

**Oracle Utilities Billing Component**  
Installation and Configuration Guide, Volume 2  
Release 1.6.1.22 for Windows  
**E18207-23**

November 2017

Oracle Utilities Billing Component/Billing Component Installation and Configuration Guide, Volume 2,  
Release 1.6.1.22 for Windows

E18207-23

Copyright © 1999, 2017 Oracle and/or its affiliates. All rights reserved.

Primary Author: Lou Proseri

Contributor: Steve Pratt

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

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

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

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

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

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

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

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

## NOTIFICATION OF THIRD-PARTY LICENSES

Oracle Utilities software contains third party, open source components as identified below. Third-party license terms and other third-party required notices are provided below.

**License:** Apache 1.1

**Module:** Crimson v1.1.1, Xalan J2

Copyright © 1999-2000 The Apache Software Foundation. All rights reserved.

Use of Crimson 1.1.1 and Xalan J2 within the product is governed by the following (Apache 1.1):

(1) Redistributions of source code must retain the above copyright notice, this list of conditions and the disclaimer below. (2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the disclaimer below in the documentation and/or other materials provided with the distribution. (3) The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. (4) Neither the component name nor Apache Software Foundation may be used to endorse or promote products derived from the software without specific prior written permission. (5) Products derived from the software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**License:** CoolServlets.com

**Module:** CS CodeViewer v1.0 (Sun JRE Component)

Copyright © 1999 by CoolServlets.com

Use of this module within the product is governed by the following:

(1) Redistributions of source code must retain the above copyright notice, this list of conditions and the disclaimer below. (2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the disclaimer below in the documentation and/or other materials provided with the distribution. (3) Neither the component name nor CoolServlets.com may be used to endorse or promote products derived from the software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY COOLSERVLETS.COM AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."

**License:** Justin Frankel, [justin@nullsoft.com](mailto:justin@nullsoft.com)

**Module:** NSIS 1.0j (Sun JRE Component)

Use of this module within the product is governed by the following:

(1) The origin of the module must not be misrepresented, and Oracle may not claim that it wrote the original software. If Oracle uses this module in a product, an acknowledgment in the product documentation is appreciated but not required. (2) Altered source versions of the module must be plainly marked as such, and

must not be misrepresented as being the original software. (3) The following notice may not be removed or altered from any source distribution: "Justin Frankel justin@nullsoft.com".

**License:** ICU4j License

**Module:** ICU4j

Copyright © 1995-2001 International Business Machines Corporation and others. All rights reserved.

Oracle may use the software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the software, and to permit persons to whom the software is furnished to do so, provided that the above copyright notice and the permission notice appear in all copies of the software and that both the above copyright notice and the permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

**License:** Info-ZIP

**Module:** INFO-ZIP ZIP32.DLL (Binary Form)

Copyright (c) 1990-2005 Info-ZIP. All rights reserved

Use of this dll within the product is governed by the following:

(1) Redistributions of source code must retain the above copyright notice, this list of conditions and the definition and disclaimer below. (2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the definition and disclaimer below in the documentation and/or other materials provided with the distribution. The sole exception to this condition is redistribution of a standard UnZipSFX binary (including SFXWiz) as part of a self-extracting archive; that is permitted without inclusion of this license, as long as the normal SFX banner has not been removed from the binary or disabled. (3) Altered versions--including, but not limited to, ports to new operating systems, existing ports with new graphical interfaces, and dynamic, shared, or static library versions--must be plainly marked as such and must not be misrepresented as being the original source. Such altered versions also must not be misrepresented as being Info-ZIP releases--including, but not limited to, labeling of the altered versions with the names "Info-ZIP" (or any variation thereof, including, but not limited to, different capitalizations), "Pocket UnZip," "WiZ" or "MacZip" without the explicit permission of Info-ZIP. Such altered versions are further prohibited from misrepresentative use of the Zip-Bugs or Info-ZIP e-mail addresses or of the Info-ZIP URL(s). (4) Info-ZIP retains the right to use the names "Info-ZIP," "Zip," "UnZip," "UnZipSFX," "WiZ," "Pocket UnZip," "Pocket Zip," and "MacZip" for its own source and binary releases.

[Definition]: For the purposes of this copyright and license, "Info-ZIP" is defined as the following set of individuals:

Mark Adler, John Bush, Karl Davis, Harald Denker, Jean-Michel Dubois, Jean-loup Gailly, Hunter Goatley, Ed Gordon, Ian Gorman, Chris Herborth, Dirk Haase, Greg Hartwig, Robert Heath, Jonathan Hudson, Paul Kienitz, David Kirschbaum, Johnny Lee, Onno van der Linden, Igor Mandrichenko, Steve P. Miller, Sergio Monesi, Keith Owens, George Petrov, Greg Roelofs, Kai Uwe Rommel, Steve Salisbury, Dave Smith, Steven M. Schweda, Christian Spieler, Cosmin Truta, Antoine Verheijen, Paul von Behren, Rich Wales, Mike White

[Disclaimer:] "This software is provided "as is," without warranty of any kind, express or implied. In no event shall Info-ZIP or its contributors be held liable for any direct, indirect, incidental, special or consequential damages arising out of the use of or inability to use this software."

**License:** Paul Johnston

**Modules:** md5.js

Copyright (C) Paul Johnston 1999 - 2002

Use of these modules within the product is governed by the following:

(1) Redistributions of source code must retain the above copyright notice, this list of conditions and the disclaimer below. (2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the disclaimer below in the documentation and/or other materials provided with the distribution. (3) Neither the component name nor the names of the copyright holders and contributors may be used to endorse or promote products derived from the software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**License:** Jef Poskanzer

**Modules:** DES, 3xDES (Sun JRE Components)

Copyright © 2000 by Jef Poskanzer <jef@acme.com>. All rights reserved

Use of these modules within the product is governed by the following:

(1) Redistributions of source code must retain the above copyright notice, this list of conditions and the disclaimer below. (2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the disclaimer below in the documentation and/or other materials provided with the distribution. (3) Neither the component name nor the name of Jef Poskanzer may be used to endorse or promote products derived from the software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

**License:** Sun Microsystems, Inc.

**Modules:** Sun Swing Tutorials

Copyright© 1995-2006 Sun Microsystems, Inc. All Rights Reserved.

Use of these modules within the product is governed by the following:

(1) Redistributions of source code must retain the above copyright notice, this list of conditions and the disclaimer below. (2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the disclaimer below in the documentation and/or other materials provided with the distribution. (3) Neither the component name nor the name of Sun Microsystems, Inc. and contributors may be used to endorse or promote products derived from the software without specific prior written permission. (4) Oracle must acknowledge that the software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility.

THIS SOFTWARE IS PROVIDED "AS IS," WITHOUT A WARRANTY OF ANY KIND. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. ("SUN") AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR

PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

**License:** Tom Wu

**Module:** jsbn library

Copyright © 2003-2005 Tom Wu. All rights reserved

Use of this module within the product is governed by the following:

(1) Redistributions of source code must retain the above copyright notice, this list of conditions and the disclaimer below. (2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the disclaimer below in the documentation and/or other materials provided with the distribution.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL TOM WU BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

---

---

# Contents

## Chapter 1

<b>Overview</b> .....	<b>1-1</b>
Configuration Overview.....	1-2
Receivables Management Configuration.....	1-2
Workflow Management Configuration.....	1-2
What is this book?.....	1-3
Oracle Utilities Billing Component Glossary.....	1-5

## Part One

<b>Receivables Management Configuration</b> .....	<b>1-i</b>
---	------------

## Chapter 2

<b>Introduction to the Oracle Utilities Receivables Component</b> .....	<b>2-1</b>
What the Oracle Utilities Receivables Component Does .....	2-2
The Oracle Utilities Receivables Component Database .....	2-2
The Oracle Utilities Receivables Component Messaging System .....	2-2
How the Oracle Utilities Receivables Component Is Used .....	2-2
The Financial Engine and the Financial Management Modules .....	2-3
The Financial Engine.....	2-3
Billing .....	2-3
Remittance.....	2-3
Maintenance .....	2-3
Reports.....	2-4
Collections.....	2-4
The Oracle Utilities Receivables Component User Interface .....	2-4

## Chapter 3

<b>The Oracle Utilities Receivables Component Database</b> .....	<b>3-1</b>
Account Data .....	3-2
Location Tables .....	3-3
Service Tables .....	3-3
Budget Tables .....	3-3
Meter Tables .....	3-4
Transaction Data .....	3-5
Journaling Data .....	3-9

## Chapter 4

<b>Setting Up and Configuring the Oracle Utilities Receivables Component Database</b> .....	<b>4-1</b>
Setting up the Oracle Utilities Receivables Component Database .....	4-2
Lookup Tables.....	4-2
Customer/Account Tables .....	4-5
Usage Data Tables .....	4-7
Translation Tables.....	4-7

## Chapter 5

<b>The Financial Engine .....</b>	<b>5-1</b>
Financial Engine Functions - Summary.....	5-2
Transactions.....	5-2
Credit Application.....	5-2
Journaling.....	5-2
Sub-Ledger Rollup.....	5-2
General Ledger Update.....	5-2
Balancing Controls.....	5-3
Transactions .....	5-4
Transaction Data.....	5-4
Transaction Types.....	5-8
Transaction Processing.....	5-9
Credit Application .....	5-13
Credit Application Processing for Charge or Credit Transaction - Immediate.....	5-13
Credit Application Processing for Charge or Credit Transaction - Specified .....	5-15
Credit Application Processing for Charge or Credit Transaction - Invoice ID.....	5-15
Credit Application Processing for Charge or Credit Transaction - Receivable Type.....	5-15
Credit Application Processing for Cancel Charge Transaction .....	5-16
Credit Application Processing for Cancel Credit Transaction.....	5-17
Journaling.....	5-18
Journaling Processing.....	5-18
Journal Translation Rules.....	5-19
Sub-Ledger Rollup .....	5-20
Sub-Ledger Roll-Up Processing.....	5-20
Using the Sub-Ledger Roll-Up Command Line Program .....	5-21
General Ledger Update .....	5-22
General Ledger Update Processing.....	5-22
Using the General Ledger Update Command Line Program .....	5-23
Balancing Controls .....	5-24
Transaction Balancing.....	5-24
Journal Balancing.....	5-25
Account Balancing.....	5-26

## Chapter 6

<b>Billing.....</b>	<b>6-1</b>
Billing Database Tables .....	6-2
Billing Functions.....	6-3
Billing Function Processing.....	6-3
Billing Functions .....	6-5
Deprecated Functions.....	6-14

## Chapter 7

<b>Remittance .....</b>	<b>7-1</b>
Remittance Database Tables.....	7-2
Payment Tables .....	7-2
Payment Assistance Tables.....	7-2
Remittance Functions .....	7-4
Payment Data .....	7-4
Remittance Function Processing.....	7-7

## Chapter 8

<b>Maintenance.....</b>	<b>8-1</b>
Maintenance Functions .....	8-2
Maintenance Data .....	8-2
Maintenance Function Processing.....	8-3



## Chapter 9

<b>Collections</b> .....	<b>9-1</b>
Collections Database Tables.....	9-2
Collections Agency Tables.....	9-2
Collection Arrangements Tables.....	9-2
Collection Exemptions Tables.....	9-3
Collection Message Tables.....	9-3
Other Collections Tables.....	9-5
Collections Arrangements.....	9-7
Creating Collections Arrangements.....	9-7
Collection Arrangement Data.....	9-7
Collections Arrangements Processing.....	9-7
Collection Exemptions.....	9-11
Creating Collections Exemptions.....	9-11
Collection Exemption Data.....	9-11
Reviewing Collections Exemptions.....	9-12
Collections Processing and Activities.....	9-14
Selecting Accounts for Collections.....	9-14
Collections Process Activities.....	9-18
Collection Activities Payload Information.....	9-33

## Chapter 10

<b>Reports</b> .....	<b>10-1</b>
Report Database Tables.....	10-2
Oracle Utilities Receivables Component Reports.....	10-3
Payment Posting Report.....	10-4
Issue Refund Report.....	10-6
Account Balance Report.....	10-8
AR Aging Report.....	10-10
G/L Activity Report.....	10-12
System Balance Report.....	10-14
Market Participant Aging Report.....	10-16
Running Oracle Utilities Receivables Component Reports.....	10-18
Viewing Oracle Utilities Receivables Component Reports.....	10-20
Creating Custom Reports.....	10-21

## Chapter 11

<b>Configuring Oracle Utilities Receivables Component Security</b> .....	<b>11-1</b>
Oracle Utilities Receivables Component Security.....	11-2
Financials Features.....	11-2
Important Notes about Assigning Oracle Utilities Receivables Component Permissions.....	11-3

## Part Two

<b>Workflow Management Configuration</b> .....	<b>1-i</b>
--	------------

## Chapter 12

<b>Setting Up Workflow Management Database Tables</b> .....	<b>12-1</b>
COM Object Tables.....	12-2
COM Object Type Table.....	12-2
COM Object Table.....	12-2
Process Context Tables.....	12-3
Variable Source Table.....	12-3
Context Value Table.....	12-3
Process Context Value Table.....	12-3

## Chapter 13

<b>The Workflow Engine</b> .....	<b>13-1</b>
Workflow Engine Functions .....	13-2
Workflow Engine Components .....	13-3
Workflow Engine API .....	13-3
Workflow Engine Message Queue.....	13-3
Workflow Engine Executable.....	13-5
Workflow Engine Processing.....	13-6
Start Process.....	13-6
Navigate Process .....	13-6
Start Activity .....	13-7
Terminate Activity .....	13-7
Suspend Process.....	13-8
Resume Process.....	13-8
Terminate Process.....	13-8
Activity Finished .....	13-8
Activity Expired .....	13-8
Activity In Error.....	13-9
Activity Event.....	13-9
Workflow Function Activities .....	13-10

## Chapter 14

<b>The Workflow Scheduler</b> .....	<b>14-1</b>
Workflow Scheduler Functions and Processing.....	14-2
Workflow Scheduler Components .....	14-3
Workflow Scheduler API.....	14-3
Workflow Scheduler Executable .....	14-3

## Chapter 15

<b>The Rules Language Engine</b> .....	<b>15-1</b>
Rate Form Activities and the Rules Language Engine .....	15-2
Rules Language Engine Components .....	15-3
Rules Language Engine Message Queue.....	15-3
Rules Language Execution Engine Executable.....	15-4
Creating Rate Forms for use with the Rules Language Engine .....	15-5
No Required Input Data.....	15-5
Input Data from Context.....	15-5

## Part Three

<b>Receivables Management Interfaces</b> .....	<b>1-i</b>
--	------------

## Chapter 16

<b>Oracle Utilities Receivables Component Financial Engine Interface</b> .....	<b>16-1</b>
Methods, Interfaces, and Syntax .....	16-2
Interface Arguments .....	16-3
Input Values .....	16-4
xmlAccount.....	16-4
xmlTransaction.....	16-6
xmlGenLedgerParams.....	16-12
xmlJournalParams .....	16-13
xmlQuery.....	16-13
Return Values.....	16-14
xmlGenLedgerFile .....	16-14

## Chapter 17

<b>Oracle Utilities Receivables Component Billing Interface .....</b>	<b>17-1</b>
Methods, Interfaces, and Syntax .....	17-2
Interface Arguments .....	17-7
Input Values .....	17-8
xmlAcctTrans .....	17-8
xmlInstallmentPlan .....	17-10
xmlAccountIn .....	17-12
Return Values .....	17-14
xmlAccountOut .....	17-14
Deprecated Methods .....	17-17
XML Examples .....	17-19

## Chapter 18

<b>Oracle Utilities Receivables Component Remittance Interface .....</b>	<b>18-1</b>
Methods, Interfaces, and Syntax .....	18-2
Interface Arguments .....	18-4
Input Values .....	18-5
xmlPayment .....	18-5
xmlBatchPayment .....	18-9
xmlPaymentFile .....	18-11
xmlAcctBill .....	18-13

## Chapter 19

<b>Oracle Utilities Receivables Component Maintenance Interface .....</b>	<b>19-1</b>
Methods, Interfaces, and Syntax .....	19-2
Interface Arguments .....	19-8
Input Values .....	19-9
xmlTransaction .....	19-9
xmlBatchRefund .....	19-10
xmlBatchTransaction .....	19-13
xmlTransactionFile .....	19-14

## Chapter 20

<b>Oracle Utilities Receivables Component Collections Interface .....</b>	<b>20-1</b>
Method, Interface, and Syntax .....	20-2
Interface Arguments .....	20-8
Input Values .....	20-10
xmlColArrangementIn .....	20-10
xmlAccount .....	20-11
xmlExemptionIn .....	20-12
xmlArrangementIn .....	20-13
xmlPaymentIn .....	20-15
Return Values .....	20-17
xmlColArrangementOut .....	20-17
xmlContext .....	20-17
xmlCollInfo .....	20-17
xmlTransactionOut .....	20-18
xmlAccountOut .....	20-18
xmlExemptionOut .....	20-18
xmlArrangementOut .....	20-19
xmlPaymentOut .....	20-21

## Chapter 21

<b>Messaging Interface</b> .....	<b>21-1</b>
Methods, Interfaces, and Syntax .....	21-2
Interface Arguments .....	21-5
Input Values .....	21-6
xmlMessageIn .....	21-6
xmlMessageListIn .....	21-8
Return Values .....	21-10
xmlMessageOut/xmlMessageListOut .....	21-10

## Part Four

<b>Workflow Management Interfaces</b> .....	<b>1-i</b>
---	------------

## Chapter 22

<b>Workflow Management Activity Implementations Interface</b> .....	<b>22-1</b>
Methods, Interfaces, and Syntax .....	22-2
Interface Arguments .....	22-4
Input Values .....	22-5
xmlActivityImplementationIn.....	22-5
xmlMessageListIn .....	22-5
Return Values .....	22-7
xmlActivityImplOut/xmlActivityImplListOut .....	22-7

