClientAdministration Interface Overview

Ensuring the client system, usually a document repository, is active and acquiring a list of the expected meta-data to be provided during each document input are the main functions of this interface.

The ClientAdministration Interface in Java

```
package com.oracle.documaker.ucmimporter.client;  
  
import java.util.List;
```

```

import com.oracle.documaker.ucmimporter.exceptions.ClientException;
import com.oracle.documaker.ucmimporterlib.MetadataDefinition;

public interface ClientAdministration {
    /**
     * Returns the meta-data information each file needs to provide for import
     * functionality.
     *
     * @return The collection of MetadataDefinition objects representing import
     * information
     * @throws ClientException if a content management system error occurs
     */
    public List<MetadataDefinition> getMetadataInfo() throws ClientException;

    /**
     * Pings the content management server to see if it is alive.
     *
     * @return True if the server responds, False otherwise
     * @throws ClientException if a content management system error occurs
     */
    public boolean ping() throws ClientException;
}

```

The Client Interface Overview

The Client interface provides the communication channel between the Connector's engine and the document repository (UCM). It consists of three entry points: `beginImport` called before any files are imported, `importFile` called to import a specific file, and `endImport` called after all file import attempts have been made.

The Client Interface in Java

```

package com.oracle.documaker.ucmimporter.client;

import com.oracle.documaker.ucmimporter.exceptions.ClientException;
import com.oracle.documaker.ucmimporterlib.FileData;

/**
 * ClientInterface describes the functionality that must be implemented by each
 * client class.
 *
 * @since 1.0
 */
public interface ClientInterface {
    /**
     * Initializes the import of a batch of files.
     *
     * @throws ClientException If the batch import initialization fails
     */
    public void beginImport() throws ClientException;

    /**
     * Imports a single file into the content management system.
     *
     * @param fileData The meta-data associated with the file being imported.
     * @throws ClientException If a non-import related error occurs during
     * processing. Import errors will be denoted in the
     * fileData object's resultCode field.
     */
    public void importFile(FileData fileData) throws ClientException;
}

```

```
/**
 * Finalizes the import of a batch of files.
 *
 * @throws ClientException If the batch import finalization fails
 */
public void endImport() throws ClientException;
}
```

Other Interfaces

There are a number of other interfaces that source and client implementations must provide, but these are minor and have to do mainly with supporting classes. See the javadocs for information about these.

The Documaker Source System Implementation

The Source System interfaces described above are implemented to create the Source for Oracle Documaker. This section is notes on the specifics of the Documaker Source implementation. Since the Source and Client systems must cooperate to make the Connector implementation a success, these notes would be useful to someone implementing a different Client system to be used with the Documaker Source.

Source-Interface Implementation

configure Method:

The Documaker source class has many flexible configuration options which are provided via a Properties object argument to the **configure** method. The **configure** method is used to configure a Documaker source instance prior to retrieving records / documents.

The **configure** method also takes as argument a list of MetaDataDefinition objects that define the column names, types and sizes expected by UCM. This list is used by the Documaker source to make sure any UCM requirements are met prior to returning a batch of documents for import.

Logging Conflict

The core Connector and Documaker source both use the Log4j logging library. At startup, the code queries the `-Dlog4j.configuration` property which should point to a valid log4j XML configuration file. If this parameter does not exist, then the default configuration is applied (as defined by the BasicConfigurator). Today, this second set of configuration data overrides the configuration described in the document.

Here is an example of a command line property:

```
-Dlog4j.configuration=log4j.xml
```

getHouseKeeper Method:

The **getHouseKeeper** method returns an instance of the HouseKeeperSingleton Class which implements the Housekeeping interface. This class is responsible for performing the cleanup of disk and table resources. It removes the left over empty directories produced by the Documaker Batch process once their documents have been imported into UCM and removed from disk by the Documaker Connector. Table records are only removed when the current date time stamp exceeds the RETENTION date time stamp and their STATUSCD value is equal to 1.

getFileBatch Method:

Once configured, the Documaker Source returns documents ready for import into UCM via calls to its **getFileBatch** method. The **getFileBatch** method also sets the STATUSCD column value equal to 3 for the records that represent the documents returned so that other Documaker source instances know not to

process those records again. It also sets the STARTTIME time stamp value to indicate at which time the import process began.

ackProcess Method:

The Documaker source updates the STATUSCD column for each record that represents a document once that import process takes place. It does this via the **ackProcess** method which takes as argument a batch of documents that were processed by UCM. The **ackProcess** method sets the value equal to 1 for successful imports and 2 for failed imports. In addition, **ackProcess** also updates the ENDTIME time stamp to indicate when the import process ended. It then calculates the RETENTION time stamp value based on the value provided by the *platform.retention.time* configuration option and updates the column with the expiration date. During failed imports, **ackProcess** also updates the RESULTDESC column.

isValid Method:

Validates the DocumakerSource instance.

setMaxBatchSize Method:

Sets the number of records **getFileBatch** method should return.

close Method:

Cleans up resources and JDBC connections when the DocumakerSource instance is no longer needed.

getTableColumns Method:

(Extension to base interface) Returns a List of ColumnMetadata objects representing the table metadata.

Housekeeping-Interface Implementation

The Housekeeping interface is implemented for Documaker in the HouseKeeperSingleton Class. This section is notes on the method implementations in that class.

cleanUp Method:

Performs cleanup of disk and table resources. Call this method periodically to perform cleanup of disk and table resources. Disk resources are only cleaned if there is one or more roots specified in the *platform.housekeeping.root* property passed into the getInstance method. Folders under each root are only cleaned if they contain the file specified by *platform.housekeeping.transact.file* property, which signifies they are ready for cleanup. Table resources (records) are only cleaned (removed) if the current date exceeds their RETENTION value.

close Method:

Cleans up resources and JDBC connections when the HouseKeeperSingleton instance is no longer needed.

Appendix A – Windows Service Application

dm_ucm_connector.exe Service Application

This application is a launcher/wrapper for the Oracle DC Connector. It loads and calls the more generic **dm_service.dll** with parameters specific to the Oracle DC. This application is set up to be run as the Windows service. It can perform this setup itself or undo this setting by being run from the command line. If run from the command line, there must be a parameter of either:

install	Installs the Windows service and terminates.
uninstall	Removes the Windows service and terminates.

Example: `dm_ucm_connector.exe install`

If the program is run without any parameter, it expects to be running as a Windows service. It loads the `dm_service.dll`, redirects both *standard out* and *standard error* output to files and attempts to register with the Windows Service Manager. If the application is not running as a service, these calls will fail and the application terminates. Therefore, this application should not be run from the command line.