## Chapter 23

<b>Workflow Management Process Versions Interface</b> .....	<b>23-1</b>
Methods, Interfaces, and Syntax .....	23-2
Interface Arguments .....	23-7
Input Values .....	23-9
xmlProcessVersionIn .....	23-9
xmlProcessVersionListIn.....	23-10
xmlProcessIn .....	23-10
xmlProcessListIn.....	23-11
xmlProcessActivityIn.....	23-11
xmlProcessActivityListIn.....	23-13
Return Values .....	23-14
xmlProcessVersionOut/xmlProcessVersionListOut .....	23-14
xmlProcessOut/xmlProcessListOut.....	23-14
xmlProcessActivityOut/xmlProcessActivityListOut .....	23-14

## Chapter 24

<b>Workflow Management Process Instance Interface</b> .....	<b>24-1</b>
Methods, Interfaces, and Syntax .....	24-2
Interface Arguments .....	24-5
Input Values .....	24-7
xmlProcessInstanceIn .....	24-7
xmlProcessInstanceListIn.....	24-8
xmlActivityInstanceIn .....	24-8
xmlActivityInstanceListIn .....	24-9
xmlActivityEvent .....	24-9
Return Values .....	24-10
xmlProcessInstanceOut/xmlProcessInstanceListOut .....	24-10
xmlActivityInstanceOut/xmlActivityInstanceListOut .....	24-10

## Part Five

Appendices.....	1-i
-----------------	-----

### Appendix A

Oracle Utilities Data Repository Receivables Component Database Schema .....	A-1
Oracle Utilities Receivables Component Database Schema.....	A-2
Oracle Utilities Receivables Component-Collections Database Schema.....	A-3

### Appendix B

Oracle Utilities Data Repository Workflow Management Database Schema .....	B-1
Oracle Utilities Billing Component - Workflow Management / Reports Database Schema.....	B-2

### Appendix C

Messaging.....	C-1
The Purpose of the Messaging System .....	C-2
Messaging Tables.....	C-3
Message Types .....	C-4
Message Queues .....	C-5
Default Message Queues.....	C-5
Message Queue Table Templates .....	C-6
Messaging Functions.....	C-7
How the Messaging Functions Work .....	C-8

### Appendix D

Financial Management Rules Language Statements.....	D-1
Using the Financial Management Statements .....	D-2
Using User-Defined Attributes .....	D-6
Post Charge Or Credit Statement.....	D-7
Post Tax Statement.....	D-9
Post Installment Statement.....	D-11
Post Statement Statement.....	D-13
Post Bill Statement.....	D-15
Post Payment Statement .....	D-17
Post Adjustment Statement.....	D-19
Post Refund Statement.....	D-21
Post Writeoff Statement .....	D-23
Post Deposit Statement .....	D-25
Post Deposit Interest Statement.....	D-27
Post Deposit Application Statement.....	D-29
Cancel Transaction Statement.....	D-31
CALCULATE_LATEPAYMENT Function .....	D-33
FMGETBILLINFO Function.....	D-34
PROCESSAUTOPAYMENT Function.....	D-35
Deprecated Statements .....	D-36
Post Service Charge Statement .....	D-36
Post Deferred Service Charge Statement .....	D-38
Post Budget Service Charge Statement.....	D-40
Post Budget Bill Charge Statement .....	D-42
Post Budget Bill Trueup Statement.....	D-44
Post Installment Charge Statement.....	D-46

### Appendix E

Workflow Management Rules Language Statements .....	E-1
Using the Workflow Management Statements .....	E-2
Process Start Statement.....	E-3
Process Suspend Statement.....	E-5
Process Resume Statement.....	E-7

Process Terminate Statement.....	E-9
Process Event Statement .....	E-11

## Appendix F

<b>XML Rules Language Statements and Functions .....</b>	<b>F-1</b>
XML Overview.....	F-2
XML Data Types .....	F-2
Using Stem.Tail XML Identifiers .....	F-3
XML Statements.....	F-4
Identifier Statement .....	F-4
OPTIONS Statement.....	F-5
XML_ELEMENT Statement.....	F-6
FOR EACH x IN XML_ELEMENT_OF 0 Statement.....	F-8
XML_OP Statement.....	F-9
XML/Document Object Management Functions.....	F-12
DOMDOCCREATE Function.....	F-13
DOMDOCLOADFILE Function.....	F-14
DOMDOCLOADXML Function.....	F-15
DOMDOCSAVEFILE Function .....	F-16
DOMDOCGETROOT Function .....	F-17
DOMDOCADDPI Function .....	F-18
DOMNODEGETNAME Function .....	F-19
DOMNODEGETTYPE Function .....	F-20
DOMNODEGETVALUE Function .....	F-21
DOMNODEGETCHILDCT Function.....	F-22
DOMNODEGETFIRSTCHILD Function .....	F-23
DOMNODEGETSIBLING Function.....	F-24
DOMNODECREATECHILDELEMENT Function .....	F-25
DOMNODESETATTRIBUTE Function.....	F-26
DOMNODEGETCHILDELEMENTCT Function.....	F-27
DOMNODEGETFIRSTCHILDELEMENT Function.....	F-28
DOMNODEGETSIBLINGELEMENT Function.....	F-29
DOMNODEGETATTRIBUTECT Function.....	F-30
DOMNODEGETATTRIBUTEI Function.....	F-31
DOMNODEGETATTRIBUTEBYNAME Function .....	F-32
DOMNODEGETBYNAME Function .....	F-33
Using the XML Statements and Functions .....	F-34
Reading from XML Documents and Files.....	F-34
Creating XML Documents and Files.....	F-35

## Index

# Chapter 1

---

## Overview

This chapter provides an overview of the configuration of the receivables management and workflow management functionality of Oracle Utilities Billing Component, including:

- **Configuration Overview**
- **What is this book?**

## Configuration Overview

Configuring the receivables management and workflow management functionality of Oracle Utilities Billing Component involves the following steps:

### Receivables Management Configuration

- Set up Oracle Utilities Receivables Component database records as described in **Chapter 4: Setting Up and Configuring the Oracle Utilities Receivables Component Database**.
- Set up and configure any rate schedules needed by Oracle Utilities Receivables Component using the Oracle Utilities Rules Language, including the financial management statements described in **Appendix D: Financial Management Rules Language Statements**.
- Set up and configure reports to run as described in **Chapter 10: Reports**.
- Set up and configure security for use with Oracle Utilities Receivables Component as described in **Chapter 11: Configuring Oracle Utilities Receivables Component Security**.

### Workflow Management Configuration

- Set up and configure workflow management database records as described in **Chapter 12: Setting Up Workflow Management Database Tables**.
- Set up and configure activity implementations, processes, process versions, and process activities using the workflow management user interface. See the **Part Four: Workflow Management** in the *Oracle Utilities Billing Component User's Guide* for more information.
- Set up and configure any rate schedules needed by the workflow processes using the Oracle Utilities Rules Language, including the workflow management statements described in **Appendix E: Workflow Management Rules Language Statements** and XML statements and functions described in **Appendix F: XML Rules Language Statements and Functions**.
- Set up and configure security for use with workflow management as described in **Chapter 11: Configuring Oracle Utilities Receivables Component Security**.



---

## What is this book?

This book describes how to configure the receivables management and workflow management functionality of Oracle Utilities Billing Component. Configuration of the billing and contract management functionality of Oracle Utilities Billing Component is described in Volume 1. This book includes the following:

- **Chapter 1: Overview** (this chapter) provides an overview of the configuration process for the receivables management and workflow management functionality of Oracle Utilities Billing Component.
- **Chapter 2: Introduction to the Oracle Utilities Receivables Component** provides an overview of the receivables management functionality of Oracle Utilities Billing Component.
- **Chapter 3: The Oracle Utilities Receivables Component Database** describes the database tables primarily used receivables management functionality of Oracle Utilities Billing Component.
- **Chapter 4: Setting Up and Configuring the Oracle Utilities Receivables Component Database** describes how to set up database records used by Oracle Utilities Receivables Component
- **Chapter 5: The Financial Engine** describes the financial engine and core functions used by Oracle Utilities Receivables Component.
- **Chapter 6: Billing** describes the billing functions of Oracle Utilities Receivables Component.
- **Chapter 7: Remittance** describes the remittance (payment) functions of Oracle Utilities Receivables Component.
- **Chapter 8: Maintenance** describes the maintenance functions of Oracle Utilities Receivables Component.
- **Chapter 9: Collections** describes the collections functions of Oracle Utilities Receivables Component.
- **Chapter 10: Reports** describes the collections functions of Oracle Utilities Receivables Component.
- **Chapter 11: Configuring Oracle Utilities Receivables Component Security** describes how to configure security for use with the receivable management functionality of Oracle Utilities Billing Component.
- **Chapter 12: Setting Up Workflow Management Database Tables** describes how to set up database records used by the workflow management functionality of Oracle Utilities Billing Component.
- **Chapter 13: The Workflow Engine** describes the workflow engine used by the workflow management functionality of Oracle Utilities Billing Component.
- **Chapter 14: The Workflow Scheduler** describes the workflow scheduler used by the workflow management functionality of Oracle Utilities Billing Component.
- **Chapter 15: The Rules Language Engine** describes the Rules Language engine used by the workflow management functionality of Oracle Utilities Billing Component.
- **Chapter 16: Oracle Utilities Receivables Component Financial Engine Interface** describes the Financial Engine interface, used to invoke the basic receivables management functions of Oracle Utilities Receivables Component.
- **Chapter 17: Oracle Utilities Receivables Component Billing Interface** describes the Billing interface, used to invoke the billing functions of Oracle Utilities Receivables Component.

- **Chapter 18: Oracle Utilities Receivables Component Remittance Interface** describes the Remittance interface, used to invoke the remittance/payment functions of Oracle Utilities Receivables Component.
- **Chapter 19: Oracle Utilities Receivables Component Maintenance Interface** describes the Maintenance interface, used to invoke the maintenance functions of Oracle Utilities Receivables Component.
- **Chapter 20: Oracle Utilities Receivables Component Collections Interface** describes the Collections interface, used to invoke the collections functions of Oracle Utilities Receivables Component.
- **Chapter 21: Messaging Interface** describes the Messaging interface, used to invoke the messaging functions of Oracle Utilities Receivables Component (including workflow management).
- **Chapter 22: Workflow Management Activity Implementations Interface** describes the Activity Implementation interface, used to invoke workflow management activity processing functions.
- **Chapter 23: Workflow Management Process Versions Interface** describes the Process Versions interface, used to invoke workflow management process and process version functions.
- **Chapter 24: Workflow Management Process Instance Interface** describes the Process Instance interface, used to invoke workflow management processing functions.
- **Appendix A: Oracle Utilities Data Repository Receivables Component Database Schema** provides a database schema diagram of the tables in the Oracle Utilities Data Repository used by the receivables management functionality of Oracle Utilities Billing Component.
- **Appendix B: Oracle Utilities Data Repository Workflow Management Database Schema** provides a database schema diagram of the tables in the Oracle Utilities Data Repository used by the workflow management functionality of Oracle Utilities Billing Component.
- **Appendix C: Messaging** describes the messaging functions used by the receivables management and workflow management functionality of Oracle Utilities Billing Component.
- **Appendix D: Financial Management Rules Language Statements** describes Rules Languages statements used by the receivables management functionality of Oracle Utilities Billing Component.
- **Appendix E: Workflow Management Rules Language Statements** describes Rules Languages statements used by the workflow management functionality of Oracle Utilities Billing Component.
- **Appendix F: XML Rules Language Statements and Functions** describes XML Rules Language statements and functions used by the WorkFlow Manager application.

# Oracle Utilities Billing Component Glossary

This appendix contains a glossary of terms used in Oracle Utilities Billing Component and other Oracle Utilities product documentation.

Term	Definition
<b>Available</b>	An account is “available” for billing if it is “eligible” and if all data required to compute its bill is available to Oracle Utilities Billing Component.
<b>Bill Determinant</b>	A value used to compute a customer bill (e.g., kWh). Also referred to as <i>billing determinant</i> .
<b>Billing Mode Flag</b>	A value in the Oracle Utilities Data Repository -- assigned at the account-, rate code-, or rate-schedule level -- that determines how an account is billed (Approval Required, Fully Automatic, or Manual Start).
<b>Bill Month</b>	Revenue month; the accounting month that the income from a bill is assigned to.
<b>Channel</b>	An element on an interval data recorder assigned to measure a specific end use or other quantity of interest at a specific unit of measure.
<b>Channel Group</b>	Any user-specified combination of interval data channels whose data you want to total for billing or other purposes.
<b>CIS Account</b>	<i>(Note: your company may or may not use CIS accounts.)</i> In some Customer Information Systems, all billing entities are identified by an account ID—for example, customers, accounts, account service locations, or others. Oracle Utilities Billing Component, on the other hand, recognizes these entities as different levels of the billing hierarchy, and assigns them different ids accordingly. This makes it possible for Oracle Utilities Billing Component to calculate usage and charges for flexible grouping of entities. In Oracle Utilities Billing Component, the CIS Account Table is an optional table that lets you “map” entities recognized by your CIS system to entities recognized by Oracle Utilities Billing Component.
<b>Contract</b>	A rate form that contains special Rules Language statement specific to an account.
<b>Cut</b>	A data record containing interval values for one recorder channel, for the period between read dates.
<b>Eligible</b>	An account is “eligible” for billing if today's date falls within a user-defined number of days before and after the account's scheduled meter read date.
<b>Group</b>	<i>See Channel Group.</i>
<b>IPH</b>	Intervals per hour; reciprocal of minutes per interval (MPI).
<b>Interval Data</b>	A series of values that represents a customer' demand (kW) or other quantity measured on a periodic basis, such as every 5, 15, 30, or 60 minutes. Also referred to as <i>load data</i> or <i>time-series data</i> .
<b>MPI</b>	Minutes per interval; reciprocal of intervals per hour (IPH).

<b>Term</b>	<b>Definition</b>
<b>Override</b>	A modification to the tariffs normally used to compute an account's bill. An override might be long-term, such as an “economic development discount,” or it might be short-term, coinciding with a “special event” such as an interruption or maintenance period.
<b>Post-window</b>	Number of days after an account's scheduled read date that Oracle Utilities Billing Component's Automatic and Approval Required modules continue scanning for the account's billing data in order to bill it.
<b>Pre-window</b>	Number of days before an account's scheduled read date that Oracle Utilities Billing Component's Automatic and Approval Required modules begin scanning for the accounts billing data in order to bill it.
<b>Rate Form</b>	The generic term applied to a contract, rate schedule, or rider.
<b>Rate Schedule</b>	The rate form that Oracle Utilities Billing Component uses to compute the bill for an account; may include contracts and/or riders.
<b>Read Date</b>	Date that an account's meter is scheduled to be read. It is determined by the billing cycle code assigned to the account. Oracle Utilities Billing Component stores the read date in the Billing History record, to associate the bill with the correct billing cycle.
<b>Recorder</b>	A device used to capture and store interval data. The data is retrieved either manually or by telecommunications means.
<b>Rider</b>	A set of Rules Language statements that can be included in another rate form, avoiding the necessity of having to re-create often-used sets of statements.
<b>Special Events</b>	An event which causes alternative charges (overrides) to be applied to an account's bill. In Oracle Utilities Billing Component, the terms <i>special events</i> and <i>overrides</i> are used somewhat interchangeably.
<b>Tariff Rider</b>	A tariff rider that a billing analyst or other user has explicitly attached to an account via the Account Rider History Table.
<b>UOM</b>	Unit of Measure.

# Part One

---

## Receivables Management Configuration

Part One describes configuration of the receivables management functionality of Oracle Utilities Billing Component (also known as Oracle Utilities Receivables Component), and contains the following chapters:

- **Chapter 2: Introduction to the Oracle Utilities Receivables Component**
- **Chapter 3: The Oracle Utilities Receivables Component Database**
- **Chapter 4: Setting Up and Configuring the Oracle Utilities Receivables Component Database**
- **Chapter 5: The Financial Engine**
- **Chapter 6: Billing**
- **Chapter 7: Remittance**
- **Chapter 8: Maintenance**
- **Chapter 9: Collections**
- **Chapter 10: Reports**
- **Chapter 11: Configuring Oracle Utilities Receivables Component Security**

---

# Chapter 2

---

## Introduction to the Oracle Utilities Receivables Component

This chapter provides an overview of Oracle Utilities Receivables Component. This includes:

- **What the Oracle Utilities Receivables Component Does**
- **The Financial Engine and the Financial Management Modules**

## What the Oracle Utilities Receivables Component Does

The Oracle Utilities Receivables Component performs a number of financial management and accounts receivable functions, allowing users to process billing, remittance, adjustments, transfers, writeoffs, refunds, and collections transactions.

At the core of Oracle Utilities Receivables Component is the Financial Engine, which is primarily responsible for credit application and journaling. Transactions processed by the Financial Engine come from a number of modules, each specific to a financial management area. These modules are described in more detail in **The Financial Engine and the Financial Management Modules** below.

The Oracle Utilities Receivables Component also provides interfaces for other systems, including the Oracle Utilities Receivables Component User Interface, batch processes, an Event Management System (EMS)/Enterprise Application Integration (EAI) systems, a Web Server, and other Oracle Utilities products, such as Oracle Utilities Billing Component or the Oracle Utilities Load Profiling and Settlement. These can trigger functions in one or more of the financial management modules, which in turn trigger the Financial Engine. See **Part Three: Receivables Management Interfaces** for more information about the Oracle Utilities Receivables Component interfaces.

## The Oracle Utilities Receivables Component Database

The transactions processed by the financial management modules are recorded in a set of database tables in the Oracle Utilities Data Repository. These tables store the specific account data, transaction data, and journaling data. The database tables used by Oracle Utilities Receivables Component are described in **Chapter 3: The Oracle Utilities Receivables Component Database**.

## The Oracle Utilities Receivables Component Messaging System

The Oracle Utilities Receivables Component includes a messaging system, designed to support external interfacing, control activities, work queues, and internal messages (such as collections processing). The Oracle Utilities Receivables Component Messaging System is described in **Appendix C: Messaging**.

## How the Oracle Utilities Receivables Component Is Used

As noted above, the Oracle Utilities Receivables Component allows users to process billing, remittance, adjustments, transfers, writeoffs, refunds, and other financial transactions.

Several of the Oracle Utilities Receivables Component functions operate 'behind the scenes', meaning that the functions are triggered automatically when certain operations are performed by external systems interfaced with the Financial Engine through one of the Oracle Utilities Receivables Component modules. For instance, when a charge or credit transaction is posted from a billing or remittance system, the Financial Engine creates a record in the Transaction Table in the Oracle Utilities Data Repository. Depending on the specifics of that transaction, it then creates an appropriate journal transaction record in another table that is later rolled up into a subledger account. See **Chapter 5: The Financial Engine** for more information about how the AR Engine processes transactions.

Several Oracle Utilities Receivables Component functions can be performed through the Oracle Utilities Billing Component user interface, a web-enabled interface that allows users to process adjustments, write-offs, refunds, and other functions (see the *Oracle Utilities Billing Component User's Guide* for more information).



---

# The Financial Engine and the Financial Management Modules

This section describes the Financial Engine and the modules that comprise the Oracle Utilities Receivables Component. Each is further described in later chapters.

## The Financial Engine

The Financial Engine is the central component in the Oracle Utilities Receivables Component. Refer to **Chapter 5: The Financial Engine** for more information about how the Financial Engine processes transactions, performs credit application, and performs journaling.

## Billing

The Billing module provides functions to post service charges, budget bill charges, and installment charges. Refer to **Chapter 6: Billing** for more information.

## Remittance

The Remittance modules provides interfaces to receive payments from multiple sources (lockbox, direct debit, and credit cards). During the payment processing, the Remittance module performs validations and exception handling for each payment, generating work queue items for unpostable payments. Refer to **Chapter 7: Remittance** for more information.

## Maintenance

The Maintenance module allows users to perform various maintenance operations on accounts, including transfers, adjustments, refunds and write-offs. Each of these is described in more detail below. Refer to **Chapter 8: Maintenance** for more information. Most of the Maintenance module functions can be performed using the Oracle Utilities Receivables Component user interface, or from external systems connected to Oracle Utilities Receivables Component through an interface.

### Transfers

The Transfers functions are used to transfer payments from one account to another. When a transfer is processed, the Transfers module triggers the transfer function in the Financial Engine, cancelling the transaction on the old (transferred from) account and creating a new transaction for the new (transferred to) account. The specific steps performed by the Financial Engine when processing a transfer are described in **Chapter 5: The Financial Engine**.

### Adjustments

Companies sometimes need to make account adjustments to charge or credit customers for non-recurring events, including marketing incentive programs, concession credits, or billing errors that could not be corrected through cancellation and rebilling. The Adjustments functions encompass the financial and customer account management actions necessary to post adjustments to customer accounts.

### Refunds

The Refunds functions are used to post and cancel refund transactions against an account, as well as update the status of refunds in the system.

### Write-Offs

The Write-Offs module is used to write off accounts that are deemed uncollectible. When a write-off is processed for a given account, any outstanding charges for that account are cancelled through the Financial Engine.

## Reports

Oracle Utilities Receivables Component also provides the capability to produce a number of reports related to various types of activities performed by Oracle Utilities Receivables Component, including payment posting, issuance of refunds, account balances, accounts receivable aging, and general ledger activity. Refer to **Chapter 10: Reports** for more information.

## Collections

Collections encompasses the actions required to collect outstanding receivables. The Collections module enables identification of delinquent accounts and tracking of the collection arrangements and exemptions, as well as performing collections processing. Refer to **Chapter 9: Collections** for more information.

## The Oracle Utilities Receivables Component User Interface

As noted earlier, a number of the Oracle Utilities Receivables Component functions can be performed through the Oracle Utilities Receivables Component user interface. This application is a web-based application used to perform system configuration, access business rule definition information, and access account, transaction, ledger and journal, and payment and collections data and functionality. It is also used to perform various financial transactions, including adjustments, write-offs, and transfers, as well as managing work queues. The *Oracle Utilities Billing Component User's Guide* contains more information concerning the user interface.

# Chapter 3

---

## The Oracle Utilities Receivables Component Database

This chapter provides an overview and descriptions of several of the database tables in the Oracle Utilities Data Repository that are used by the Oracle Utilities Receivables Component. The tables described in this chapter relate to the core of most of the functions of Oracle Utilities Receivables Component and the Financial Engine. Other tables used by specific modules, such as Billing, Remittance, etc., are described in later chapters.

## Account Data

Account data is data that defines specific accounts and how transactions related to specific accounts are processed by Oracle Utilities Receivables Component and the Financial Engine.

### Account Table

The Account Table stores information about individual accounts. An account is the smallest billable entity in the system. Account entities typically are the root of virtually all activity in the system. An account receives a single bill statement from its billing operating company per billing period. Specific information relating to the Oracle Utilities Receivables Component stored in the Account Table includes:

- The Account's Billing Address (from the Address Table), and
- Region (the region associated with the account, from the Region Table).
- Currency (the currency associated with the account, from the LS Currency table).

### Account FME Table

The Account FME Table stores information about individual accounts that is maintained by the Oracle Utilities Receivables Component. Specific information relating to the Oracle Utilities Receivables Component stored in the Account FME Table includes:

- Unique ID of the Account,
- Last Balance Time (the date and time the account was last balanced),
- Last Account Balance (the account's balance as of the last balance time. When the account is balanced, the amount in the Activity field is added to the Balance field.),
- New Activity (the sum of the financial activity for the account since the account was last balanced),
- Transferred To Account (applicable if the account balance has been transferred to another account),
- Receivable Status (CURRENT, PAST DUE, COLLECTIONS, UNCOLLECTIBLE, or UNREFUNDABLE),
- Write-Off Reason (from the Write-Off Reason Table. This field is applicable only if the Receivable Status is "UNCOLLECTIBLE" or "UNREFUNDABLE". Otherwise this should be NULL)
- Last Transaction Number (the last transaction number used by the account. This is linked to the most recent Transaction number in the Transaction Table; see below), and
- Allow Refund (a flag that indicates if automatic refunds are allowed for the account).

### Note

Values in the Last Balance, Last Balance Time, New Activity, and Last Transaction No. columns should be created **only** by the Financial Engine, and should **not** be inserted or updated manually.

### Auto Payment Plan Table

Records in the Auto Payment Plan Table represent automatic payment plans for a given account for a specified time period. The existence of a current automatic payment plan for an account triggers an automatic draft of the specified automatic payment vendor account when the account is billed. More information concerning automatic payments can be found in **Chapter 7: Remittance**.

---

## Write-Off Reason Table

The Write-Off Reason Table stores reasons for writing off an account, such as “Sold”, “Deceased”, “Bankrupt”, or “Bad Debt”. This is a lookup table used by the Account Table. If the “Write-Off Reason” column in the Account Table is populated, the Account’s Receivable Status should be “UNCOLLECTIBLE” or “UNREFUNDABLE”.

## Location Tables

Location tables store information about the physical location of accounts and services. These include the Address, Premise, and Region tables.

### Address Table

The Address Table stores physical address information. Records in this table are used in the Premise Table.

### Premise Table

The Premise Table stores the physical location of service. This is typically a building, or a location within a building.

### Region Table

The Region Table stores additional geographical attributes that are optionally used by accounts.

## Service Tables

The Service tables describe the types of services provided to accounts, and the relationships among accounts, services, and locations.

### Service Type Table

The Service Type Table is used to define all service-oriented businesses or commodities provided by the client operating company. Typical service types might include “Electric”, “Gas”, or “Cable”.

### Service Plan Table

The Service Plan Table associates an account with a service type. An account may have several services active at any one time.

## Budget Tables

The Budget tables store data related to budget types and budget plans used by accounts.

### Budget Type

Records in the Budget Type Table represent specific types of budget plans recognized by the application. The application uses budget types to determine how true-ups should be handled.

### Budget Plan

Records in the Budget Plan Table indicate that a given account is on a particular budget plan for a specific time period.

## Meter Tables

The Meter tables store usage data for scalar meters and the relationships between meters and accounts.

### **Meter Table**

Records in the Meter Table represent scalar meters in the system.

### **Meter Read Table**

Records in the Meter Read Table represent readings for a meter.

### **Meter History Table**

Records in the Meter History Table represent snapshots of a meter's use and related information. The Meter History Table associates meters with accounts.

## Transaction Data

Transaction data is at the heart of the functionality of the Oracle Utilities Receivables Component and the Financial Engine. The following tables contain data concerning transactions and how those transactions are processed by the Financial Engine.

### Note

Records in the Transaction and Credit Application Tables should be created **only** by the Financial Engine.

### Transaction ID Table

The Transaction ID Table is used to configure different adjustment transaction types, refund transaction types, etc. This table is configurable and can accommodate various financial transaction IDs based on specific requirements. At least one Transaction ID is required for each Transaction Type (see Transaction Type Table, below).

### Transaction Type Table

Records in the Transaction Type Table represent different types of financial transactions recognized by the application, including the following:

Transaction Type	Code
Charge or Credit	CHRGCRDT
Tax	TAX
Installment	INST
Statement	STMT
Bill	BILL
Payment	PAYMENT
Deposit	DEPOSIT
Deposit Interest	DEPINT
Deposit Application	DEPAPP
Adjustment	ADJ
Refund	RFND
Write-Off	WRTOFF
Balance Transfer	BALXFR

The following transaction types have been effectively replaced by the Charge or Credit (CHRGCRDT) transaction type, but are still available for users of previous versions of Oracle Utilities Receivables Component.

Transaction Type	Code
Service Charge	SRVCHG
Deferred Service Charge	DEFSRVCHG
Budget Service Charge	BDGTSRVCHG
Budget Bill Charge	BDGTCHG

Transaction Type	Code
Budget Bill True-up	BDGTTRP
Installment Charge	INSTCHG

**Note:** While additional Transaction Types can be added to the Transaction Type Table, types on the list provided should NOT be deleted.

### LSTransaction Table

Records in the LSTransaction Table represent financial transactions for an account. Each transaction represents a single unit of work that occurs at a point in time and that may be cancelled as a whole.

Some of the specific data stored in the LSTransaction Table includes:

- Unique ID for the transaction,
- Account ID (the parent account of the transaction),
- Transaction Number (unique number for the transaction within the account),
- Transaction ID and Type, and Time,
- Revenue Month (the month to which any revenue associated with the transaction applies),
- User ID (user identification associated with the transaction),
- Application Method (indicates the credit application method)
- Debit and Credit Account ID (optional debit and credit journal account IDs associated with this transaction),
- Cost Center ID (optional cost center ID associated with this transaction),
- Cancel Time, Cancel Revenue Month, Cancel User ID, Cancel Reason, Cancel Note , Cancel Debit Account ID, Cancel Credit Account ID, and Cancel Cost Center ID (time, revenue month, user ID, reason for the cancellation of the transaction, an optional note, optional debit/credit accounts, and cost center associated with the cancellation of the transaction),
- Charge or Credit (whether the transaction is a charge or credit),
- Amount (the amount of the transaction, including the currency code for the transaction. In the case of a consumption charge, the amount is the amount of consumption),
- Balance (outstanding balance of charge or credit, including the currency code for the balance. The Balance is NULL if the amount does not affect the balance of the account, such as in the case of consumption charges, and deferred or budget service charges),
- Billed Or Paid Date (date the charge was billed or credit paid),
- Due Date (data that payment for a charge is due),
- Receivable Type (from the Receivable Type Table; see below),
- Charge Type (from the Charge Type Table; see below),
- Rate Form, Rate Code, Operating Company, and Jurisdiction,
- Statement Date (optional date of statement associated with the transaction),
- Invoice ID and Date (optional ID and date of invoice associated with the transaction),
- Cancel Statement Date (optional date of statement associated with cancelling the transaction),
- Cancel Invoice ID and Cancel Invoice Date (optional ID and date of invoice associated with cancelling the transaction),
- Bill Cycle Date (optional date of bill cycle associated with the transaction),



- Unique ID of associated Bill History Record (from the Bill History Table),
- Unique ID of the subaccount from which the transaction originated (from the Account Table),
- Transferred To Transaction/Transferred From Transaction (for transactions that have been transferred from one account to another),
- Payment, Service Plan, Budget Plan (associated Payment, Service/Budget Plan records),
- Installment Plan (installment plan associated with deferred charge or installment transactions),
- Deposit (unique ID of deposit associated with Deposit, Deposit Interest, Deposit Application, or Installment transactions),
- Unique ID of batch in which the transaction was posted (from Batch Transaction table), and
- Unique ID of the batch in which the transaction was cancelled (from the Batch Transaction table).

### Batch Transaction Table

Records in this table represent a batch of transactions, each of which contains one or more transactions on multiple accounts. Data stored in the Batch Payment Table includes:

- A unique ID for the batch transaction,
- Transaction File (a unique pathname of the transaction file that includes this batch),
- Batch No (unique number that identifies the batch within the transaction file),
- Date (the date on which the batch was received),
- Number of Transactions (the total number of transactions in the batch),
- Amount (the total amount of the batch transaction, including the currency code for the transactions), and
- Cancel Time (the date/time at which the batch was cancelled).

### Cancel Reason Table

Records in the Cancel Reason Table represent types of reasons for which a transaction might be cancelled.

### Receivable Type Table

Records in the Receivable Type Table represent types of receivables to which a financial transaction amount might apply. Typical receivable types might include “Disco Electric” or “State Tax”. Payments on an account are applied to receivable types based on the application priority of the receivable type (see the description of the **Application Priority** Table below). Each charge type (see below) is associated with one receivable type, but a given receivable type might be associated with more than one charge type.

### Charge Type Table

Records in the Charge Type Table represent types of detailed charges generated during bill calculation. Charge types are defined to roll up to a receivable type when charges are posted as financial transactions.

Though most charge types are financial charges, records in the Charge Type Table can also represent consumption charges; that is, charges that are based on consumption rather than dollars. This is handled through the “Consumption?” and “Unit of Measure (UOM)” columns on the Charge Type Table. If the “Consumption?” value is “Yes”, the UOM column indicates the specific unit of measure for the consumption charge.

### **Credit Application Table**

Each record in the Credit Application Table represents an application of a credit. The rules for how credits are applied to charges are detailed in **Chapter 5: The Financial Engine**.

### **Application Priority Table**

Records in the Application Priority Table indicate the application priority of the associated receivable type. When credits are applied to charges with the same bill date, they will be applied in order of the application priority of the receivable type of any outstanding charges. The highest priority is 1. Receivable types with the same priority will be applied in the order in which they are retrieved from the database (by age, and then by transaction number). The Ignore Age field is used to specify whether age should be factored first in the application of credits. The default for this field is NULL, which means that all charges are ordered by age. Any receivable types without a corresponding application priority record will be applied last. More information concerning how the Application Priority Table interacts with the Financial Engine can be found in **Chapter 5: The Financial Engine**.

---

## Journaling Data

Journaling data defines how transactions are journaled both at the sub-ledger level and the general ledger level.

### Note

Records in the Journal Transaction and Subledger Account tables should be created **only** by the Financial Engine.

### Cost Center Table

Records in the Cost Center Table represent cost centers used by the ledger.

### Cost Center Translation Table

The Cost Center Translation Table associates a market/tax jurisdiction (represented by Operating Company and Jurisdiction) combination with a cost center.

### Journal Account Table

Records in the Journal Account Table represent journal accounts used by the ledger, including:

- Currency (the currency associated with the journal account, from the LS Currency table).

### Journal Translation Table

The Journal Translation Table is used to indicate how transactions and credit applications are journaled for the sub-ledger. A single transaction or credit application may have zero or two journal entries associated with it.

Specific information stored in the Journal Translation Table includes:

- Translation Type (the type of record ("T" for TRANSACTION, 'A' for CREDITAPPLICATION) the translation represents),
- Transaction ID (associated transaction ID),
- Charge or Credit (whether the translation is for a charge ("CH") or credit ("CR") type transaction),
- Receivable Type and Charge Type (associated Receivable and Charge type),
- Operating Company, Jurisdiction, Region, Revenue Code (based on the account associated with the transaction),
- Rate Form and Rate Code (based on the transaction being journaled),
- Payment Method (associated payment method related to the transaction),
- Debit Account ID (associated journal account ID to be charged), and
- Credit Account ID (associated journal account ID to be credited).

## Journal Transaction Table

Records in the Journal Transaction Table represent journal entries for the ledger associated with financial transactions. The Journal Transaction Table is not explicitly related to the financial transaction tables so that it may be implemented in a separate database.

Specific information stored in the Journal Transaction Table includes:

- Account ID,
- Transaction No. (transaction number of related transaction),
- Cancel Flag (indicates whether the related transaction was cancelled or posted (Yes/No)),
- Credit Application No. (the credit application number of the related transaction),
- Transaction Time (time of the associated transaction)
- Subledger Account ID (associated subledger account for the transaction, from the Journal Account Table)
- Cost Center (the cost center to which the transaction amount applies)
- Revenue Month (the month to which any associated revenue is applied)
- Debit or Credit (whether the transaction is a debit or credit)
- Amount (amount of the journal entry)
- Rolled Up Date (the date the transaction was rolled up to the appropriate subledger account).

## Subledger Account Table

The Subledger Account Table maintains financial account balances for all sub-ledger accounts necessary to maintain the CIS database. The balances of the accounts in this table are updated from the Journal Transaction Table at scheduled intervals. The accounts in this table may be rolled up to accounts in the general ledger for the entire company at scheduled intervals.

Specific information stored in the Subledger Account Table includes:

- Subledger Account ID
- Cost Center (cost center to which the balance applies)
- Revenue Month (revenue month to which the balance applies)
- Balance (balance for the account for the indicated revenue month)
- Balance Time (time of the balance for the account)
- Balance to GL (amount of the balance that has been rolled up to the General Ledger)
- Closed Date (the date when the associated revenue month for this account was closed. A Null value indicates that the revenue month is still open).