The dm_service.dll

The `dm_service.dll` is a generic tool for running a Java application as a Windows Service. The DLL is loaded and called by the `dm_ucm_connector.exe` application. It is parameterized by a parameter file named after the application, in this case `dm_ucm_connector.properties`. The DLL is therefore reusable for other Java applications by writing a wrapper application with a different name. Oracle DC only uses it from the `dm_ucm_connector.exe` application.

The dm_ucm_connector.properties File

When called to run a Java application, `dm_service.dll` looks for a `name.properties` file and uses the contents to load and run a java application as a service. The `name` portion of the properties file name is derived from the name of the calling application, so in the case of the Oracle DC, the service configuration is read from the file:

`dm_ucm_connector.properties`

The properties in this file and the default values provided in the case of the Oracle DC are given in the table below.

Oracle DC Service Wrapper Properties	
Property Name	Oracle DC Default Value Description
<code>service.debugging</code>	0 Set to 1 to enable debug-level logging to the file dm_ucm_connector-service.log .
<code>service.jvm.args.length</code>	2

	Count of service.jvm.args.# arguments. The properties starting with the service.jvm prefix define the parameters that are passed to the JVM when it is created. These are <i>not</i> the parameters that will be passed to the main() function in Java (see the service.main prefix items below).
service.jvm.args.1	-Djava.class.path\=lib/DocucorpStartup.jar 1 st argument to the JVM
service.jvm.args.2	-Dlog4j.configuration\=log4j.xml 2 nd argument to the JVM
service.startup.class	com/docucorp/startup/Startup Path to the Java class which contains the main() function called to start the Java application.
service.path	C:\Program Files\oracle_dm_ucm_connector\jre\bin;C:\Program Files\oracle_dm_ucm_connector\jre\bin\client; Directories pre-pended to the PATH for the service session. The main use of this is to define the JVM to be used to run the program.
service.main.args.length	5 Count of service.main.args.# arguments below. The properties starting with the service.main prefix define the parameters that are passed to the Java <i>main</i> function.
service.main.args.1	com.oracle.documaker.ucmimporter.Driver 1 st argument to the <i>main</i> Java class above.
service.main.args.2	-config 2 nd argument...
service.main.args.3	ucm_connector.properties 3 rd argument...
service.main.args.4	-sourceconfig 4 th argument...
service.main.args.5	documaker_db.properties 5 th argument...

Appendix B – Example Files Using TRN_FIELDS INI Options to Map Index Data in Documaker

XML Extract Input to Documaker

This file is the inbound variable data to Documaker. This example contains two transactions, each of which will generate one or more output documents. Each transaction is an XML document and each starts with the `<?xml...>` header record indicated in **bold**. The XML is concatenated into a stream into Documaker. In this example text, the **bold blue** text is mapped data which will be used as metadata in UCM.

```
<?xml version="1.0" encoding="UTF-8"?>
<InterfaceRequest>
  <Header>
    <Key1>DOCCDEMO</Key1>
    <Key2>LIFE</Key2>
    <KeyID>67-875747</KeyID>
    <Run_Date>01-OCT-2008 04:12:58 PM</Run_Date>
    <TRANCODE>NB</TRANCODE>
    <DOCTYPE>LIFE</DOCTYPE>
    <PRODUCT>Foundation Life</PRODUCT>
    <SECGROUP>Archived</SECGROUP>
    <AUTHOR>Steven Saunders</AUTHOR>
    <CABINET>CAB1</CABINET>
  </Header>
  <SystemRequest>
    <MessageID>1236474</MessageID>
    <Target>EPOLICY</Target>
    <Target>
      <GO>35235</GO>
      <mode>print</mode>
    </Target>
    <CMD>Print</CMD>
  </SystemRequest>
  <Data>
    <POLICY_NUMBER>67-875747</POLICY_NUMBER>
    <POLICY_ISSUE_DATE>01-OCT-2008 04:12:58 PM</POLICY_ISSUE_DATE>
    <EFFDATE>01-NOV-2008 12:00:00.00 AM</EFFDATE>
    <EXPDATE>01-NOV-2009 12:00:00.00 AM</EXPDATE>
    <CLASS_OF_RISK>A</CLASS_OF_RISK>
    <STATE_CODE>TX</STATE_CODE>
    <PAYEE>Carl Roberts</PAYEE>
    <CUSTID>cjr01</CUSTID>
    <INSURED>
      <PREFIX>Mr.</PREFIX>
      <FNAME>Carl</FNAME>
      <MNAME></MNAME>
      <LNAME>Roberts</LNAME>
      <SEX>M</SEX>
      <ADDRESS1>2727 Paces Ferry Road</ADDRESS1>
      <ADDRESS2>Suite II-900</ADDRESS2>
      <CITY>Atlanta</CITY>
      <STATE>GA</STATE>
      <ZIP>30339</ZIP>
      <BIRTHDATE>15-JUL-1980</BIRTHDATE>
      <INSSAN>123456789</INSSAN>
    </INSURED>
  </Data>
</InterfaceRequest>
```

```

        <DAYPHONE>2148762789778</DAYPHONE>
        <NIGHTPHONE>2148974464</NIGHTPHONE>
        <BIRTHCITY>Anaheim</BIRTHCITY>
        <BIRTHSTATE>CA</BIRTHSTATE>
        <DRIVERSTATE>FL</DRIVERSTATE>
        <DRIVERLICENSE>987987YIU</DRIVERLICENSE>
    </INSURED>
    <AGENT>
        <PREFIX>Mr.</PREFIX>
        <FNAME>John</FNAME>
        <LNAME>Doe</LNAME>
        <ADDRESS1>2727 Paces Ferry Road</ADDRESS1>
        <CITY>Atlanta</CITY>
        <STATE>GA</STATE>
        <ZIP>30339</ZIP>
        <EMAIL>jdoe@amergen.com</EMAIL>
        <PHONE>2148582200</PHONE>
        <AgentNo>R98798</AgentNo>
        <CustServPhone>8882637436</CustServPhone>
        <CustServOpenTime>8:00</CustServOpenTime>
        <CustServCloseTime>5:00</CustServCloseTime>
        <CustServTimeZone>eastern</CustServTimeZone>
    </AGENT>
    <POLICY_DATA>
        <PolicyValue>10000000</PolicyValue>
        <PolicyIssueDate>01032005</PolicyIssueDate>
        <PolicyEndDate>01032025</PolicyEndDate>
        <IssueState>GA</IssueState>
        <CostofInsurance>99200</CostofInsurance>
        <CostofInsuranceRate>992</CostofInsuranceRate>
        <CostofInsurance_Option>Level</CostofInsurance_Option>
        <Smoker>N</Smoker>
        <DeathBenefitType>Increasing</DeathBenefitType>
        <AnnualPremium>101900</AnnualPremium>
        <PremiumFrequency>Monthly</PremiumFrequency>
        <PremiumAmount>8492</PremiumAmount>
        <FlatExtra>0</FlatExtra>
        <AdminCharges>2700</AdminCharges>
        <MultipleExtra>0</MultipleExtra>
    </POLICY_DATA>
    <BENEFICIARY>
        <Name>Mary Smith</Name>
        <Relationship>Wife</Relationship>
    </BENEFICIARY>
    <BENEFICIARY>
        <Name>Holy Anna Smith</Name>
        <Relationship>Daughter</Relationship>
    </BENEFICIARY>
    </Data>
</InterfaceRequest>
<?xml version="1.0" encoding="UTF-8"?>
<InterfaceRequest>
    <Header>
        <Key1>DOCCDEMO</Key1>
        <Key2>LIFE</Key2>
        <KeyID>99-456789</KeyID>
        <Run_Date>12-OCT-2008 10:31:12.01 AM</Run_Date>
        <TRANCODE>NB</TRANCODE>
        <DOCTYPE>LIFE</DOCTYPE>
        <PRODUCT>Foundation Life</PRODUCT>
        <SECGROUP>Archived</SECGROUP>
        <AUTHOR>Carl Roberts</AUTHOR>
        <CABINET>CAB1</CABINET>
    </Header>

```

```

</Header>
<SystemRequest>
  <MessageID>1236474</MessageID>
  <Target>EPOLICY</Target>
  <Target>
    <GO>35235</GO>
    <mode>print</mode>
  </Target>
  <CMD>Print</CMD>
</SystemRequest>
<Data>
  <POLICY_NUMBER>99-456789</POLICY_NUMBER>
  <POLICY_ISSUE_DATE>10-OCT-2008 10:31:12.01 AM</POLICY_ISSUE_DATE>
  <EFFDATE>01-NOV-2008 12:00:00.01 AM</EFFDATE>
  <EXPDATE>01-NOV-2009 12:00:00.01 AM</EXPDATE>
  <CLASS_OF_RISK>A</CLASS_OF_RISK>
  <STATE_CODE>GA</STATE_CODE>
  <PAYEE>Steven Saunders</PAYEE>
  <CUSTID>ssaunder</CUSTID>
  <INSURED>
    <PREFIX>Mr.</PREFIX>
    <FNAME>Steven</FNAME>
    <MNAME>J</MNAME>
    <LNAME>Saunders</LNAME>
    <SEX>M</SEX>
    <ADDRESS1>2727 Paces Ferry Road</ADDRESS1>
    <ADDRESS2>Suite II-900</ADDRESS2>
    <CITY>Atlanta</CITY>
    <STATE>GA</STATE>
    <ZIP>30339</ZIP>
    <BIRTHDATE>14-FEB-1970</BIRTHDATE>
    <INSSAN>012345678</INSSAN>
    <DAYPHONE>2148762789778</DAYPHONE>
    <NIGHTPHONE>2148974464</NIGHTPHONE>
    <BIRTHCITY>Pittsburg</BIRTHCITY>
    <BIRTHSTATE>PN</BIRTHSTATE>
    <DRIVERSTATE>GA</DRIVERSTATE>
    <DRIVERLICENSE>987987YIU</DRIVERLICENSE>
  </INSURED>
  <AGENT>
    <PREFIX>Mr.</PREFIX>
    <FNAME>John</FNAME>
    <LNAME>Doe</LNAME>
    <ADDRESS1>2727 Paces Ferry Road</ADDRESS1>
    <CITY>Atlanta</CITY>
    <STATE>GA</STATE>
    <ZIP>30339</ZIP>
    <EMAIL>jdoe@amergen.com</EMAIL>
    <PHONE>2148582200</PHONE>
    <AgentNo>R98798</AgentNo>
    <CustServPhone>8882637436</CustServPhone>
    <CustServOpenTime>8:00</CustServOpenTime>
    <CustServCloseTime>5:00</CustServCloseTime>
    <CustServTimeZone>eastern</CustServTimeZone>
  </AGENT>
  <POLICY_DATA>
    <PolicyValue>10000000</PolicyValue>
    <PolicyIssueDate>01032005</PolicyIssueDate>
    <PolicyEndDate>01032025</PolicyEndDate>
    <IssueState>GA</IssueState>
    <CostofInsurance>99200</CostofInsurance>
    <CostofInsuranceRate>992</CostofInsuranceRate>
    <CostofInsurance_Option>Level</CostofInsurance_Option>

```

```

        <Smoker>N</Smoker>
        <DeathBenefitType>Increasing</DeathBenefitType>
        <AnnualPremium>101900</AnnualPremium>
        <PremiumFrequency>Monthly</PremiumFrequency>
        <PremiumAmount>8492</PremiumAmount>
        <FlatExtra>0</FlatExtra>
        <AdminCharges>2700</AdminCharges>
        <MultipleExtra>0</MultipleExtra>
    </POLICY_DATA>
    <BENEFICIARY>
        <Name>Mary Saunders</Name>
        <Relationship>Wife</Relationship>
    </BENEFICIARY>
    <BENEFICIARY>
        <Name>Holy Anna Saunders</Name>
        <Relationship>Daughter</Relationship>
    </BENEFICIARY>
</Data>
</InterfaceRequest>