# Chapter 4

---

## **Setting Up and Configuring the Oracle Utilities Receivables Component Database**

This chapter describes how to set up and configure the Oracle Utilities Data Repository used by the Oracle Utilities Receivables Component, including how to set up the necessary data in the Oracle Utilities Data Repository, and how to configure specific tables in the database to properly process transactions.

## Setting up the Oracle Utilities Receivables Component Database

In order for Oracle Utilities Receivables Component to function properly, the Oracle Utilities Data Repository must contain specific types of data that is used by the Oracle Utilities applications when calculating charges and processing different types of financial transactions. These types of data are stored in four primary types of tables in the Oracle Utilities Data Repository, including:

- **Lookup Tables**
- **Customer/Account Tables**
- **Usage Data Tables**
- **Translation Tables**

Tables corresponding to each of these types are described below. More information about the tables described in this chapter can be found in previous chapters of this manual.

### Lookup Tables

Lookup tables contain data that is used by other tables in the Oracle Utilities Data Repository. This includes data such as operating companies, jurisdictions, addresses, and other information that is referred to by other tables in the database.

#### Populating Lookup Tables

Lookup tables should be the first tables populated, since most of the other data makes use of one or more lookup records. Lookup tables are populated through use of either the PLIMPORT command line program or the Data Manager Import Utility. Refer to the *Data Manager User's Guide* for more information about using these tools.

#### Individual Lookup Tables

Below are brief descriptions of the lookup tables used by Oracle Utilities Receivables Component. In addition to providing a general description of the data stored in each table, any specific issues related to records in the table are listed as well.

The following list does not include all the lookup tables in the Oracle Utilities Data Repository. It only lists those specifically used by Oracle Utilities Receivables Component.

##### Address Table

The Address Table stores physical address information. Records in this table are used in the Premise Table.

##### Budget Type Table

Records in the Budget Type Table represent specific types of budget plans recognized by the application. The application uses budget types to determine how true-ups should be handled.

##### Cancel Reason Table

Records in the Cancel Reason Table represent types of reasons for which a transaction might be cancelled.

##### Charge Type Table

Records in the Charge Type Table represent types of detailed charges generated during bill calculation. Charge types are defined to roll up to a receivable type when charges are posted as financial transactions.

**Note:** The Charge Type Table includes data from both the Units of Measure Table and the Receivable Type Table. Be sure that both of these tables are populated before populating the Charge Type Table.

Though most charge types are financial charges, records in the Charge Type Table can also represent consumption charges; that is, charges that are based on consumption rather than dollars. This is handled through the “Consumption?” and “Unit of Measure (UOM)” columns on the Charge Type Table. If the “Consumption?” value is “Yes”, the UOM column indicates the specific unit of measure for the consumption charge.

### Cost Center Table

Records in the Cost Center Table represent cost centers used by the ledger.

### Journal Account Table

Records in the Journal Account Table represent journal accounts used by the ledger.

### LSCurrency Table

Records in the LSCurrency Table represent the available currency types. The LSCurrency table has three columns:

- **Currency Code:** Indicates the three-letter ISO 4217 code to be used. If the currency symbol is not specified in the **Currency Format** column, the three-letter currency code is used to format monetary values. For example, if CURRENCYCODE is set as USD, the positive values are displayed as “12.34 USD” and negative values are displayed as “(12.34 USD)”.
- **Currency Format:** Used to format currency output values. This column contains an XML string containing the formatting information for currency. The format of the string is as follows:

```
<currency [NumDigits='Integer with max value 9'] [LeadingZero='Integer']
[Grouping='Integer'] [DecimalSep='String'] [ThousandSep='String']
[NegativeOrder='Integer'] [PositiveOrder='Integer']
[CurrencySymbol='String'] />
```

#### Attribute Descriptions:

- **NumDigits:** An integer that specifies the number of decimal digits to display.
- **LeadingZero:** Specifies whether or not to use leading zeros in decimal fields.
- **Grouping:** Specifies the size of each group of digits to the left of the decimal separator. Values in the range of 0 - 9 and 32 are valid. For example, grouping by thousands is indicated by 3.
- **DecimalSep:** The character used as the decimal separator.
- **ThousandSep:** The character used as the thousands separator.
- **NegativeOrder:** Specifies the negative currency mode.
- **PositiveOrder:** Specifies the positive currency mode.
- **Currency Symbol:** String indicating the currency symbol for the currency. For example, the currency symbol for US Dollars is “\$”

All attributes are optional. If an attribute is not set, the value for the attribute is taken from the Windows Regional Settings. If the Currency Symbol is not set, the Currency Code is used for formatting. In this case, Positive/Negative order parameters set will not have any effect.

- **Is Default:** A flag that indicates if the currency is the default for the database. If there are more than one record in the LSCurrency table, one must be set as the default.

### Meter Table

Records in the Meter Table represent scalar meters in the system.

**Premise Table**

The Premise Table stores the physical location of service. This is typically a building, or a location within a building.

**Note:** The Premise Table includes data from the Address Table. Be sure that the Address Table is populated before populating the Premise Table.

**Payment Method Table**

Records in the Payment Method Table represent the different payment methods (Check, Visa, Direct Debit) for payments coming into the system, including automatic payments.

**Note:** The Payment Method Table includes data from the Payment Source Table. Be sure that the Payment Source Table is populated before populating the Payment Method Table.

**Payment Source Table**

Records in the Payment Source Table represent the source of payments coming into the system, including automatic payments. This represents the vendor from whom the payment was sent, such as Chase, Lockbox, etc.

**Receivable Type Table**

Records in the Receivable Type Table represent types of receivables to which a financial transaction amount might apply. Typical receivable types might include “Disco Electric” or “State Tax”. Payments on an account are applied to receivable types based on the application priority of the receivable type (see the description of the **Application Priority** Table below). Each charge type (see below) is associated with one receivable type, but a given receivable type might be associated with more than one charge type.

**Service Type Table**

The Service Type Table is used to define all service-oriented businesses or commodities provided by the client operating company. Typical service types might include “Electric”, “Gas”, or “Cable”.

**Transaction ID Table**

The Transaction ID Table is used to configure different adjustment transaction types, refund transaction types, etc. This table is configurable and can accommodate various financial transaction IDs as needed based on specific requirements. At least one Transaction ID is required for each Transaction Type (see Transaction Type Table, below).

**Note:** The Transaction ID Table includes data from the Transaction Type Table. Be sure that the Transaction Type Table is populated before populating the Transaction ID Table.

**Transaction Type Table**

Records in the Transaction Type Table represent different types of financial transactions recognized by the application, including the following:

<b>Transaction Type</b>	<b>Code</b>
Charge or Credit	CHRGCRDT
Tax	TAX
Installment	INST
Statement	STMT
Bill	BILL
Payment	PAYMENT



Transaction Type	Code
Deposit	DEPOSIT
Deposit Interest	DEPINT
Deposit Application	DEPAPP
Adjustment	ADJ
Refund	RFND
Write-Off	WRTOFF
Balance Transfer	BALXFR

The following transaction types have been effectively replaced by the Charge or Credit (CHRGCRDT) transaction type, but are still available for users of previous versions of Oracle Utilities Receivables Component.

Transaction Type	Code
Service Charge	SRVCHG
Deferred Service Charge	DEFSRVCHG
Budget Service Charge	BDGTSRVCHG
Budget Bill Charge	BDGTCHG
Budget Bill True-up	BDGTTRP
Installment Charge	INSTCHG

**Note:** While additional Transaction Types can be added to the Transaction Type Table, types on the list provided should NOT be deleted.

### Write-Off Reason Table

The Write-Off Reason Table stores reasons for writing off an account, such as “Sold”, “Deceased”, “Bankrupt”, or “Bad Debt”. This is a lookup table used by the Account Table. If the “Write-Off Reason” column in the Account Table is populated, the Account’s Receivable Status should be “UNCOLLECTIBLE”.

## Customer/Account Tables

Customer/Account data is data that defines specific customers and accounts and how transactions related to them are processed by Oracle Utilities Receivables Component and the Financial Engine.

### Populating Customer/Account Tables

Customer/Account tables should be populated after lookup tables, since many of them use one or more lookup records. Customer/Account tables are populated through use of either the PLIMPORT command line program or the Data Manager Import Utility. Refer to the *Data Manager User’s Guide* for more information about using these tools.

### Individual Customer/Account Tables

Below are brief descriptions of the Customer/Account tables used by Oracle Utilities Receivables Component. In addition to providing a general description of the data stored in each table, any specific issues related to records in the table are listed as well.

The following list does not include all the customer/account-related tables in the Oracle Utilities Data Repository. It only lists those specifically used by Oracle Utilities Receivables Component.

### Customer Table

The Customer Table stores information about customers. Customers are the parent records of accounts in the Oracle Utilities Data Repository. Each account is associated to a single customer, though a customer can be associated with more than one account.

### Account Table

The Account Table stores information about individual accounts. An account is the smallest billable entity in the system. Account entities typically are the root of virtually all activity in the system. An account receives a single bill statement from its billing operating company per billing period.

**Setting Up the Default and Master/Summary Accounts:** In addition to individual accounts in the database, there are two specific types of accounts that need to be set up: the Default Account and Master/Summary Accounts.

- **Setting Up the Default Account:** The Default Account is an account that acts as a catch-all account for payment transactions that are unable to be posted to the proper account. When a payment can't be posted to the correct account, Oracle Utilities Receivables Component posts it to the Default Account, so it can later be identified and posted to the proper account. When setting up the Default Account, it's a good idea to use an easily identifiable ID (such as 999999).
- **Setting Up Master/Summary Accounts:** Master/Summary accounts have a relationship to their subordinate accounts that is similar to the relationship between customers and accounts. They are the accounts that are billed for activity and charges of their subordinate accounts. Master/Summary accounts are identified through association to specific Rate Codes in the Rate Code History Table. See **Chapter Three: Billing Rules and Definitions** in the *Oracle Utilities Billing Component User's Guide* for more information.

### Account FME Table

The Account FME Table stores information about individual accounts that is maintained by the Oracle Utilities Receivables Component.

**Important Note:** Values in the Last Balance, Last Balance Time, New Activity, and Last Transaction No. columns should be created only by the Financial Engine, and should not be inserted through other means.

### Auto Payment Plan Table

Records in the Auto Payment Plan Table represent automatic payment plans for a given account for a specified time period. The existence of a current automatic payment plan for an account triggers an automatic draft of the specified automatic payment vendor account when the account is billed. More information concerning automatic payments can be found in **Chapter Five: Remittance**.

### Budget Plan Table

Records in the Budget Plan Table indicate that a given account is on a particular budget plan for a specific time period.

### Service Plan Table

The Service Plan Table associates an account with a service type. An account may have several services active at any one time.

**Note:** The Auto Payment Plan, Budget Plan, and Service Plan tables includes data from the Account Table. Be sure that the Account Table is populated before populating these tables.

## Usage Data Tables

Usage Data tables store usage data for the individual accounts in the system. This includes data such as kWh readings (in the Bill History Table), meter readings, and other data.

### Populating Usage Data Tables

Usage tables are populated on a regular basis, as the usage data is collected by the client. Nearly all usage records rely on both lookup data as well as customer/account data. Like other tables in the database, usage tables are populated through use of either the PLIMPORT command line program or the Data Manager Import Utility. Refer to the *Batch Executables Guide* and the *Data Manager User's Guide* for more information about using these tools.

### Individual Usage Data Tables

Below are brief descriptions of the usage tables used by Oracle Utilities Receivables Component. In addition to providing a general description of the data stored in each table, any specific issues related to records in the table are listed as well.

Note that this does not include all the usage tables in the Oracle Utilities Data Repository. It only lists those specifically used by Oracle Utilities Receivables Component.

#### Bill History Table

The Bill History and Bill History Value tables store account usage data for specified bill periods.

#### Meter Read Table

Records in the Meter Read Table represent readings for a meter.

#### Meter History Table

Records in the Meter History Table represent snapshots of the a given meter's use and related information. The Meter History Table associates meters with accounts and/or service plans. Meter History records should be associated with either an Account or a Service Plan, but not both.

#### Interval Data

Interval data is time-series data used to record usage (among other values) for a given recorder,channel. This data can be used by Oracle Utilities Billing Component to calculate billing determinants, which are in turn used in bill calculations. If you use interval data to calculate billing determinants, you need to supply interval data as well.

## Translation Tables

Translation tables are tables in the Oracle Utilities Data Repository that are used to determine how records in other tables are processed by Oracle Utilities Receivables Component. This includes which cost centers (from the Cost Center Table) and journal accounts (from the Journal Account Table) specific transactions should be applied to, as well as how credits should be applied to charges.

### Populating Translation Tables

Translation tables should be populated when Oracle Utilities Receivables Component is first installed and implemented, and only changed or updated as needed. Translation records rely on several types of lookup data and customer/account data. Like other tables, the translation tables can be populated through use of either the PLIMPORT command line program or the Data Manager Import Utility.

## Individual Translation Tables

Below are brief descriptions of the translation tables used by Oracle Utilities Receivables Component. In addition to providing a general description of the purpose of each table, specific guidelines for setting up and configuring the table are provided as well.

### Application Priority Table

Records in the Application Priority Table indicate the application priority of the associated receivable type. When credits are applied to charges with the same bill date, they will be applied in order of the application priority of the receivable type of any outstanding charges. The highest priority is 1. Receivable types with the same priority will be applied in the order in which they are retrieved from the database (by age, and then by transaction number). The Ignore Age field is used to specify whether age should be factored first in the application of credits. The default for this field is NULL, which means that all charges are ordered by age. Any receivable types without a corresponding application priority record will be applied last. More information concerning how the Application Priority Table interacts with the Financial Engine can be found in **Chapter Three: The Financial Engine**.

Records in the Application Priority Table consist of distinct combinations of the following:

- Receivable Type (Receivable Type of the transaction),
- Operating Company, and
- Jurisdiction.

**Configuring the Application Priority Table:** Assign a Priority to each unique combination of Receivable Type, Operating Company, and Jurisdiction. Set the Ignore Age flag to Yes if the age of the credit should be factored first in the application of credits.

### Cost Center Translation Table

The Cost Center Translation Table associates a market/tax jurisdiction (represented by Operating Company and Jurisdiction) combination with a cost center. Records in the Cost Center Translation Table consist of distinct combinations of Operating Company and Jurisdiction. Each unique combination of Operating Company and Jurisdiction is associated with a unique Cost Center ID (from the Cost Center Table).

**Configuring the Cost Center Translation Table:** For each cost center to be set up in this table, create a record that includes the appropriate Operating Company and Jurisdiction and Cost Center ID.

### Journal Translation Table

The Journal Translation Table is used to indicate how transactions and credit applications are journaled for the sub-ledger. A single transaction or credit application may have zero or two journal entries associated with it.

Records in the Journal Translation Table consist of distinct combinations of the following:

- Translation Type (the type of record the translation represents),
- Transaction ID (associated transaction ID),
- Charge or Credit (whether the translation is for a charge ('CH') or credit ('CR') type transaction),
- Receivable Type and Charge Type (associated Receivable and Charge type),
- Operating Company, Jurisdiction, Region, Revenue Code (based on the account associated with the transaction),
- Rate Code (based on the transaction being journaled), and
- Payment Method (associated payment method related to the transaction).

Each unique combination of the above variables is associated with Credit and Debit Journal Account IDs (both from the Journal Account Table). When a transaction is posted to Oracle Utilities Receivables Component, the Financial Engine compares that transaction to the above combination of variables to determine which Credit and Debit Journal Accounts the transaction should be posted to.

**Configuring the Journal Translation Table:** Assign a Credit and Debit Account ID (from the Journal Account Table) to each unique combination of the above variables. Note that you needn't use all the variables provided in the Journal Translation Table. It's best to use only as many as necessary.

We recommend that you set up two "Default" Journal Application records (one Credit Application record, one Transaction record) in this table that serve as "catch alls" to ensure that all transactions get posted to a Journal Account. These records should consist of a Transaction Type, Credit Journal Account ID, and Debit Journal Account ID only. All other fields should be left NULL. Ideally, the Credit/Debit Journal Accounts (which can be the same journal account) should be specifically set up for this purpose, but can be any account from the Journal Account Table.



# Chapter 5

---

## The Financial Engine

This chapter describes the core functionality of the Oracle Utilities Receivables Component, specifically, the Financial Engine and the specific functions it performs, including:

- **Transactions**
- **Credit Application**
- **Journaling**
- **Sub-Ledger Rollup**
- **General Ledger Update**
- **Balancing Controls**

Understanding how the Financial Engine performs these functions is a critical step in learning how to use and operate the Oracle Utilities Receivables Component. The descriptions in this chapter explain how the above functions are performed by the Financial Engine. Specific applications of some of these processes, such as processing of Billing or Remittance transactions, are described in more detail in later chapters of this manual.

## Financial Engine Functions - Summary

The Financial Engine performs six functions that are central to the overall functionality supported by the Oracle Utilities Receivables Component. Each of these functions is briefly described below.

### Transactions

Processing of transactions is the primary function of the Financial Engine. The transactions function posts charges, credits, and other transactions to the Financial Management database tables in the Oracle Utilities Data Repository (see **Chapter 3: The Oracle Utilities Receivables Component Database** for more information). Processing transactions includes posting, transferring, and cancelling transactions. In addition, each of the Oracle Utilities Receivables Component Modules (p. 1-3) applies specific rules and conditions to transaction processing. See **Transactions** on page 5-4 for a more detailed description.

### Credit Application

The credit application function applies credit transactions to outstanding charges. How credits are applied to an account's charges is defined in the Application Priority Table in the Oracle Utilities Data Repository (described in **Chapter 3: The Oracle Utilities Receivables Component Database**). See **Credit Application** on page 5-13 for a more detailed description.