```

Example TRN_FIELDS INI Setup

This section of the Documaker configuration INI file (either FSI SYS . INI or FSI USER . INI) defines new GVM variable names for the subset of data fields in the above XML input that we want to reference and use for UCM metadata. That is, this file creates new simple names for the fields above that are highlighted in **bold blue**.

```

; XPATH to data elements in XML import file listed above,
; stores data in named GVMs
;
;   GVM Name      = XPATH to XML input file field for value
;   -----
< TRN_FIELDS >
KEY1              = !/InterfaceRequest/Header/Key1
KEY2              = !/InterfaceRequest/Header/Key2
KEYID             = !/InterfaceRequest/Header/KeyID
TRANCODE         = !/InterfaceRequest/Header/TRANCODE
RUNDATE          = !/InterfaceRequest/Header/Run_Date
PRODUCT          = !/InterfaceRequest/Header/PRODUCT
SECGROUP         = !/InterfaceRequest/Header/SECGROUP
DOCTYPE         = !/InterfaceRequest/Header/DOCTYPE
CABINET          = !/InterfaceRequest/Header/CABINET
AUTHOR           = !/InterfaceRequest/Header/AUTHOR
CURRUSER         = !/InterfaceRequest/Header/CURRUSER
CUSTID           = !/InterfaceRequest/Header/CUSTID
POLNUM           = !/InterfaceRequest/Data/POLICY_NUMBER
INSFNAME         = !/InterfaceRequest/Data/INSURED/FNAME
INSLNAME         = !/InterfaceRequest/Data/INSURED/LNAME
INSADD1          = !/InterfaceRequest/Data/INSURED/ADDRESS1
INSADD2          = !/InterfaceRequest/Data/INSURED/ADDRESS2
INSCITY          = !/InterfaceRequest/Data/INSURED/CITY
INSSTATE         = !/InterfaceRequest/Data/INSURED/STATE
INSZIP           = !/InterfaceRequest/Data/INSURED/ZIP
INSPHONE         = !/InterfaceRequest/Data/INSURED/DAYPHONE
INSDOB           = !/InterfaceRequest/Data/INSURED/BIRTHDATE
WIPREASON        = !/InterfaceRequest/Data/WIPREASON
INDEX01          = !/InterfaceRequest/Data/AGENT/AgentNo
INDEX02          = !/InterfaceRequest/Data/EFFDATE
INDEX03          = !/InterfaceRequest/Data/EXPDATE
INDEX04          = !/InterfaceRequest/Data/AGENT/AgentNo
INDEX05          = !/InterfaceRequest/Data/EFFDATE

```

```

INDEX06      = !/InterfaceRequest/Data/EXPDATE
INDEX07      = !/InterfaceRequest/Data/AGENT/AgentNo
INDEX08      = !/InterfaceRequest/Data/EFFDATE
INDEX09      = !/InterfaceRequest/Data/EXPDATE
INDEX10      = !/InterfaceRequest/Data/AGENT/AgentNo
INDEX11      = !/InterfaceRequest/Data/EFFDATE
INDEX12      = !/InterfaceRequest/Data/EXPDATE
AGENCYID     = !/InterfaceRequest/Data/AGENT/AgentNo
EFFDATE      = !/InterfaceRequest/Data/EFFDATE
EXPDATE      = !/InterfaceRequest/Data/EXPDATE

```

Example TRNDFDFL.DFD Setup

With the fields mapped into GVM variables, the attributes of each new GVM variable have to be described to Documaker. The GVM variables defined above are highlighted below in **bold blue** text.

```

< FIELDS >
  FIELDNAME = PKG_Offset
  FIELDNAME = TRN_Offset
  FIELDNAME = X_Offset
  FIELDNAME = NA_Offset
  FIELDNAME = POL_Offset
  FIELDNAME = SentToManualBatch
  FIELDNAME = KEY1
  FIELDNAME = KEY2
  FIELDNAME = KEYID
  FIELDNAME = TRANCODE
  FIELDNAME = RUNDATE
  FIELDNAME = CURRUSER
  FIELDNAME = AGENCYID
  FIELDNAME = EFFDATE
  FIELDNAME = EXPDATE
  FIELDNAME = PRODUCT
  FIELDNAME = SECGROUP
  FIELDNAME = AUTHOR
  FIELDNAME = CABINET
  FIELDNAME = DOCTYPE
  FIELDNAME = ONE
  FIELDNAME = TWO
  FIELDNAME = CUSTID
  FIELDNAME = POLNUM
  FIELDNAME = INSFNAME
  FIELDNAME = INSLNAME
  FIELDNAME = INSADD1
  FIELDNAME = INSADD2
  FIELDNAME = INSCITY
  FIELDNAME = INSSTATE
  FIELDNAME = INSZIP
  FIELDNAME = INSPHONE
  FIELDNAME = INSDOB
  FIELDNAME = WIPREASON
  FIELDNAME = INDEX01
  FIELDNAME = INDEX02
  FIELDNAME = INDEX03
  FIELDNAME = INDEX04
  FIELDNAME = INDEX05
  FIELDNAME = INDEX06
  FIELDNAME = INDEX07
  FIELDNAME = INDEX08
  FIELDNAME = INDEX09

```

```

FIELDNAME = INDEX10
FIELDNAME = INDEX11
FIELDNAME = INDEX12

< FIELD:PKG_Offset >
  INT_TYPE = LONG
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:TRN_Offset >
  INT_TYPE = LONG
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:X_Offset >
  INT_TYPE = LONG
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:NA_Offset >
  INT_TYPE = LONG
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:POL_Offset >
  INT_TYPE = LONG
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 10
  KEY = N
  REQUIRED = N
< FIELD:SentToManualBatch >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 3
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 2
  KEY = N
  REQUIRED = N
< FIELD:KEY1 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 101
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 100
  KEY = Y
  REQUIRED = Y
< FIELD:KEY2 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 101
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 100
  KEY = Y
  REQUIRED = Y
< FIELD:KEYID >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 101
  EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
  EXT_LENGTH = 100
  KEY = N
  REQUIRED = N
< FIELD:TRANCODE >

```



```

    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:RUNDATE >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:CURRUSER >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 101
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 100
    KEY = N
    REQUIRED = N
< FIELD:AGENCYID >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:EFFDATE >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:EXPDATE >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:PRODUCT >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:SECGROUP >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:AUTHOR >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:CABINET >

```

```

    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:DOCTYPE >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:ONE >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 11
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 10
    KEY = Y
    REQUIRED = Y
< FIELD:TWO >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 11
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 10
    KEY = Y
    REQUIRED = Y
< FIELD:CUSTID >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:POLNUM >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 101
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 100
    KEY = N
    REQUIRED = N
< FIELD:INSFNAME >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:INSLNAME >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:INSADD1 >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 101
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 100
    KEY = N
    REQUIRED = N
< FIELD:INSADD2 >

```

```

    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 101
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 100
    KEY = N
    REQUIRED = N
< FIELD:INSCITY >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:INSSTATE >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 4
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 3
    KEY = N
    REQUIRED = N
< FIELD:INSZIP >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 12
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 11
    KEY = N
    REQUIRED = N
< FIELD:INSPHONE >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:INSDOB >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:WIPREASON >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 26
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 25
    KEY = N
    REQUIRED = N
< FIELD:INDEX01 >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:INDEX02 >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:INDEX03 >

```

```

    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:INDEX04 >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:INDEX05 >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:INDEX06 >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:INDEX07 >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:INDEX08 >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:INDEX09 >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:INDEX10 >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:INDEX11 >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 31
    EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = N
< FIELD:INDEX12 >

```

```
INT_TYPE = CHAR_ARRAY  
INT_LENGTH = 31  
EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM  
EXT_LENGTH = 30  
KEY = N  
REQUIRED = N
```

Example RCBDFFDL.DFD Setup

For this example, the RCBDFFDL.DFD file is identical to the TRNDFDDL.DFD file.