### Journaling

The journaling function creates journal entries that correspond to posted and cancelled transactions and credit application records. Every time a transaction is posted or cancelled, or credit application records are created, corresponding journal entries are created in the Journal Transaction Table based on rules defined in the Journal Translation and Cost Center Translation tables (see **Chapter 3: The Oracle Utilities Receivables Component Database** for more information on these tables). See **Journaling** on page 5-18 for a more detailed description.

### Sub-Ledger Rollup

The sub-ledger roll-up function “rolls up” records in the Journal Transaction Table to their corresponding sub-ledger account records. This process can be performed using a command line program or by an external system through an interface. See **Sub-Ledger Rollup** on page 5-20 for a more detailed description.

### General Ledger Update

The general ledger update function outputs records from the Sub-ledger Account table for updating an external general ledger system. This process can be performed using a command line program or by an external system through an interface. See **General Ledger Update** on page 5-22 for a more detailed description.



## Balancing Controls

The balancing controls function operates on three levels: the transaction level, the journal level, and the account level. Transaction-level balancing is triggered automatically every time a transaction is posted or cancelled to verify that the latest activity for the corresponding account is still in balance. Journal-level balancing is used at scheduled intervals to balance records in the Journal Transaction Table against corresponding records in the Transaction and Credit Application tables. Account-level balancing is used at scheduled intervals to balance the records in the Transaction Table for a user-specified subset of accounts in the database. Account- and journal-level balancing can be performed using a command line program or by an external system through an interface. See **Balancing Controls** on page 5-24 for a more detailed description.

## Transactions

Transactions post charges, credits, and other transactions to the Financial Management database tables in the Oracle Utilities Data Repository (see **Chapter 3: The Oracle Utilities Receivables Component Database** for more information), and includes posting, transferring, and cancelling transactions.

Each of the Financial Management modules applies specific rules, processes, and conditions to the transactions functions described in this section. These are described in later chapters of this book.

## Transaction Data

Each transaction processed by the Financial Engine is represented by a record in the Transaction Table (p. 3-6) in the Oracle Utilities Data Repository that contains the data required by the Financial Engine to process the transaction correctly. This processing can include triggering a credit application if appropriate (p. 5-13), creating corresponding journal entries (p. 5-18), or other processes.

### Transaction

The data that comprises a transaction in the Financial Engine includes the following:

- A unique identifier for the transaction,
- Transaction Number (a unique number for the transaction within the account),
- Application Method (indicates the appropriate credit application method),
- Defer Balance (indicates whether or not to defer the balance associated with the transaction),
- Account (from the Account Table),
- Transaction Type (from the Transaction Type Table),
- Transaction ID (from the Transaction ID Table),
- Transaction Time (the time of the transaction),
- Revenue Month (the month to which any revenue associated with the transaction applies),
- User ID (user identification associated with the transaction),
- Note (a note associated with posting of the transaction),
- Debit Account (an optional debit journal account ID associated with posting transaction, from the Journal Account Table),
- Credit Account (an optional credit journal account ID associated with posting transaction, from the Journal Account Table),
- Cost Center (an optional cost center ID associated with posting transaction (from the Cost Center Table),
- Cancel Time, Cancel Revenue Month, Cancel User ID, Cancel Reason, Cancel Note, Cancel Debit Account, Cancel Credit Account, Cancel Cost Center (time, revenue month, user ID, reason, note, debit account, credit account, and cost center for the cancellation of the transaction),
- Charge or Credit (whether the transaction is a charge or credit),
- Amount (the amount of the transaction, including the currency code for the transaction. In the case of a consumption charge, the amount is the amount of consumption),
- Balance (outstanding balance of charge or credit, including the currency code for the balance. This is NULL if the amount does not affect the Balance),
- Billed Or Paid Date (date the charge was billed or credit paid),

- Due Date (date that payment for a charge is due),
- Receivable Type Data (from the Receivable Type Table),
- Charge Type Data (from the Charge Type Table),
- Rate Code Data (associated Rate Code and Rate Form from the Rate Code Table),
- Operating Company (the associated operating company from the Operating Company Table),
- Jurisdiction (the associated jurisdiction from the Jurisdiction Table),
- Statement Date (optional date of statement with which the transaction is associated),
- Invoice ID (associated invoice ID for the transaction),
- Invoice Date (associated invoice date for the transaction),
- Cancel Statement Date (optional date of statement with which the cancellation of the transaction is associated),
- Cancel Invoice ID (associated invoice ID for the cancelled transaction),
- Cancel Invoice Date (associated invoice date for the cancelled transaction),
- Bill Cycle Date (associated bill cycle from the Bill Cycle Table),
- Bill History (associated record from the Bill History Table),
- Sub Account (optional unique ID of originating sub-account for summary billing, from the Account Table),
- Transferred To Transaction/Transferred From Transaction (for transactions that have been transferred from one account to another),

**Note:** Payment data is populated only for Payment (PYMNT) transactions.

- Payment, Service Plan, Budget Plan (associated Payment, Service/Budget Plan records),
- Installment Plan (installment plan associated with deferred charge or installment transactions),
- Deposit (unique ID of deposit associated with Deposit, Deposit Interest, Deposit Application, or Installment transactions)
- Deposit Interest Rate (interest rate on deposits for Deposit transactions),
- Tax Amount (tax amount for related taxed transactions, including the currency code for the tax amount),
- Tax Rate (tax rate for either Tax transactions or related taxed transaction),
- Tax Exempt (indicates tax exempt status for related taxed transactions),
- Bill Start and Stop Times (the start and stop times for BILL transactions),
- Suspend Auto Payment (indicates that automatic payments for the bill transaction should be suspended)
- Batch Transactions (batch transaction from where the transaction came),
- Batch Cancel (batch transaction that cancelled the transaction),
- Related Transactions (any related transactions for this transaction. For transfer transactions, this is the list of transferred transactions. For adjustments or refunds, this is an optional list of transactions that are being adjusted or refunded), and
- User-Defined Attributes (a collection of user-defined attributes for the transaction). These include the following:

- Name (the name for the attribute. This should be the actual column name in the database),
- Type (the type of the attribute, i.e. CHAR, VARCHAR, DECIMAL, INTEGER, etc.),
- Size (optional size of the attribute. Required for CHAR and VARCHAR types),
- Precision (optional precision of the attribute. Required for DECIMAL types),
- Scale (optional scale of the attribute. Required for DECIMAL types),
- Value (the value of the attribute).

**Note:** Including user-defined attributes in transaction requires corresponding columns in the Transaction Table (p. 3-6). It is an error to include user-defined attributes in a transaction without corresponding columns in the Transaction Table.

## Batch Transactions

A batch transaction contains a number of transactions. The data that comprises a batch transaction includes the following:

- A unique identifier for the batch transaction,
- Batch No (a unique number of the batch transaction within the payment file),
- Cancel (indicates that the batch should be processed in Cancel mode),
- Restart (indicates that the batch should be processed in Restart mode),
- Transaction ID (the optional Transaction ID for the transaction in the batch). Forwarded to individual transactions if not already provided,
- Revenue Month (the optional revenue month for the transactions in the batch). Forwarded to individual transactions if not already provided,
- Transaction File (the transaction file from where the batch came). Optional for processing. This includes the unique path and file name of the transaction file,
- Date (the date of the batch transaction). Optional for processing. Forwarded to individual transactions if not already provided,
- Number of Transactions (the total number of transactions in the batch). Optional for processing,
- Amount (the total amount of all the transactions in the batch, including the currency code for the amount). Optional for processing,

**Note:** All transactions within a batch must use the same currency.

- Cancel Revenue Month (the optional revenue month for a cancelled batch. If not provided, the current month will be used),
- Cancel Reason Code (an optional reason code for canceling batch),
- Cancel Note (an optional note associated with cancelled batch), and
- Max Errors (the maximum number of transaction errors allowed prior to stopping the process). Optional for processing, and
- Transactions (the individual transactions in the batch). Required for processing.

---

## Transaction Files

A transaction file contains a number of batch transactions. The data that comprises a transaction file includes the following:

- Name (the unique file path name of the transaction file) Required for processing,
- Cancel (indicates that the file should be processed in Cancel mode),
- Restart (indicates that the file should be processed in Restart mode),
- Transaction ID (an optional Transaction ID for the batch transactions in the transaction file) Forwarded to individual batch transactions if not already provided,
- Revenue Month (the optional revenue month for the batch transactions in the transaction file) Forwarded to individual batch transactions if not already provided,
- Date (the optional date of the transaction file) Forwarded to individual batch transactions if not already provided,
- Number of Batches (the total number of batches in the transaction file) Optional for processing,
- Number of Transactions (the total number of transactions in the payment file) Optional for processing,
- Amount (the total amount of all transactions (or batch transactions) in the transaction file, including the currency code for the amount) Optional for processing,

**Note:** All batches within a transaction file must use the same currency.

- Cancel Revenue Month (the optional revenue month for a cancelled file. If not provided, the current month will be used),
- Cancel Reason Code (an optional reason code for canceling file),
- Cancel Note (an optional note associated with cancelled file), and
- Max Errors (the maximum number of batch transaction errors allowed prior to stopping the process) Optional for processing,
- Max Errors Per Batch (the maximum number of transaction errors allowed per batch transaction prior to stopping the batch transaction process) Forwarded to individual batch transactions if not already provided. Optional for processing, and
- Batch Transaction (individual batch transactions in the transaction file) Required for processing.

## Transaction Types

The following transaction types are currently supported by the Oracle Utilities Receivables Component. All of these are initiated through one of the Financial Management modules (Billing, Remittance, or Maintenance) or by an external system (such as an external payment or billing system) through an interface.

<b>Transaction Type</b>	<b>Code</b>
Charge or Credit	CHRGCRDT
Tax	TAX
Installment	INST
Statement	STMT
Bill	BILL
Payment	PAYMENT
Deposit	DEPOSIT
Deposit Interest	DEPINT
Deposit Application	DEPAPP
Adjustment	ADJ
Refund	RFND
Write-Off	WRTOFF
Balance Transfer	BALXFR

### Deprecated Transaction Types

The following transaction types have been effectively replaced by the Charge or Credit (CHRGCRDT) transaction type, but are still available for users of previous versions of the Oracle Utilities Receivables Component.

<b>Transaction Type</b>	<b>Code</b>
Service Charge	SRVCHG
Deferred Service Charge	DEFSRVCHG
Budget Service Charge	BDGTSRVCHG
Budget Bill Charge	BDGTCHG
Budget Bill True-up	BDGTTRP
Installment Charge	INSTCHG

---

## Transaction Processing

The transactions functions of the Financial Engine includes three primary types of processing: posting transactions, transferring transactions, and cancelling transactions. Each is described below.

### Post Transaction

The Post Transaction function sends transactions from the various Oracle Utilities Receivables Component modules (p. 2-3) and records them in the Financial Management tables of the Oracle Utilities Data Repository. The Financial Engine uses this transaction data to process the transaction. The diagram below provides a high-level illustration of the Post Transaction process.

When a transaction is posted by the Financial Engine, the following occurs:

1. A transaction record is created and inserted into the Transaction Table.

During this step (between the creation and posting of the transaction record), a transaction-level balancing control is triggered to verify that the account is in balance before posting the transaction to the Transaction Table, and the Activity field on the Account record is updated to reflect the current transaction. See **Transaction Balancing** on page 5-24 for more information.

2. If the transaction is a credit and the Account has outstanding charges, or is a Bill Charge transaction and there is an unapplied credit (see **Chapter 6: Billing**), a credit application record is created and inserted into the Credit Application Table. The specific data contained in this record is based on the Charge Type and Receivable Type of the transaction and the configuration of the Application Priority Table. This step is described in this chapter in **Credit Application** (p. 5-13).

**Note:** This step is skipped if the Application Method is DEFERRED.

3. Two journal records are created and inserted into the Journal Transaction Table. The specifics of these journal records are based on the type of transaction and the corresponding Cost Center, and the configuration of the Journal Translation Table. This step is described in more detail later in **Journaling** (p. 5-18).

## Post Transaction Processing in Detail

The steps above describe the overall Transaction Posting process of the Financial Engine. The steps outlined below describe the transaction posting process in more detail. Most of these steps apply to all types of transactions. If a step applies to a specific transaction type, it is noted in brackets ([ ]).

1. Get data concerning the parent Account. (This is always provided.)
2. If account's Receivable Status is “Uncollectible”, then post an “UNPOSTABLE\_TRANSACTION” message that includes the appropriate message structure and an appropriate error message, and stop processing.
3. Calculate the transaction number. Increment the value in the Last Transaction Number field on the Account record and update the record.
4. Determine appropriate data for the transaction record (including any user-defined attributes) as outlined under **Transaction Data**, above. The Amount of the transaction should be equal to or greater than zero. The CURRENCY attribute of the Amount is checked against the Currency value of the account against which the transaction is being posted using the following rules: 1) If a currency type is specified for the account in the Account table then the transaction must indicate the same currency type or it is an error. 2) If a currency type is not specified for the account in the Account table, then the transaction must indicate either no specific currency type or the default currency type or it is an error. The Balance of the transaction should equal the amount if not deferred, or NULL if deferred.
5. Create a transaction record with attributes from Step 4 above.
6. Perform transaction-level balancing controls (see p. 5-24), and update the Activity field on the Account record. If error, then post message and fail. The message should include the following.
  - The message type should be “POST\_TXACTION\_ERROR”.
  - The error message.
7. Apply credits as specified by the Application Method (see **Credit Application** on page 5-13). If the transaction is a charge transaction and an associated deposit is affected, then update the deposit's principal balance accordingly.
8. Perform transaction journaling (see **Journaling** on page 5-18).
9. Insert the transaction record (including any user-defined attributes) into the Transaction Table.
10. If an associated budget plan is provided, then update the budget plan's variance accordingly.
11. If an installment plan is provided, then invoke the Create Installment Plan function of the Billing module. [Deferred Charges]
12. Post a control activity message. The message should include the following.
  - The message type should be “<transaction type>\_TXACTION\_POSTED”.
  - The transaction using the <TRANSACTION> XML element format.
  - The credit applications, if any.
  - The journal transactions, if any.

## Triggering the Post Transaction Function

The Post Transaction function is triggered by one of the Oracle Utilities Receivables Component modules, such as Billing or Remittance. These modules are in turn triggered either by the user interface, or by an external system (such as an external payment or billing system) through an interface.



## Transfer Transaction Amount

The Transfer Transaction Amount function is used to transfer a previously posted transaction from one account to another. The transfer process is based on:

- the transaction to be transferred (referred to as the “from” transaction),
- the “from” account (which represents the account the transaction is being transferred from), and
- the “to” account (which represents the account the transaction is being transferred to).

The steps outlined below apply to all transaction types.

1. The Financial Engine makes a copy of the transferred transaction and posts it to the “to” account with the Revenue Month as provided or the current month, and “Transferred From Transaction” unique ID (UIDTXFRFROMTRANS) equal to the unique ID of the transferred transaction.
2. The Financial Engine cancels the transferred transaction with a cancel reason code of “TXFR” and update its “Transferred To Transaction” (UIDTXFRTOTRANS) value to the unique ID of transaction posted in Step 1 above.

## Triggering the Transfer Transaction Amount Function

The Transfer Transaction Amount function is most often invoked through the Oracle Utilities Receivables Component user interface, but can also be triggered through one of the modules by an external system using an interface.

## Cancel Transaction

The Cancel Transaction function is used to cancel a previously posted or transferred transaction. The steps outlined below describe the transaction canceling process. Most of the steps apply to all types of transactions. If a step applies to a specific transaction type, it is noted in brackets ([ ]).

1. Determine the transaction to be cancelled (this is always provided).
2. Calculate the cancel time (current time).
3. Determine the cancel revenue month (either provided or current month).
4. Determine the cancel user ID (provided via DataSource object).
5. Determine the cancel reason (this is either provided or NULL).
6. Determine the cancel note (this is either provided or NULL).
7. Perform transaction-level controls (see below). If error then post message and fail. The message should include the following.  
The message type should be “CANCEL\_TXACTION\_ERROR”.  
The error message.
8. Unapply any previously applied credits (see **Credit Application** on page 5-13). If the transaction is a charge transaction and an associated deposit is affected, then update the deposit’s principal balance accordingly.
9. Perform transaction journaling (see **Balancing Controls** on page 5-24).
10. Update the transaction record.
11. If an associated installment plan exists then invoke the Cancel Installment Plan function of the Billing module. [Deferred Charges]
12. If any taxes exist for the cancelled transaction then post the appropriate tax adjustments and update the associated tax records.
13. If an associated budget plan is provided, then update the budget plan’s variance accordingly.

14. Post a control activity message. The message should include the following.

The message type should be "TXACTION\_CANCELLED".

The transaction using the <ACCOUNT> and <TRANSACTION> XML element formats.

The credit applications, if any.

The journal transactions, if any.

### **Triggering the Cancel Transaction Function**

The Cancel Transaction function is most often invoked through the Oracle Utilities Receivables Component user interface, but can also be triggered through one of the modules or by an external system using an interface.

### **Cancel Transfer Amount**

The Cancel Transfer Amount function cancels a previous transfer amount transaction. The steps outlined below apply to all transaction types.

1. The Financial Engine verifies that the transaction has been transferred. If not, it is an error.
2. The Financial Engine cancels the transferred "to" transaction, and sets the cancel Revenue Month, Cancel Reason Code, and Cancel Note attributes, if provided.
3. The Financial Engine makes a copy of and re-posts the cancelled transferred "from" transaction.

### **Triggering the Cancel Transfer Amount Function**

The Cancel Transfer Amount function is most often invoked through the Oracle Utilities Receivables Component user interface, but can also be triggered through one of the modules or by an external system using an interface.

## Credit Application

The credit application function applies credit transactions to accounts that have outstanding charges associated with them. The specific priority by which credits are applied to an account's charges are defined in the Application Priority Table in the Oracle Utilities Data Repository (described in **Chapter 3: The Oracle Utilities Receivables Component Database**).

The Credit Application function is triggered by any one of a specific set of transactions. These include:

- Posting a charge transaction,
- Posting a credit transaction,
- Cancelling a charge transaction, and
- Cancelling a credit transaction.