Appendix C – Example Documaker DAL Scripts

These scripts are referenced in the DAL configuration of the Oracle Documaker INI file. This configuration was described earlier, but is presented here for easy reference:

```
; Enable the Banner and Transaction DAL Scripting  
< BATCH6 >  
    EnableBatchBanner      = Yes  
    EnableTransBanner      = Yes  
    BatchBannerBeginScript  = AOR_PREB  
    BatchBannerEndScript   = AOR_POSTB  
    TransBannerBeginScript = AOR_PRET  
    TransBannerEndScript   = AOR_POSTT  
    Printer                 = Printer6
```

In the example presented earlier in the text, they are called during processing of the Batch6 grouping. The code for the routines is presented below, with the main entry points listed above shown below in the order they are called. The TransBanner... routines are called repeatedly for each transaction that is part of the batch, before the final call to BatchBannerEndScript.

```
BatchBannerBeginScript  = AOR_PREB  
    TransBannerBeginScript = AOR_PRET  
    TransBannerEndScript   = AOR_POSTT  
BatchBannerEndScript    = AOR_POSTB
```

BatchBannerBeginScript = AOR_PREB

BEGINSUB AOR_PREB

```
* -----  
*   Begin batch  
*   Clear variables once per Recip Batch  
* -----  
#AOR_Debug=GETINIBOOL(,PRINTERID(),"AORDebug")  
  
IF #AOR_Debug  
    RPLogMsg(NL() & "   ** AOR_PREB:" & NL() )  
END  
AOR_RecipBatch = AOR_RecipBatch  
#AOR_BatchCount = #AOR_BatchCount  
#AOR_Processed = #AOR_Processed  
#AOR_Count     = #AOR_Count  
AOR_TableName = AOR_TableName  
#AOR_Init      = #AOR_Init  
IF AOR_RecipBatch != RECIPTATCH()  
    PUTINIBOOL("RunMode","CheckNextRecip",0)  
    #AOR_PerBatch = GETINISTRING(,PRINTERID(),"AORFilesPerBatch","999")  
    AOR_RecipBATCH = RECIPTATCH()  
    #AOR_SubBatch = 0  
    #AOR_Count    = 0  
    #AOR_BatchCount = 0  
END  
AOR_BatchID = RECIPTATCH()  
IF #AOR_Init = 0  
    AOR_JobID = UNIQUESTRING()
```

```

    AOR_TableName = GETINISTRING(,PRINTERID(),"AORTable","AOR")
    DBOPEN(AOR_TableName,"ODBC",".\deflib\aur.dfd", "READ&WRITE&CREATE_IF_NEW")
    DBPREPVARs(AOR_TableName,"AORTABLERecord")
END
#AOR_Init = 1
#AOR_DoEOB = 1
ENDSUB

```

TransBannerBeginScript = AOR_PRET

```

BEGINSUB AOR_PRET
* -----
*   Begin Transaction
*   Setup new filename for recipient batch output file
* -----
IF #AOR_Debug
    RPLogMsg(NL() & "   ** AOR_PRET:" & NL() )
END
AOR_BatchID      = AOR_BatchID
AOR_BatchDir     = AOR_BatchDir
#AOR_Batch       = #AOR_Batch
#AOR_Count       = #AOR_Count
#AOR_PerBatch    = #AOR_PerBatch
#AOR_Processed   = #AOR_Processed
AOR_TransID     = GVM("KEY1") & "-" & GVM("KEY2") & "-" & \
    GVM("KEYID") & "-" & GVM("TRANCODE")
#AOR_Count += 1
IF (#AOR_Count > #AOR_PerBatch)
    #AOR_Count -=1
    CALL("AOR_EOB")
    #AOR_Count = 1
END
TranFile = CALL("AOR_NEWFILE")
#AOR_Exists = PATHEXIST(AOR_BatchDir)
IF #AOR_Exists = 0
    PATHCREATE(AOR_BatchDir)
    #AOR_Exists = PATHEXIST(AOR_BatchDir)
    IF #AOR_Exists = 0
        RPErrorMsg(NL() & "*** AOR batch directory " & \
            AOR_Batchdir & "does not exist!")
    END
END
#AOR_DoEOB = 0
SETDEVICENAME(TranFile)
BREAKBATCH()
ENDSUB

```

TransBannerEndScript = AOR_POSTT

```

BEGINSUB AOR_POSTT
* -----
*   End Transaction
*   Insert new table row with metadata for UCM
*   from GVM variables and reference to uniquely named
*   recipient print stream output.
* -----
IF #AOR_Debug
    RPLogMsg(NL() & "   ** AOR_POSTT:" & NL() )
END
#AOR_PerBatch = #AOR_PerBatch
AOR_BatchDir = AOR_BatchDir