**Note:** The triggering of the Credit Application function can be overridden if the Application Method for the Transaction is DEFERRED.

Every time a charge or credit transaction is posted and every time a transaction is cancelled, the Financial Engine creates appropriate credit application records. The Financial Engine performs specific steps based on the specific action that triggers the Credit Application function, and the specific credit application method. Oracle Utilities Receivables Component supports five credit application methods:

- Deferred: Do not apply credits at this time
- Immediate: Apply any unapplied credits to any outstanding charges
- Specified: Apply current charge or credit transaction to specified transactions.
- Invoice ID: Apply current charge or credit transaction to outstanding charges or unapplied credits with same invoice ID.
- Receivable Type: Apply current charge or credit transaction to outstanding charges or unapplied credits with specific receivable types.

Each of the transaction types and credit application methods is described below.

### Credit Application Processing for Charge or Credit Transaction - Immediate

1. The Financial Engine finds any unapplied credit transactions (those in which the Balance is greater than zero) and orders them by "Billed or Paid Date" and then by Transaction Number (in ascending order). If there are no unapplied credits and the triggering transaction is a charge or is deferred, then processing is done. If the triggering transaction is a credit transaction or there are unapplied credit transactions, go on to Step 2.
2. The Financial Engine finds any outstanding charge transactions (those in which the Balance is greater than zero) and orders by application priority (based on the configuration of the Application Priority table), and then by Transaction Number (in ascending order). If there are no outstanding charges and the triggering transaction is a credit or is deferred then processing is done. See **Application Priority Table** on page 4-8 for more information about the Application Priority table.
3. For each unapplied credit transaction and outstanding charge transaction, the Financial Engine updates the Balance by subtracting either the current credit transaction balance or the current charge transaction balance, whichever is the lesser amount, and inserts a record in the Credit Application Table for the amount subtracted with a Credit Transaction unique ID (UIDCREDITTRANS) value equal to the unique ID of the current credit transaction (UIDTRANSACTION) and a Charge Transaction unique ID (UIDCHARGETRANS) value equal to the unique ID of the current charge transaction (UIDTRANSACTION). The Cancel field should be "N" and the Trigger Translation unique ID (UIDTRIGGERTRANS) value

should be equal to the unique ID (UIDTRANSACTION) of the triggering credit or charge transaction. When the balance reaches zero for either the current credit transaction or current charge transaction, the Financial Engine then moves on to the next one. This process continues until there are no more unapplied credit transactions or no more outstanding charge transactions. The Credit Application No. for the first credit application record for the triggering transaction should be 1 and incremented for each successive credit application record for the triggering transaction. If an unapplied credit is encountered that was posted using the Receivable Type Application Method, the Financial Engine determines the Receivable Type(s) that the credit may be applied to. The Financial Engine then checks each outstanding charge transaction and applies as much of the credit to compatible charges as possible, then advances to the next credit.

4. If the triggering transaction is a non-deferred credit and there are still remaining outstanding charges, the Financial Engine applies the triggering credit transaction to the outstanding charges in the same manner as in Step 3 above.
5. If the triggering transaction is a non-deferred charge and there are still remaining unapplied credits, the Financial Engine applies the triggering charge transaction to the unapplied credits in the same manner as in Step 3 above.
6. The Financial Engine creates journal records for each record inserted into the Credit Application Table. The specifics of these journal records are based on the type of transaction and the corresponding Cost Center, and the configuration of the Journal Translation Table. This step is described in more detail later in this chapter in the section entitled **Journaling** on page 5-18.

## Credit Application Processing for Charge or Credit Transaction - Specified

1. If previous charge transactions are specified along with current credit transaction, the Financial Engine orders the charges by application priority, then by Transaction Number (in ascending order).
2. For each unapplied credit transaction and outstanding charge transaction as specified, the Financial Engine updates the Balance values and inserts a record in the Credit Application Table record, similar to the IMMEDIATE application method described above. The Directed flag on each new Credit Application record should be set to “Yes”. For an unapplied credit that was posted using the Receivable Type Application Method, the Financial Engine determines the Receivable Type(s) to which that credit may be applied. The Financial Engine then checks each outstanding charge transaction and applies as much of the credit as possible to compatible charges, then advances to the next credit.
3. The Financial Engine creates journal records for each record inserted into the Credit Application Table. The specifics of these journal records are based on the type of transaction and the corresponding Cost Center, and the configuration of the Journal Translation Table. This step is described in more detail later in **Journaling** on page 5-18).

## Credit Application Processing for Charge or Credit Transaction - Invoice ID

1. The Financial Engine finds all outstanding charge (if triggering transaction is a credit) or credit (if triggering transaction is a charge) transactions (those in which the Balance is greater than zero) with an Invoice ID value matching that of the triggering transaction (it is an error if the credit transaction does not have an invoice id). It orders outstanding charges by application priority, then Transaction Number (in ascending order).
2. If the sum of the balances of the outstanding transactions is less than the balance of the triggering transaction, it is an error.
3. For each outstanding transaction found, along with the triggering transaction, the Financial Engine updates the Balance values and inserts a record in the Credit Application Table record in a manner similar to the IMMEDIATE application method described above. The Directed flag on each new Credit Application record should be set to “Yes”. If an unapplied credit is encountered that was posted using the Receivable Type Application Method, the Financial Engine determines the Receivable Type(s) to which the credit may be applied. The Financial Engine then checks each outstanding charge transaction and applies as much of the credit as possible to compatible charges, then advances to the next credit.
4. The Financial Engine creates journal records for each record inserted into the Credit Application Table. The specifics of these journal records are based on the type of transaction and the corresponding Cost Center, and the configuration of the Journal Translation Table. This step is described in more detail later in this chapter in the section entitled **Journaling** on page 5-18).

## Credit Application Processing for Charge or Credit Transaction - Receivable Type

1. The Financial Engine finds any unapplied credit transactions (those in which the Balance is greater than zero) and orders them by “Billed or Paid Date” and then by Transaction Number (in ascending order). If there are no unapplied credits and the triggering transaction is a charge or is deferred, then processing is done. If the triggering transaction is a credit transaction or there are unapplied credit transactions, go on to Step 2.
2. The Financial Engine finds any outstanding charge transactions (those in which the Balance is greater than zero) and orders by application priority (based on the configuration of the Application Priority Table), and then by Transaction Number (in ascending order). If there are no outstanding charges and the triggering transaction is a credit or is deferred then processing is done.

3. For each unapplied credit transaction and outstanding charge transaction, the Financial Engine updates the Balance by subtracting either the current credit transaction balance or the current charge transaction balance, whichever is the lesser amount, and inserts a record in the Credit Application Table for the amount subtracted with a Credit Transaction unique ID (UIDCREDITTRANS) value equal to the unique ID of the current credit transaction (UIDTRANSACTION) and a Charge Transaction unique ID (UIDCHARGETRANS) value equal to the unique ID of the current charge transaction (UIDTRANSACTION). The Cancel field should be "N" and the Trigger Translation unique ID (UIDTRIGGERTRANS) value should be equal to the unique ID (UIDTRANSACTION) of the triggering credit or charge transaction. When the balance reaches zero for either the current credit transaction or current charge transaction, the Financial Engine then moves on to the next one. This process continues until there are no more unapplied credit transactions or no more outstanding charge transactions. The Credit Application No. for the first credit application record for the triggering transaction should be 1 and incremented for each successive credit application record for the triggering transaction. If an unapplied credit is encountered that was posted using the Receivable Type Application Method, the Financial Engine determines the Receivable Type(s) that the credit may be applied to. The Financial Engine then checks each outstanding charge transaction and applies as much of the credit to compatible charges as possible, then advances to the next credit.
4. If the triggering transaction is a non-deferred credit and there are remaining outstanding charges, the Financial Engine applies the triggering credit transaction to the outstanding charges in the same manner as in Step 3.
5. If the triggering transaction is a non-deferred charge and there are remaining unapplied credits, the Financial Engine applies the triggering charge transaction to the unapplied credits in the same manner as in Step 3.
6. The Financial Engine creates journal records for each record inserted into the Credit Application Table. The specifics of these journal records are based on the type of transaction and the corresponding Cost Center, and the configuration of the Journal Translation Table. This step is described in more detail later in **Journaling** on page 5-18).

## Credit Application Processing for Cancel Charge Transaction

A cancel charge transaction cancels a previously posted charge. Credits previously applied to the cancelled charge transaction will be unapplied. However, the actual credit applied to the charge transaction may not be the credit transaction that is unapplied. Instead, the amount that was applied to the charge transaction will be used to increase the balance of the most recently applied credit transactions. Whenever a charge transaction is cancelled (see **Cancel Transaction** on page 5-11), the Credit Application function of the Financial Engine performs the following steps:

1. The Financial Engine determines the difference between the cancelled charge transaction's Amount and Balance. If the difference is zero, then processing is complete.
2. Unapply any directed credit applications for the cancelled charge.
3. The Financial Engine then finds all applied credit transactions for the account and orders by "Billed or Paid Date", and then by Transaction Number (in descending order).
4. For each credit transaction found in Step 3 above:
  - If this is a credit that was directed by Receivable Type, the Financial Engine checks to make sure the triggering transaction's Receivable Type is one of the acceptable Receivable Types for the credit. If not, skip to next credit.
  - The Financial Engine updates both the Balance of the current credit transaction and the cancelled charge transaction by adding either the difference between the charge transaction Amount and Balance or the credit transaction Amount and Balance, whichever is the lesser amount, until the cancelled charge transaction Balance reaches its Amount.

5. The Financial Engine inserts a record in the Credit Application Table for each affected credit transaction in Step 3 above. The unique ID for this record (UIDCREDITTRANS) should be the unique ID (UIDTRANSACTION) of the credit transaction. The unique ID of the charge transaction (UIDCHARGETRANS) should be the unique ID (UIDTRANSACTION) of the cancelled charge transaction. The “Cancel” field should be ‘Y’. The unique ID of the triggering transaction (UIDTRIGGERTRANS) should be the unique ID value (UIDTRANSACTION) of the cancelled charge transaction. The Credit Application No. for the first credit application record for the triggering transaction should be 1, and is incremented for each successive credit application record for the triggering transaction.
6. The Financial Engine creates journal records for each record inserted into the Credit Application Table. The specifics of these journal records are based on the type of transaction and the corresponding Cost Center, and the configuration of the Journal Translation Table. This step is described in more detail later in **Journaling** on page 5-18).

## Credit Application Processing for Cancel Credit Transaction

A cancel credit transaction cancels a previously posted credit. Whenever a credit transaction is cancelled (see **Cancel Transaction** on page 5-11), the Credit Application functionality of the Financial Engine performs the following steps:

1. If the Balance of the cancelled credit transaction equals its Amount, then processing is complete. Otherwise, go on to Step 2.
2. Unapply any directed credit applications for the cancelled charge.
3. The Financial Engine finds all paid charge transactions for the account and orders them according to application priority (based on the configuration of the Application Priority Table), and then by the Transaction Number (in descending order).
4. For each paid charge transaction found in Step 3 above:
  - If the credit was directed by Receivable Type, the Financial Engine checks to make sure the triggering transaction’s Receivable Type is one of the acceptable Receivable Types for the credit. If not, the Financial Engine skips to the next credit.
  - The Financial Engine updates both the Balance of the charge transaction and the cancelled credit transaction by adding the lesser of the difference between the cancelled credit transaction Amount and Balance or current charge transaction Amount and Balance, until the cancelled credit transaction Balance reaches its Amount.
5. For each of the affected charge transactions from Step 3 above, a record is inserted into Credit Application Table with the Amount added to the charge's Balance. The unique IDs of the credit transaction (UIDCREDITTRANS) and the triggering transaction (UIDTRIGGERTRANS) should be the same as the unique ID (UIDTRANSACTION) of the cancelled credit transaction. The “Cancel” field should be ‘Y’. The unique ID of the charge transaction (UIDCHARGETRANS) should be the same as the unique ID (UIDTRANSACTION) of the affected charge transaction. The Credit Application No. for the first credit application record for the triggering transaction should be 1, and is incremented for each successive credit application record for the triggering transaction.
6. The Financial Engine creates journal records for each record inserted into the Credit Application Table. The specifics of these journal records are based on the type of transaction and the corresponding Cost Center, and the configuration of the Journal Translation Table. This step is described in more detail later in **Journaling** on page 5-18).

## Journaling

Journaling creates journal entries that correspond to posted and cancelled transactions and credit application records. Every time a transaction is posted or cancelled, or credit application records are created, corresponding journal entries are created in the Journal Transaction Table based on rules defined in the Journal Translation and Cost Center Translation tables (see **Chapter 3: The Oracle Utilities Receivables Component Database** for more information on these tables).

### Journaling Processing

Whenever a transaction is posted or cancelled by the Financial Engine, corresponding journal entries are inserted into the Journal Transaction Table. The journal entries associated with each transaction are defined in the Journal Translation and Cost Center Translation tables. Each record in the Journal Translation Table defines two journal entries (a debit and a credit) that correspond to a Transaction or Credit Application record. The steps below outline the journaling process.

1. The Financial Engine locates the corresponding Journal Translation record for the Transaction or Credit Application record (see below).
2. The Financial Engine next locates the corresponding Cost Center from the Cost Center Translation record with matching Operating Company and Jurisdiction (this may be NULL if there is no match). For Credit Application records, the Operating Company and Jurisdiction values will come from the associated charge transaction record via the unique ID for the charge transaction (UIDCHARGETRANS).
3. For the Journal Translation record located in Step 1, the Financial Engine inserts two records into the Journal Transaction Table: one credit based on the Credit Account ID and one debit based on the Debit Account ID. The Account ID, Transaction No., Transaction Time, and Revenue Month values should come from the associated triggering transaction record (either a Transaction or Credit Application record). The Credit Application No. should come from the Credit Application record. The Cost Center ID value should be what was located in Step 2. The Amount value should be equal to the amount of the record being journalled. The Sub-Ledger Account ID and the Debit or Credit values are indicated by the Journal Translation record. The Rolled Up Date value should initially be NULL.
4. If the Transaction or Credit Application record is being cancelled, then the Credit Account ID and Debit Account ID should be switched, effectively journaling the exact opposite entries compared to when the record was initially posted.



## Journal Translation Rules

The Financial Engine uses the following rules to locate the appropriate Journal Translation record(s) for each record type. Only the Journal Translation record with the most specific match (weighted descending from top to bottom) should be used for journaling.

### Journaling Transaction Records

1. The Translation Type should be 'T' (Transaction).
2. The Transaction ID should match the transaction's Transaction ID or be NULL.
3. The Charge or Credit field should match the transaction's Charge or Credit field or be NULL.
4. The Currency field for the Credit Account and Debit Account must match the Currency field for the transaction's Account.
5. the journal translation records have credit/debit accounts with the same currency type as the account against which the transaction is being posted. Otherwise it is an error.
6. The Receivable Type should match the transaction's Receivable Type or be NULL.
7. The Charge Type should match the transaction's Charge Type or be NULL.
8. The Operating Company should match the parent Account's Operating Company or be NULL.
9. The Jurisdiction should match the parent Account's Jurisdiction or be NULL.
10. The Region should match the parent Account's Region or be NULL.
11. The Revenue Code should match the parent Account's Revenue Code or be NULL.
12. The Rate Form should match the Transaction Rate Form or be NULL.
13. The Rate Code should match the Transaction Rate Code or be NULL.
14. For Payment transaction types, the Payment Method should match the associated Payment's Payment Method or be NULL.

### Journaling Credit Application Records

1. The Translation Type should be 'A' (Credit Application).
2. The Transaction ID should match the associated credit transaction's Transaction ID or be NULL.
3. The Charge or Credit field is ignored.
4. The Receivable Type should match the associated charge transaction's Receivable Type or be NULL.
5. The Charge Type should match the associated charge transaction's Charge Type or be NULL.
6. The Operating Company should match the parent Account's Operating Company or be NULL.
7. The Jurisdiction should match the parent Account's Jurisdiction or be NULL.
8. The Region should match the parent Account's Region or be NULL.
9. The Revenue Code should match the parent Account's Revenue Code or be NULL.
10. The Rate Form should match the parent charge transaction's Rate Form or be NULL.
11. The Rate Code should match the parent charge transaction's Rate Code or be NULL.
12. For Payment transaction types, the Payment Method should match the associated payment's Payment Method of the parent credit transaction or be NULL.

## Sub-Ledger Rollup

The Sub-Ledger Roll-up function “rolls up” records in the Journal Transaction Table to their corresponding sub-ledger account records. This process can be performed using a command line program (see **Using the Sub-Ledger Roll-Up Command Line Program** on page 5-21) or by an external system through an interface.

### Sub-Ledger Roll-Up Processing

At scheduled intervals, the records in the Journal Transaction Table are “rolled up” to their corresponding account records in the Sub-Ledger Account Table. The steps outlined below describe the sub-ledger roll-up process.

1. The Financial Engine locates all the records in the Journal Transaction Table with a Rolled Up Date value of NULL, and groups them by Sub-Ledger Account ID, Cost Center, and Revenue Month.
2. The Financial Engine then sums the Amount values for each group created in Step 1.
3. For each group created in Step 1, the Financial Engine locates the corresponding Sub-Ledger Account record by Sub-Ledger Account ID, Cost Center, and Revenue Month. If the Closed Date value is not NULL, the next Revenue Month record should be used. If the record does not exist, then it is created with an initial Balance value of \$0.00, Balance to GL value of \$0.00, and a Closed Date value of NULL.
4. The Financial Engine then updates each corresponding Sub-Ledger Account record by increasing the Balance value by the amount from Step 2, and by updating the Balance Time value to the current time. Also, the Financial Engine sets the Running Balance value equal to the previous revenue month's Running Balance plus the current month's Balance.
5. The Financial Engine then updates each record in the Journal Transaction Table from Step 1 by setting the Rolled Up Date value to the current date.
6. The Financial Engine then posts a “SUBLEDGER\_ROLLEDUP” control activity message. The Message should contain the following data:
  - Sub-Ledger Rollup element data, including:  
Start Time, Stop Time, and Sub-Ledger Account ID, and
  - Sub-Ledger Account element data, including:  
ID, Cost Center, Revenue Month, Previous Balance, Previous Balance Time, New Balance, New Balance Time, Running Balance, Closed Date, Activity Balance, Activity Count, Rolled Forward Balance, and Rolled Forward Count.

There should be a Sub-Ledger Account element for each Account/Cost Center/Revenue Month combination that was updated. ID, Cost Center, Revenue Month, Previous Balance, Previous Balance Time, New Balance, New Balance Time, Running Balance, and Closed Date are self-explanatory. Activity Balance should contain the total amount rolled up to this record. Activity Count should contain the total number of journal transactions that were rolled up to this record. Rolled Forward Count should contain the number of journal transactions that were destined for a previous revenue month but were rolled forward to this record because the previous revenue month was closed. Rolled Forward Balance should contain the total amount corresponding to these rolled forward transactions.

## Using the Sub-Ledger Roll-Up Command Line Program

As noted above, the Sub-Ledger Roll-Up function can be performed using a command line program. The Sub-Ledger Roll-Up command line program (UPSBLDGR.EXE) uses the syntax shown below. Parameter switches are case insensitive (i.e. you can enter them in either upper or lower case (-c or -C)). If a parameter includes a space, you must enclose it in quotes (for example, -s "11/01/1999 12:00:00"). The syntax is:

```
upsbldgr.exe -d <connectstring> [-q <qualifier>] [-lcfg logging configuration filename]
```

The command must be entered on one line. Also, you must either change to the directory in which the program is stored (typically, \LODESTAR\bin) before entering the command, or specify the path in the command. To view a list of all parameters on-screen, type **upsbldgr -?** at the command prompt.

Parameter	Description
-d	<p><i>connectstring</i> is database connection information for the Oracle Utilities Data Repository. This parameter is <b>required</b> and must be in one of the following formats:</p> <p>For Oracle databases:</p> <pre>"Data Source=&lt;data_source&gt;;User ID=&lt;user_id&gt;;Password=&lt;password&gt;;LSProvider=ODP;"</pre> <p>where:</p> <ul style="list-style-type: none"> <li>• &lt;data_source&gt; is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\&lt;machine&gt;\oracle\network\admin directory)</li> <li>• &lt;user_id&gt; is the user ID for the database connection</li> <li>• &lt;password&gt; is the password for the supplied user ID.</li> </ul>
-q	<i>qualifier</i> is the optional qualifier for the data source.
-lcfg	<i>logging configuration filename</i> Name of an optional logging configuration file that specifies where error and log messages are sent. If you omit this parameter, the application creates a log file named UPSBLDGR.LOG in the LOG directory.

## General Ledger Update

The General Ledger Update function outputs records from the Sub-ledger Account table to update an external general ledger system. This process can be performed using a command line program (see **General Ledger Update Processing** on page 5-22) or by an external system through an interface.

### General Ledger Update Processing

At scheduled intervals the records in the Sub-Ledger Account table should be output to a General Ledger File for updating the General Ledger. The steps outlined below describe the general ledger update process.

1. For each sub-ledger account mapped to a general ledger account, the Financial Engine checks the collection of mapped sub-ledger accounts to determine if the sub-ledger account has already been added. If so, it is an error. If not, the Financial Engine adds the sub-ledger account to the collection.
2. The Financial Engine selects all records in the Sub-Ledger Account table where the Revenue Month equals the given revenue month provided (or the prior calendar month/year if not provided), orders the records by sub-ledger account id, and creates a collection of sub-ledger accounts.
3. For each Sub-Ledger account in the collection, the Financial Engine checks to determine if the Sub-Ledger account has been mapped.
4. If a Sub-Ledger account has been mapped: The Financial Engine determines the general ledger account that the sub-ledger account is being mapped to. For each sub-ledger account in the mapped collection that is associated with this general ledger account, the Financial Engine does the following:
  - Finds the Sub-Ledger account in the Sub-Ledger account collection.
  - If the Sub-Ledger account is NOT usage, the Financial Engine adds its Balance to the Balance of the general ledger account, adds its Balance to GL to the Old Balance to GL of the general ledger account, and adds the difference between the Balance and Balance to GL (Balance - Balance to GL) to the Current Balance to GL of the general ledger account.
  - If the Sub-Ledger account is usage, the Financial Engine adds its Balance to the Quantity of the general ledger account, adds its Balance to GL to the Old Quantity to GL of the general ledger account, and adds the difference between the Balance and Balance to GL (Balance - Balance to GL) to the Current Quantity to GL of the general ledger account.
5. If sub-ledger account has not been mapped: The Financial Engine copies the Sub-Ledger account directly to the general ledger account including copying its Balance to the Balance of the general ledger account, copying its Balance to GL field to the Old Balance to GL field of the general ledger account, and copying the difference between the Balance and Balance to GL (Balance - Balance to GL) to the Current Balance to GL field of the general ledger account.
6. The Financial Engine updates the Balance to GL of each Sub-Ledger account processed to be the same as the Balance. Once each Sub-Ledger account has been processed, the Financial Engine removes it from the mapped Sub-Ledger account collection and deletes it, and removes it from the Sub-Ledger account collection and deletes it.
7. The Financial Engine posts a “GENLEDGER\_UPDATED” control activity message. The message should include the General Ledger Parameters, Sub-Ledger Count, and General Ledger Count.

The Sub-Ledger Count and the General Ledger Count are the number of Sub-Ledger accounts and General Ledger accounts processed, respectively.

8. When processed in batch mode using the command line program, the Financial Engine adds the processed General Ledger account to the GENLEDGERFILE.xml. If a General Ledger filename is passed to the command line program, the Financial Engine uses it. Otherwise, the Financial Engine generates the filename as “gnldgr” + the Revenue Month (as “YYYYMO”) + “.xml”.

## Using the General Ledger Update Command Line Program

As noted above, the general ledger update function can be performed using a command line program. The General Ledger Update command line program (UPGNLDGR.EXE) uses the syntax shown below. Parameter switches are case insensitive (i.e. you can enter them in either upper or lower case (-c or -C)). If a parameter includes a space, you must enclose it in quotes (for example, -s “11/01/1999 12:00:00”). The syntax is:

```
upgnldgr.exe -d <connectstring> [-q <qualifier>] -i <genledgerparams> -o <genledgerfile>
[-lcfg logging configuration filename]
```

The command must be entered on one line. Also, you must either change to the directory in which the program is stored (typically, \LODESTAR\bin) before entering the command, or specify the path in the command. To view a list of all parameters on-screen, type **upgnldgr -?** at the command prompt.

Parameter	Description
-d	<p><i>connectstring</i> is database connection information for the Oracle Utilities Data Repository. This parameter is <b>required</b> and must be in one of the following formats:</p> <p>For Oracle databases:</p> <pre>"Data Source=&lt;data_source&gt;;User ID=&lt;user_id&gt;;Password=&lt;password&gt;;LSProvider=ODP;"</pre> <p>where:</p> <ul style="list-style-type: none"> <li>• &lt;data_source&gt; is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\&lt;machine&gt;\oracle\network\admin directory)</li> <li>• &lt;user_id&gt; is the user ID for the database connection</li> <li>• &lt;password&gt; is the password for the supplied user ID.</li> </ul>
-q	<i>qualifier</i> is the optional qualifier for the data source.
-i	<i>genledgerparams</i> is an XML file containing general ledger parameters.
-o	<i>genledgerfile</i> is an XML output file (GENLEDGERFILE.XML unless otherwise specified).
-lcfg	<i>logging configuration filename</i> Name of an optional logging configuration file that specifies where error and log messages are sent. If you omit this parameter, the application creates a log file named UPGNLDGR.LOG in the LOG directory.

## Balancing Controls

The Balancing Controls function operates on three levels: the transaction level, the journal level, and the account level. Transaction-level balancing is triggered automatically every time a transaction is posted or cancelled to verify that the corresponding account is still in balance. Journal-level balancing is used at scheduled intervals to balance records in the Journal Transaction Table against corresponding records in the Transaction and Credit Application tables. Account-level balancing is used at scheduled intervals to balance the financial transaction records for a user-specified subset of accounts in the database.

### Transaction Balancing

Every time that a transaction is posted or cancelled by the Financial Engine, the transaction balancing function is triggered to verify that the account is still in balance. This process occurs just after the transaction record is created, but before the record is inserted into the Transaction Table (see **Post Transaction** on page 5-9). The steps outlined below describe the transaction balancing process.

1. The Financial Engine finds the Balance Time and Activity values for the parent Account record.
2. The Financial Engine then finds all previously posted Transaction records for the Account whose Balance is not NULL and Transaction Time is greater than the Balance Time in Step 1.
3. The Financial Engine then finds all previously cancelled Transaction records for the Account whose Balance is not NULL and Cancel Time is greater than the Balance Time in Step 1.
4. The Financial Engine then sums the Amount values of all posted “Charge” Transaction records and subtracts from that the Amount values of all posted “Credit” Transaction records in Step 2.
5. The Financial Engine then sums the Amount values of all cancelled “Credit” Transaction records and subtracts from that the Amount values of all cancelled “Charge” Transaction records in Step 3.
6. The Financial Engine then verifies that the sum of the amounts calculated in steps 4 and 5 equals the Activity amount in Step 1. If not, the process stops and the Financial Engine returns an error message.
7. Last, the Financial Engine updates the Account Activity value to include any activity (Balance is not NULL) from the current transaction (add “Charge” Transaction Amount value or subtract “Credit” Transaction Amount value, or do the exact opposite for cancelled Transaction records).

## Journal Balancing

At scheduled intervals the records in the Journal Transaction Table will be balanced against records in the Transaction and Credit Application tables.

Journal balancing can be performed using a command line program or by an external system through an interface. Using either method requires entering specific parameters, including:

- Start Time
- Stop Time
- Revenue Month

The Start Time, Stop Time and Revenue Month filter the balance activity over their respective parameters. These parameters are included in an XML string that is passed to the Financial Engine either through the command line program (see below) or an interface.

### Journal Balancing Processing

1. The Financial Engine locates all Transaction and Credit Application records with a Transaction Time or Cancel Time value within the provided Start and Stop times (inclusive) and a Revenue Month value equal to the given revenue month (if provided).
2. The Financial Engine then groups the records located in Step 1 above by Subledger Account ID (both Debit and Credit), Cost Center, and Revenue Month, summing the Amount values for debits and subtracting the Amount values for credits within each group.
3. The Financial Engine then locates all the Journal Transaction records with a Transaction Time value within the provided time range (inclusive) and a Revenue Month value equal to the given revenue month (if provided).
4. The Financial Engine then groups the records from Step 3 by Sub-Ledger Account ID, Cost Center, and Revenue Month, summing the Amount values for debits and subtracting the Amount values for credits within each group.
5. The Financial Engine then compares the counts and the amounts for matching groups from steps 2 and 4 (except those records where the Sub-Ledger Account ID is null). If any do not match, a “JOURNAL\_BALANCE\_ERROR” message is posted. This message should include the following:
  - Journal Account data, including:
    - ID, Cost Center, Revenue Month, Transaction Count, Transaction Amount, Credit Application Count, Credit Application Amount, Journal Count, and Journal Amount.

ID, Cost Center, and Revenue Month are self-explanatory. Transaction Count is the total number of financial transaction records that were journaled for this Account, Cost Center, and Revenue Month combination. Transaction Amount is the sum of the amounts for these transactions. Credit Application Count is the total number of Credit Application records that were journaled for this Account, Cost Center, and Revenue Month combination. Credit Application Amount is the sum of the amounts for these transactions. Journal Count is the number of journal transactions that resulted. Journal Amount is the total amount that was journaled.
6. The Financial Engine then posts a “JOURNAL\_BALANCED” control activity message. The message should include the following:
  - Journal Balanced data, including:
    - Journal Parameters, Start Time, Stop Time, and
  - Journal Account data, including:
    - ID, Cost Center, Revenue Month, Transaction Count, Transaction Amount, Credit Application Count, Credit Application Amount, Journal Count, Journal Amount.

ID, Cost Center, and Revenue Month are self-explanatory. Transaction Count is the total number of financial transaction records that were journaled for this Account, Cost Center, and Revenue Month combination. Transaction Amount is the sum of the amounts for these transactions. Credit Application Count is the total number of financial transaction records that were applied as credits for this Account, Cost Center, and Revenue Month combination. Credit Application Amount is the sum of the amounts for these transactions. Journal Count is the number of journal transactions that resulted. Journaled Amount is the total amount that was journaled.

## Using the Balance Journal Command Line Program

As noted earlier, the Journal Balancing function can be performed using a command line program. The Balance Journal command line program (BALJRNL.EXE) uses the syntax shown below. Parameter switches are case insensitive (i.e. you can enter them in either upper or lower case (-c or -C)). If a parameter includes a space, you must enclose it in quotes (for example, -s "11/01/1999 12:00:00").

```
baljrnl.exe -d <connectstring> [-q <qualifier>] [-lcfg logging configuration filename]
-@ <xmlInputFile
```

The command must be entered on one line. Also, you must either change to the directory in which the program is stored (typically, \LODESTAR\bin) before entering the command, or specify the path in the command. To view a list of all parameters on-screen, type **BALJRNL -?** at the command prompt.

Parameter	Description
-d	<p><i>connectstring</i> is database connection information for the Oracle Utilities Data Repository. This parameter is <b>required</b> and must be in one of the following formats:</p> <p>For Oracle databases:</p> <p>"Data Source=&lt;data_source&gt;;User ID=&lt;user_id&gt;;Password=&lt;password&gt;;LSProvider=ODP;"</p> <p>where:</p> <ul style="list-style-type: none"> <li>• &lt;data_source&gt; is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\&lt;machine&gt;\oracle\network\admin directory)</li> <li>• &lt;user_id&gt; is the user ID for the database connection</li> <li>• &lt;password&gt; is the password for the supplied user ID.</li> </ul>
-q	<i>qualifier</i> is the optional qualifier for the data source.
-lcfg	<i>logging configuration filename</i> Name of an optional logging configuration file that specifies where error and log messages are sent. If you omit this parameter, the application creates a log file named BALJRNL.LOG in the LOG directory.
-@	<i>xmlInputFile</i> is an XML file that contains specific arguments used to perform the journal balancing.

## Account Balancing

At scheduled intervals, the financial transaction records for a unique subset of all Accounts in the database will be balanced. A batch process (command line program) will be configured to do this so that over a period of one billing cycle (usually one month) all Accounts in the database are



balanced. The process does not specify how the Accounts are to be partitioned into subsets, but allows users to configure a query to be used as the basis for what Accounts are balanced. This query must have a select list containing only the unique ID values for the Accounts to be balanced. This query is included in an XML string that is passed to the Financial Engine either through the command line program (see below) or an interface.

## Account Balancing Processing

1. The Financial Engine locates all appropriate Account records based on the query provided.
2. For each account record, the Financial Engine locates all the posted and cancelled Transaction records whose Balance is not NULL that occurred before the Account's Balance Time. The Financial Engine then adds all the posted "Charge" type record Amounts and cancelled "Credit" type record Amounts and subtracts from that all posted "Credit" type record Amounts and all cancelled "Charge" type record Amounts.
3. For each account record, the Financial Engine locates all the posted and cancelled Transaction records whose Balance is not NULL that occurred after the Account's Balance Time. It then adds all the posted "Charge" type record Amounts and cancelled "Credit" type record Amounts and subtracts from that all posted "Credit" type record Amounts and all cancelled "Charge" type record Amounts.
4. For each account record, the Financial Engine locates all non-cancelled Transaction records whose Balance is not NULL. It then adds all "Charge" type record Balances and subtracts from that all "Credit" type record Balances.
5. If the amount from Step 2 matches the LASTBALANCE value of the account and the amount from Step 3 matches the NEWACTIVITY value of the account and the amount from Step 4 matches the sum of the LASTBALANCE and NEWACTIVITY values of the account then update the BALANCE.
6. If the amount from Step 2 matches the Last Balance value of the account, the amount from Step 3 matches the New Activity value of the account, and the amount from Step 4 matches the sum of the Last Balance and New Activity values of the account, the Financial Engine updates the Balance value to the sum of the current balance plus the current activity, updates the Balance Time to the current time, and updates the Activity value to zero.
7. If the amounts in Step 5 do not match, the Financial Engine posts an "ACCOUNT\_BALANCE\_ERROR" message. The message should include:
  - The unique ID of the Account,
  - The Account's Balance and Activity, and
  - The calculated Balance and Activity.
8. After the account balancing process is complete, the Financial Engine posts an "ACCOUNTS\_BALANCED" control activity message. The message should include the following:
  - The query,
  - Start Time and Stop Time,
  - The number of Accounts balanced, and
  - The number of Account balance errors.

## Using the Balance Accounts Command Line Program

As noted earlier, the Account Balancing function can be performed using a command line program. The Balance Accounts command line program (BALACCT.EXE) uses the syntax shown below. Parameter switches are case insensitive (i.e. you can enter them in either upper or lower case (-c or -C)). If a parameter includes a space, you must enclose it in quotes (for example, -s "11/01/1999 12:00:00")

**balacct.exe** -d <connectstring> [-q <qualifier>] [-lcfg *logging configuration filename*]  
 -@ *xmlInputFile*

The command must be entered on one line. Also, you must either change to the directory in which the program is stored (typically, \LODESTAR\bin) before entering the command, or specify the path in the command. To view a list of all parameters on-screen, type **balacct -?** at the command prompt.