```



```

#AOR_Count = #AOR_Count
#AOR_Debug = GETINIBOOL(,PRINTERID(),"AORDebug")
AOR_TableName = AOR_TableName
AOR_BatchID = AOR_BatchID
AOR_TransID = AOR_TransID
AOR_JobID   = AOR_JobID
*
*       Change to match variables defined in "rcbdfdfldfd"
*       as needed for the implementation
*
* Documaker UCMImporter job processing fields
  AORTABLERecord.JOBID   = AOR_JobID
  AORTABLERecord.BATCHID = AOR_BatchID
  AORTABLERecord.TRANID  = AOR_TransID
  AORTABLERecord.DOCID   = AOR_FName
* TITLE required field by UCM 30 characters max, shows up in search results
  AORTABLERecord.TITLE   = AOR_TransID
* ContentID/Name assigment
  AORTABLERecord.STATUSCD = 0
* Documaker UCMImporter required field for specifying full name of
* file to import
  AORTABLERecord.PFILE   = DEVICENAME()
* UCM required field has to exist in UCM pick list for types or
* will fail to import
  IF HAVEGVM("DOCTYPE")
    AORTABLERecord.TYPE   = GVM("DOCTYPE")
  END
* UCM required field
  IF HAVEGVM("AUTHOR")
    AORTABLERecord.AUTHOR = GVM("AUTHOR")
  END
* UCM required field has to exist in UCM pick list for security groups or will
* fail to import
  IF HAVEGVM("SECGROUP")
    AORTABLERecord.SECGROUP = GVM("SECGROUP")
  END
* UCM mapped custom meta-data, if doesn't exist as same exact name in UCM custom
* fields it will not map but will not error. If UCM custom field was set as
* required and no data is mapped UCM will fail transaction.
* Truncates by default to the max
* length of UCM data type.
  IF HAVEGVM("CABINET")
    AORTABLERecord.CABINET = GVM("CABINET")
  END
  IF HAVEGVM("KEY1")
    AORTABLERecord.KEY1    = GVM("KEY1")
  END
  IF HAVEGVM("KEY2")
    AORTABLERecord.KEY2    = GVM("KEY2")
  END
  IF HAVEGVM("KEYID")
    AORTABLERecord.KEYID   = GVM("KEYID")
  END
  IF HAVEGVM("TRANCODE")
    AORTABLERecord.TRANCODE = GVM("TRANCODE")
  END
  IF HAVEGVM("RUNDATE")
    AORTABLERecord.RUNDATE  = GVM("RUNDATE")
  END
  IF HAVEGVM("CURRUSER")
    AORTABLERecord.CURRUSER = GVM("CURRUSER")
  END
  END

```

```

IF HAVEGVM("AGENCYID")
    AORTABLERecord.AGENCYID = GVM("AGENCYID")
END
IF HAVEGVM("EFFDATE")
    AORTABLERecord.EFFDATE = GVM("EFFDATE")
    AORTABLERecord.TITLE = AOR_TransID & "-" & GVM("EFFDATE")
END
IF HAVEGVM("EXPDATE")
    AORTABLERecord.EXPDATE = GVM("EXPDATE")
END
IF HAVEGVM("CUSTID")
    AORTABLERecord.CUSTID = GVM("CUSTID")
END
IF HAVEGVM("POLNUM")
    AORTABLERecord.POLNUM = GVM("POLNUM")
END
IF HAVEGVM("INSFNAME")
    AORTABLERecord.INSFNAME = GVM("INSFNAME")
END
IF HAVEGVM("INSLNAME")
    AORTABLERecord.INSLNAME = GVM("INSLNAME")
END
IF HAVEGVM("INSADD1")
    AORTABLERecord.INSADD1 = GVM("INSADD1")
END
IF HAVEGVM("INSADD2")
    AORTABLERecord.INSADD2 = GVM("INSADD2")
END
IF HAVEGVM("INSCITY")
    AORTABLERecord.INSCITY = GVM("INSCITY")
END
IF HAVEGVM("INSSTATE")
    AORTABLERecord.INSSTATE = GVM("INSSTATE")
END
IF HAVEGVM("INSZIP")
    AORTABLERecord.INSZIP = GVM("INSZIP")
END
IF HAVEGVM("INSPHONE")
    AORTABLERecord.INSPHONE = GVM("INSPHONE")
END
IF HAVEGVM("INSDOB")
    AORTABLERecord.INSDOB = GVM("INSDOB")
END
IF HAVEGVM("INDEX01")
    AORTABLERecord.INDEX01 = GVM("INDEX01")
END
IF HAVEGVM("INDEX02")
    AORTABLERecord.INDEX02 = GVM("INDEX02")
END
IF HAVEGVM("INDEX03")
    AORTABLERecord.INDEX03 = GVM("INDEX03")
END
IF HAVEGVM("INDEX04")
    AORTABLERecord.INDEX04 = GVM("INDEX04")
END
IF HAVEGVM("INDEX05")
    AORTABLERecord.INDEX05 = GVM("INDEX05")
END
IF HAVEGVM("INDEX06")
    AORTABLERecord.INDEX06 = GVM("INDEX06")
END
IF HAVEGVM("INDEX07")
    AORTABLERecord.INDEX07 = GVM("INDEX07")

```

```

END
IF HAVEGVM("INDEX08")
    AORTABLERecord.INDEX08 = GVM("INDEX08")
END
IF HAVEGVM("INDEX09")
    AORTABLERecord.INDEX09 = GVM("INDEX09")
END
IF HAVEGVM("INDEX10")
    AORTABLERecord.INDEX10 = GVM("INDEX10")
END
IF HAVEGVM("INDEX11")
    AORTABLERecord.INDEX11 = GVM("INDEX11")
END
IF HAVEGVM("INDEX12")
    AORTABLERecord.INDEX12 = GVM("INDEX12")
END
#Rtn = DBADD(AOR_TableName,"AORTABLERecord")
ENDSUB

```

BatchBannerEndScript = AOR_POSTB

BEGINSUB AOR_POSTB

```

* -----
*   End Batch
*   Any necessary clean-up.
* -----

IF #AOR_Debug
    RPLogMsg(NL() & "   ** AOR_POSTB:" & NL() )
END

#AOR_Debug = GETINIBOOL(,PRINTERID(),"AORDebug")
#AOR_Processed = #AOR_Processed
#AOR_Count = #AOR_Count
#AOR_PerBatch = #AOR_PerBatch
AOR_BatchDir = AOR_BatchDir
AOR_TableName = AOR_TableName

IF (#AOR_DoEOB = 1 AND #AOR_Count > 0)
    DBCLOSE(AOR_TableName)
    CALL("AOR_EOB")
    #AOR_Count = 0
END
ENDSUB

```

Internal Routine: AOR_NEWFILE

BEGINSUB AOR_NEWFILE

```

* -----
*   Create a new output file name.
*   Called by Pre-Transaction DAL script.
* -----

IF #AOR_Debug
    RPLogMsg(NL() & "   ** AOR_NEWFILE:" & NL() )
END
AOR_Ext = GETINISTRING(,PRINTERID(), "AORExt", ".pdf")
AOR_BatchDir = AOR_BatchDir
CALL("AOR_NEWPATH")
AOR_Drive = FILEDRIVE( AOR_BatchDir )
AOR_Path = FILEPATH( AOR_BatchDir )
AOR_Last = FILENAME( AOR_BatchDir )
AOR_FName = UNIQUESTRING()

```

```

    AOR_NewFName=FULLFILENAME( AOR_Drive,AOR_Path & AOR_Last,AOR_FName,AOR_Ext)
    RETURN( AOR_NewFName )
ENDSUB

```

Internal Routine: AOR_NEWPATH

```

BEGINSUB AOR_NEWPATH
* -----
*   Create a new output folder.
*   Called by AOR_NEWFILE DAL script.
* -----
    IF #AOR_Debug
        RPLogMsg(NL() & "   ** AOR_NEWPATH:" & NL() )
    END
    #AOR_BatchCount = #AOR_BatchCount
    AOR_BatchID = AOR_BatchID
    AOR_BatchDir = AOR_BatchDir
    #AOR_Count = #AOR_Count
    AORPath = GETINISTRING(,PRINTERID(),"AORPath")
    AOR_Drive   = FILEDRIVE( AORPath )
    AOR_Path    = FILEPATH( AORPath )
    AOR_Last    = FILENAME( AORPath )
    AOR_Rootdir = FULLFILENAME( AOR_Drive, AOR_Path, AOR_Last, "" )
    IF #AOR_Count = 1
        #AOR_BatchCount += 1
        AOR_Drive   = FILEDRIVE(AOR_RootDir)
        AOR_Path    = FILEPATH(AOR_RootDir)
        AOR_Last    = FILENAME(AOR_RootDir)
        AOR_BatchDir = FULLFILENAME(AOR_Drive,AOR_Path & AOR_Last,AOR_JobID,"")
        AOR_Drive   = FILEDRIVE(AOR_BatchDir)
        AOR_Path    = FILEPATH(AOR_BatchDir)
        AOR_Last    = FILENAME(AOR_BatchDir)
        AOR_BatchDir =
FULLFILENAME(AOR_Drive,AOR_Path&AOR_Last,AOR_BatchID&"x"&#AOR_BatchCount, "")
    END
ENDSUB

```

Internal Routine: AOR_EOB

```

BEGINSUB AOR_EOB
* -----
*   Create transact.dat file listing the contents of
*   each batch folder created with count of output files,
*   maximum output files per batch folder, statistical
*   information. Existence of this file is indicates a
*   batch folder has been completed and is used by the
*   Documaker DC source for housekeeping functions.
*   Called by Post Batch DAL script.
* -----
    IF #AOR_Debug
        RPLogMsg(NL() & "   ** AOR_EOB:" & NL() )
    END
    #AOR_PerBatch = #AOR_PerBatch
    #AOR_Count = #AOR_Count
    AOR_BatchDir = AOR_BatchDir
    AORRootDir = AORRootDir
    IF #AOR_DoEOB = 1
        AOR_LogFile = FULLFILENAME(,AOR_BatchDir,"transact", ".dat")
        DBOPEN(AOR_LogFile,"ASCII", ".\deflib\AORT.dfd", \
            "READ&WRITE&TRUNCATE&CREATE_IF_NEW")
        DBPREPVAR(AOR_LogFile,"AOREOTRecord")
    END
ENDSUB

```

```
AOREOTRecord.Record = FILENAME(AOR_BatchDir) & " " & \  
                        #AOR_PerBatch & " " & \  
                        #AOR_Count  
DBADD(AOR_LogFile, "AOREOTRecord")  
DBCLOSE(AOR_LogFile)  
END  
ENDSUB
```

Appendix D – Documaker Setup for DAL Output to a Database Table

DDL for DAL Output Records

This is an example of DDL statements to create a minimally acceptable database table for use by Documaker and the Oracle Documaker Connector. Documaker DAL writes this table and the Oracle DC reads the records for incoming documents to process.

```
CREATE TABLE "ORACLE"."AOR" (  
    "JOBID" VARCHAR2(50) NOT NULL,  
    "TRANID" VARCHAR2(50) NOT NULL,  
    "BATCHID" VARCHAR2(50) NOT NULL,  
    "DOCID" VARCHAR2(50) NOT NULL,  
    "NAME" VARCHAR2(30),  
    "TYPE" VARCHAR2(30),  
    "TITLE" VARCHAR2(255),  
    "AUTHOR" VARCHAR2(50),  
    "SECGROUP" VARCHAR2(30),  
    "CABINET" VARCHAR2(30),  
    "PFILE" VARCHAR2(255),  
    "STATUSCD" INTEGER DEFAULT 0 NOT NULL,  
    "STARTTIME" TIMESTAMP,  
    "ENDTIME" TIMESTAMP,  
    "RESULTDESC" VARCHAR2(2000),  
    "RETENTION" TIMESTAMP,  
    "CATEGORY" VARCHAR2(30),  
    "KEY1" VARCHAR2(100),  
    "KEY2" VARCHAR2(100),  
    "KEYID" VARCHAR2(100),  
    "TRANCODE" VARCHAR2(30),  
    "RUNDATE" TIMESTAMP,  
    "CURRUSER" VARCHAR2(30),  
    "AGENCYID" VARCHAR2(30),  
    "EFFDATE" TIMESTAMP,  
    "EXPDATE" TIMESTAMP,  
    "CUSTID" VARCHAR2(30),  
    "POLNUM" VARCHAR2(100),  
    "INSFNAME" VARCHAR2(30),  
    "INSLNAME" VARCHAR2(30),  
    "INSADD1" VARCHAR2(30),  
    "INSADD2" VARCHAR2(30),  
    "INSCITY" VARCHAR2(30),  
    "INSSTATE" VARCHAR2(5),  
    "INSZIP" VARCHAR2(30),  
    "INSPHONE" VARCHAR2(30),  
    "INSDOB" DATE,  
    "INDEX01" VARCHAR2(30), "INDEX02" VARCHAR2(30), "INDEX03" VARCHAR2(30),  
    "INDEX04" VARCHAR2(30), "INDEX05" VARCHAR2(30), "INDEX06" VARCHAR2(30),  
    "INDEX07" VARCHAR2(30), "INDEX08" VARCHAR2(30), "INDEX09" VARCHAR2(30),  
    "INDEX10" VARCHAR2(30), "INDEX11" VARCHAR2(30), "INDEX12" VARCHAR2(30),  
    PRIMARY KEY ("JOBID", "TRANID", "BATCHID", "DOCID") VALIDATE  
);  
  
CREATE INDEX "ORACLE"."AORIDX1" ON "ORACLE"."AOR" ("DOCID");
```

Database Table DFD (Data Format Definition) File (AOR.DFD)

This DFD file describes the database table schema to Documaker. In this example, with a database table called AOR, this file is called AOR.DFD and is used to define the interface between Oracle Documaker and the data table AOR in the Oracle database used for storage of the extract data. The DFD file is used in the DAL script where you will see references to AOR in the DBAdd(), DBOpen() and DBPrepare() calls. This file is loaded in the DBOpen() so Documaker knows what columns it has available and uses that information to insert the row in the AOR table. In the example, this file is in the deflib subdirectory per the call in the DAL script:

```
DBOPEN(AOR_TableName, "ODBC", ".\deflib\AOR.dfd", "READ&WRITE&CREATE_IF_NEW")
```

The AOR.DFD File

```
< FIELDS >
  FIELDNAME = JOBID
  FIELDNAME = TRANID
  FIELDNAME = BATCHID
  FIELDNAME = DOCID
  FIELDNAME = NAME
  FIELDNAME = TYPE
  FIELDNAME = TITLE
  FIELDNAME = AUTHOR
  FIELDNAME = SECGROUP
  FIELDNAME = PFILE
  FIELDNAME = CATEGORY
  FIELDNAME = CABINET
  FIELDNAME = STATUSCD
  FIELDNAME = KEY1
  FIELDNAME = KEY2
  FIELDNAME = KEYID
  FIELDNAME = TRANCOD
  FIELDNAME = RUNDAT
  FIELDNAME = CURRUS
  FIELDNAME = AGENCI
  FIELDNAME = EFFDAT
  FIELDNAME = EXPDAT
  FIELDNAME = INSFNA
  FIELDNAME = INSLNA
  FIELDNAME = INSADD1
  FIELDNAME = INSADD2
  FIELDNAME = INSCIT
  FIELDNAME = INSSTA
  FIELDNAME = INSZIP
  FIELDNAME = INSPHO
  FIELDNAME = INSDOB
  FIELDNAME = INDEX01
  FIELDNAME = INDEX02
  FIELDNAME = INDEX03
  FIELDNAME = INDEX04
  FIELDNAME = INDEX05
  FIELDNAME = INDEX06
  FIELDNAME = INDEX07
  FIELDNAME = INDEX08
  FIELDNAME = INDEX09
  FIELDNAME = INDEX10
  FIELDNAME = INDEX11
  FIELDNAME = INDEX12

< FIELD:JOBID >
  INT_TYPE = CHAR_ARRAY
```



```

    INT_LENGTH = 47
    EXT_TYPE = CHAR_ARRAY
    EXT_LENGTH = 47
    KEY = Y
    REQUIRED = Y

< FIELD:TRANID >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 47
    EXT_TYPE = CHAR_ARRAY
    EXT_LENGTH = 47
    KEY = Y
    REQUIRED = Y

< FIELD:BATCHID >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 47
    EXT_TYPE = CHAR_ARRAY
    EXT_LENGTH = 47
    KEY = Y
    REQUIRED = Y

< FIELD:DOCID >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 47
    EXT_TYPE = CHAR_ARRAY
    EXT_LENGTH = 47
    KEY = Y
    REQUIRED = Y

< FIELD:NAME >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 47
    EXT_TYPE = CHAR_ARRAY
    EXT_LENGTH = 47
    KEY = N
    REQUIRED = Y

< FIELD:TYPE >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 10
    EXT_TYPE = CHAR_ARRAY
    EXT_LENGTH = 10
    KEY = N
    REQUIRED = Y

< FIELD:TITLE >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 30
    EXT_TYPE = CHAR_ARRAY
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = Y

< FIELD:AUTHOR >
    INT_TYPE = CHAR_ARRAY
    INT_LENGTH = 30
    EXT_TYPE = CHAR_ARRAY
    EXT_LENGTH = 30
    KEY = N
    REQUIRED = Y

< FIELD:SECGROUP >

```

```

INT_TYPE = CHAR_ARRAY
INT_LENGTH = 30
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 30
KEY = N
REQUIRED = Y

< FIELD:PFILE >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 255
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 255
KEY = N
REQUIRED = Y

< FIELD:CATEGORY >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 30
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 30
KEY = N
REQUIRED = N

< FIELD:CABINET >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 30
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 30
KEY = N
REQUIRED = N

< FIELD:STATUSCD >
INT_TYPE = LONG
INT_LENGTH = 1
EXT_TYPE = LONG
EXT_LENGTH = 1
KEY = N
REQUIRED = Y

< FIELD:KEY1 >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 100
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 100
KEY = N
REQUIRED = N

< FIELD:KEY2 >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 100
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 100
KEY = N
REQUIRED = N

< FIELD:KEYID >
INT_TYPE = CHAR_ARRAY
INT_LENGTH = 100
EXT_TYPE = CHAR_ARRAY
EXT_LENGTH = 100
KEY = N
REQUIRED = N

```

```

< FIELD:TRANCODE >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:RUNDATE >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:CURRUSER >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 100
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 100
  KEY = N
  REQUIRED = N

< FIELD:AGENCYID >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:EFFDATE >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:EXPDATE >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N

< FIELD:CUSTID >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:POLNUM >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 100
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 100
  KEY = N
  REQUIRED = N

```

```

< FIELD:INSFNAME >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:INSLNAME >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:INSADD1 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 100
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 100
  KEY = N
  REQUIRED = N

< FIELD:INSADD2 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 100
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 100
  KEY = N
  REQUIRED = N

< FIELD:INSCITY >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:INSSTATE >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 3
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 3
  KEY = N
  REQUIRED = N

< FIELD:INSZIP >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 11
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 11
  KEY = N
  REQUIRED = N

< FIELD:INSPHONE >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

```

```

< FIELD:INSDOB >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:WIPREASON >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:INDEX01 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:INDEX02 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:INDEX03 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:INDEX04 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:INDEX05 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:INDEX06 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

```

```

< FIELD:INDEX07 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:INDEX08 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:INDEX09 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:INDEX10 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:INDEX11 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N

< FIELD:INDEX12 >
  INT_TYPE = CHAR_ARRAY
  INT_LENGTH = 30
  EXT_TYPE = CHAR_ARRAY
  EXT_LENGTH = 30
  KEY = N
  REQUIRED = N   REQUIRED = N

< KEYS >
  KEYNAME = BATCH
  KEYNAME = DOCID

< KEY:BATCH >
  EXPRESSION = JOBID+TRANID+BATCHID
  FIELDLIST = JOBID,TRANID,BATCHID

< KEY:DOCID >
  EXPRESSION = DOCID
  FIELDLIST = DOCID

```




Oracle Documaker Connector Installation, Administration and Customization Guide
July 24, 2009

Primary Authors: Eugene Thompson, Joe Roberts, Steve Saunders
Contributing Authors: Lowell Von Egger, Phil Iorio

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Copyright © 2009, Oracle and/or its affiliates. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.