Parameter	Description
-d	<p><i>connectstring</i> is database connection information for the Oracle Utilities Data Repository. This parameter is <b>required</b> and must be in one of the following formats:</p> <p>For Oracle databases:</p> <p>"Data Source=&lt;data_source&gt;;User ID=&lt;user_id&gt;;Password=&lt;password&gt;;LSProvider=ODP;"</p> <p>where:</p> <ul style="list-style-type: none"> <li>• &lt;data_source&gt; is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\&lt;machine&gt;\oracle\network\admin directory)</li> <li>• &lt;user_id&gt; is the user ID for the database connection</li> <li>• &lt;password&gt; is the password for the supplied user ID.</li> </ul>
-q	<i>qualifier</i> is the optional qualifier for the data source.
-lcfg	<i>logging configuration filename</i> Name of an optional logging configuration file that specifies where error and log messages are sent. If you omit this parameter, the application creates a log file named BALACCT.LOG in the LOG directory.
-@	<i>xmlInputFile</i> is an XML file that contains the query to be used to perform the account balancing.

# Chapter 6

---

## Billing

This chapter describes the functionality of the Billing module of the Oracle Utilities Receivables Component including an overview of some of the database tables used by the Billing module, and explanations of each of the Billing functions performed, including:

- Post/Cancel Charge or Credit,
- Post/Cancel Tax,
- Create/Cancel Installment Plan,
- Post/Cancel Installment,
- Post/Cancel Deposit
- Post/Cancel Deposit Interest
- Apply/Unapply Deposit
- Post Penalty
- Post Statement
- Post Bill
- Cancelling Transactions
- Bill Statementing

## Billing Database Tables

Of the tables described in **Chapter 3: The Oracle Utilities Receivables Component Database**, the Billing module uses the Budget tables (p. 3-3) when processing billing transactions. The descriptions below expand on those for the Budget Type and Budget Plan tables found in Chapter Two.

### **Service Type Table**

The Service Type Table is used to define all service-oriented businesses or commodities provided by the client operating company. Typical service types might include “Electric”, “Gas”, or “Cable”.

### **Service Plan Table**

The Service Plan Table associates an account with a service type. An account may have several service plans active at any one time.

### **Budget Type Table**

Records in the Budget Type Table represent specific types of budget plans recognized by the application. The application uses budget types to determine how true-ups should be handled.

### **Budget Plan Table**

Records in the Budget Plan Table indicate that a given account is on a particular budget plan for a specific time period.

### **Installment Plan Table**

Records in the Installment Plan table represent installment plans for an account.

### **Tax Record Table**

Records in the Tax Record table associate a tax transaction with zero or more taxed transactions.

### **Deposit Table**

Records in the Deposit table represent deposits for an account.

## Billing Functions

The core function of the Billing module is to handle billing-related transactions through the Financial Engine. These transactions are sent to the Billing module from external systems such as Oracle Utilities Billing Component or other billing applications.

### Billing Function Processing

Each of the Billing module functions can be triggered from either an external system via an interface or through use of the Financial Management Statements of the Oracle Utilities Rules Language. In addition, some of the Billing module functions can be triggered through the Oracle Utilities Receivables Component user interface (see the *Oracle Utilities Billing Component User's Guide*). Regardless of the manner in which they are triggered, the functions of the Billing module operate in the same manner. When triggered, each function sends transaction data to the Financial Engine, which posts the transaction.

When the functions are triggered through an interface, the transaction data is sent in an XML string to the Financial Engine. When the functions are triggered by the Rules Language, the transaction data is generated during the processing of the rate schedule. In this case, some of the data is automatically provided based on the specific account and rate form being processed, while other data must be provided in the rate schedule itself.

The data generated by bill calculation is listed below, along with where the data comes from (either internal or via the rate schedule). Required data is marked with an asterisk (\*). All other data is optional.

Data Element	Source
Account*	Rate/ACCOUNTID or Internal - Account UID
Transaction ID	Rate/TRANSACTIONID (per Transaction Type)
Revenue Month	Rate/REVENUEMONTH or Internal - Bill Month
Note	Rate/NOTE
Charge or Credit*	Rate/CHARGEORCREDIT (“CH” or “CR”)
Defer Balance	Rate/DEFERBALANCE (“TRUE” or “FALSE”. The default is “FALSE”).
Amount*	Rate/AMOUNT
Currency*	Rate/CURRENCY
Billed Date*	Rate/BILLEDDATE
Due Date*	Rate/DUEDATE
Receivable Type	Rate/RECEIVABLETYPENAME
Charge Type	Rate /CHARGETYPEID
Rate Code*	Internal - Current Rate Schedule and Code
Operating Company	Rate/OPCOCODE
Jurisdiction	Rate/JURISCODE
Statement Date	Rate/STATEMENTDATE or Internal - READDATE
Invoice ID	Rate/INVOICEID
Invoice Date	Rate/INVOICEDATE
Bill Cycle Date*	Rate/BILLCYCLEDATE or Internal - READDATE

**Note:** Charge or Credit does not apply to the Post Budget Bill Charge, Post Installment, or Post Bill functions.

**Note:** Defer Balance only applies to the Post Charge or Credit, Post Tax, Post Service Charge, Post Installment Charge and Post Bill functions.

**Note:** Statement Date only applies to the Post Statement function and is optional for the Post Charge or Credit function.

## Billing Functions

---

	Bill History	Internal - UIDBILLHISTORY if available
<b>Note:</b> Service Plan does not apply to the Post Statement or Post Bill functions.	Service Plan*	Rate/SERVICEPLAN (Start Date, Service Type, Address 1, Address 2, Address 3, City, County, State, ZIP, LDC Account ID)
<b>Note:</b> Budget Plan does not apply to the Post Service Charge, Post Deferred Service Charge, Post Statement or Post Bill functions.	Budget Plan*	Rate/BUDGETPLAN (Start Date, Budget Type, Service Plan)
<b>Note:</b> Application Method does not apply to the Post Deferred Service Charge or Post Budget Service Charge, Post Budget Bill Charge, or Post Budget Bill Trueup functions.	Application Method	Rate/APPLICATIONMETHOD (“DEFERRED”, “IMMEDIATE”, or “INVOICEID”. The default is “DEFERRED”).
<b>Note:</b> Tax Rate only applies to the Post Tax function.	Tax Rate	Rate/TAXRATE
<b>Note:</b> Related Transaction only applies to the Post Tax function.	Related Transaction	Rate/TAXEDTRANSACTIONS (UIDTRANSACTION, Amount, Tax Amount, Tax Rate, Tax Exempt)
<b>Note:</b> Installment Plan only applies to the Post Installment function.	Installment Plan	Rate/UIDINSTALLMENTPLAN (either this or INSTALLMENTPLANNO is required).
<b>Note:</b> Installment Plan No. only applies to the Post Installment function.	Installment Plan No.	Rate/INSTALLMENTPLANNO (either this or UIDINSTALLMENTPLAN is required).
<b>Note:</b> <b>Deposit</b> only applies to the Post Deposit Interest and Post Deposit Application functions.	Deposit	Rate/UIDDEPOSIT (either this or DEPOSITTIME is required).
<b>Note:</b> <b>Deposit</b> Time only applies to the Post Deposit Interest and Post Deposit Application functions.	Deposit Time	Rate/DEPOSITTIME (either this or UIDDEPOSIT is required).

## Billing Functions

The functions of the Billing module pass the data outlined above to the Financial Engine which in turn processes the transactions.

### Post Charge or Credit

This function posts either charges or credit transactions against a specified account. The transaction may be either deferred or not deferred. An optional service plan or budget plan may be associated with the transaction. If a budget plan is provided, the plan's variance will be updated accordingly. This function performs the following steps:

1. The Billing module sets the transaction type to Charge or Credit (CHRGCRDT).
2. The Billing module triggers the Post Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.

The POST CHARGEORCREDIT Rules Language statement can be used to post a charge or credit as a single transaction. Some elements of the transaction are internally generated by the rate schedule when the POST CHARGEORCREDIT Statement executes.

See **Appendix D: Financial Management Rules Language Statements** for details on the use of the POST CHARGEORCREDIT Statement.

### Cancel Charge or Credit

This function cancels a previously posted charge or credit transaction. This function performs the following steps:

1. The Billing module sets the transaction type to Charge or Credit (CHRGCRDT).
2. The Billing module triggers the Cancel Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.

### Post Tax

The Billing module allows tax transactions to be posted against an account. The individual tax transactions can be related to one or more previously posted transactions (taxed transactions). The Tax Record table (see p. 4-2) maintains these relationships, along with the taxed transaction tax amount, tax rate (if provided), and a flag indicating whether or not the taxed transaction is tax exempt. This allows the system to automatically adjust taxes if a taxed transaction is later cancelled or adjusted. It also allows for more robust tax reporting.

The Post Tax function posts tax charge or credit transactions for a specified account. The transaction may be either deferred or not deferred. An optional service plan or budget plan may be associated with the transaction. If a budget plan is provided, the plan's variance will be updated accordingly. Additionally, the tax transaction may be associated with one or more previously posted transactions. This function performs the following steps:

1. The Billing module sets the transaction type to Tax (TAX).
2. For each related taxed transaction, the Billing module:
  - a. Determines the appropriate tax rate. This is either provided for taxed transaction or provided for tax transaction or is NULL.
  - b. Determines the tax amount. This is either provided or calculated from tax rate and taxed transaction amount and the charge or credit flag.
3. The Billing module determines the tax transaction amount. This is either provided or calculated from sum of individual taxed transaction amounts, less any tax exempt amounts.
4. The Billing module triggers the Post Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.

5. The Billing module inserts a record in the Tax Record table for each related taxed transaction with appropriate values as determined above.
6. If there are no related taxed transactions, the Billing module inserts a single record in the Tax Record with a NULL unique ID for an associated taxed transaction and an Amount equal to the tax transaction amount (as well as the Tax Rate if provided).

The POST TAX Rules Language statement can be used to post a tax charge or credit as a single transaction. Some elements of the transaction are internally generated by the rate schedule when the POST TAX Statement executes.

See **Appendix D: Financial Management Rules Language Statements** for details on the use of the POST TAX Statement.

## Cancel Tax

This function cancels a previously posted tax charge or credit transaction. This function performs the following steps:

1. The Billing module sets the transaction type to Tax (TAX).
2. The Billing module triggers the Cancel Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.
3. The Billing module cancels any tax adjustment transactions related to this tax transaction. (A tax adjustment transaction results when a non-exempt taxed transaction is cancelled.)
4. The Billing module updates the Cancelled column to “Y” in all related records in the Tax Record table for this tax transaction.

## Create Installment Plan

The Billing module provides methods to create installment plans for accounts as well as post individual installment charges related to the installment plan.

This function creates an installment plan related to a previously posted deferred charge transaction. This process can alternatively occur as a single transaction if the installment plan is related to the deferred charge when the deferred charge is initially posted. This function performs the following steps:

1. The Billing module verifies that the related transaction is a deferred charge. If not it is an error.
2. The Billing module sets the installment plan's Account to that of the deferred charge transaction.
3. The Billing modules sets the following installment plan values to the corresponding deferred charge transaction values, if not explicitly provided:
  - Receivable Type,
  - Charge Type,
  - Operating Company, and
  - Jurisdiction.
4. The Billing module sets the unique ID of the deposit associated with the installment plan equal to that of the deferred charge transaction.
5. The Billing module sets the Number of Installments (NUMINSTALLMENTS) value to 1 if not provided.
6. The Billing module sets the Total Amount (TOTALAMOUNT) and Remaining Amount (REMAMOUNT) of the installment plan equal to the transaction AMOUNT of the deferred charge.



7. The Billing module calculates the Installment Amount if not provided. This is equal to the Total Amount divided by the Number of Installments, or the Total Amount minus the First Amount divided by the Number of Installments minus 1 if the First Amount is provided.
8. The Billing module inserts the Installment Plan record into the database.

### Cancel Installment Plan

This function cancels a previously created installment plan. This function performs the following steps:

1. The Billing module triggers the Cancel Installment function (p. 4-7) for each installment transaction related to the installment plan.
2. The Billing module updates the Stop Time on the Installment Plan record to the current time.

### Post Installment

This function posts a non-deferred charge transaction related to a previously created installment plan against a specified account. This function performs the following steps:

1. The Billing module sets the transaction type to Installment (INST).
2. The Billing module sets the Charge or Credit flag (CHARGEORCREDIT) to Charge (CH) and the Deferred flag (DEFERRED) to false.
3. The Billing module determines the installment amount. This is either provided or the Remaining Amount (REMAMOUNT), the Installment Amount (INSTAMOUNT), or the First Amount (FIRSTAMOUNT), depending on the current state of the installment plan.
4. The Billing module sets the following transaction attributes equal to the corresponding attributes of the installment plan if not explicitly provided:
  - Transaction ID,
  - Receivable Type,
  - Charge Type,
  - Operating Company, and
  - Jurisdiction.
5. The Billing module sets the unique ID of the deposit associated with the transaction equal to that of the installment plan's.
6. The Billing module triggers the Post Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.
7. The Billing module updates the Remaining Installments (REMINSTALLMENTS) and Remaining Amount (REMAMOUNT) values of the installment plan accordingly.
8. The Billing module updates the Stop Time of the installment plan to the current time if the Remaining Amount is now zero.

The POST INSTALLMENT Rules Language statement can be used to post an installment as a single transaction. Some elements of the transaction are internally generated by the rate schedule when the POST INSTALLMENT Statement executes.

See **Appendix D: Financial Management Rules Language Statements** for details on the use of the POST INSTALLMENT Statement.

### Cancel Installment

This function cancels a previously posted installment transaction. This function performs the following steps:

1. The Billing module sets the transaction type to Installment (INST).

2. The Billing module triggers the Cancel Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.
3. The Billing module updates the Remaining Installments (REMINSTALLMENTS) and Remaining Amount (REMAMOUNT) values of the installment plan accordingly.
4. The Billing module updates the Stop Time of the installment plan to NULL if the Remaining Amount is now greater than zero.

## Post Deposit

This function posts either a non-deferred or deferred charge transaction against a specified account. The default is non-deferred, since a deferred deposit would typically have an installment plan created for it. This function performs the following steps:

1. The Billing module sets the transaction type to Deposit (DEP) and the Charge or Credit flag (CHARGEORCREDIT) to Charge (CH).
2. The Billing module triggers the Post Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.
3. The Billing module inserts a record in the Deposit table with Account and Amount values equal to that of the transaction as well as an Interest Rate (INTERESTRATE) value if provided. The Principal Balance (PRINCIPALBAL) and Interest Balance (INTERESTBAL) values are set to zero. The remaining values are set to NULL.

The POST DEPOSIT Rules Language statement can be used to post a deposit as a single transaction. Some elements of the transaction are internally generated by the rate schedule when the POST DEPOSIT Statement executes.

See **Appendix D: Financial Management Rules Language Statements** for details on the use of the POST DEPOSIT Statement.

## Cancel Deposit

This function cancels a previously posted deposit transaction. This function performs the following steps:

1. The Billing module sets the transaction type to Deposit (DEP).
2. The Billing module triggers the Cancel Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.
3. If the deposit has a current interest balance, the Billing module triggers the Apply Deposit function (p. 4-9) for the amount of the interest balance.

## Post Deposit Interest

This Interest function posts deferred credit transactions for a specified account representing an amount of interest accrual for an associated deposit. This function performs the following steps:

1. The Billing module sets the transaction type to Deposit Interest (DEPINT), the Charge or Credit flag (CHARGEORCREDIT) to Credit (CR), and the Deferred flag (DEFERRED) to True.
2. The Billing module triggers the Post Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.
3. The Billing module updates the Interest Balance (INTERESTBAL) and Last Interest Date (LASTINTERESTDATE) values for the deposit accordingly.

The POST DEPOSIT INTEREST Rules Language statement can be used to post deposit interest as a single transaction. Some elements of the transaction are internally generated by the rate schedule when the POST DEPOSIT INTEREST Statement executes.

---

See **Appendix D: Financial Management Rules Language Statements** for details on the use of the POST DEPOSIT INTEREST Statement.

### Cancel Deposit Interest

This function cancels a previously posted deposit interest transaction. This function performs the following steps:

1. The Billing module sets the transaction type to Deposit Interest (DEPINT).
2. The Billing module triggers the Cancel Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.
3. The Billing module updates the Interest Balance (INTERESTBAL) and Last Interest Date (LASTINTERESTDATE) values for the deposit accordingly.

### Apply Deposit

This function posts non-deferred credit transactions for a specified account representing an amount of the associated deposit balance that is applied to the account. This function performs the following steps:

1. The Billing module sets the transaction type to Deposit Application (DEPAPP), the Charge or Credit flag (CHARGEORCREDIT) to Credit (CR), and the Deferred flag (DEFERRED) to False.
2. The Billing module verifies that the transaction amount is less than or equal to the sum of the Principal Balance (PRINCIPALBAL) and Interest Balance (INTERESTBAL) values of the deposit.
3. The Billing module triggers the Post Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.
4. The Billing module updates the Interest Balance (INTERESTBAL), Principal Balance (PRINCIPALBAL), and Last Applied Date (LASTAPPLIEDDATE) values for the deposit accordingly. Deposit application amounts are always taken from the interest balance first, then the principal balance if more is specified.

The POST DEPOSIT APPLICATION Rules Language statement can be used to apply a deposit as a single transaction. Some elements of the transaction are internally generated by the rate schedule when the POST DEPOSIT APPLICATION Statement executes.

See **Appendix D: Financial Management Rules Language Statements** for details on the use of the POST DEPOSIT APPLICATION Statement.

### Unapply Deposit

This function cancels a previously posted deposit application transaction. This function performs the following steps:

1. The Billing module sets the transaction type to Deposit Application (DEPAPP).
2. The Billing module triggers the Cancel Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.
3. The Billing module updates the Principal Balance (PRINCIPALBAL), Interest Balance (INTERESTBAL), and Last Applied Date (LASTAPPLIEDDATE) values for the deposit accordingly.

## Post Penalty

This function posts penalties against an account. This function inserts a record in the Name Override History table that is used to indicate that a penalty was posted against the account. Penalty Code, Penalty Time, and Outstanding Amount values must be provided, and are used as the Override Code, Start Time, and Value column values of the Name Override History record respectively. This function performs the following steps:

1. The Billing module inserts a record in the Name Override History table for the given account and provided values. The Name column value should be the same as the Override Code. The Stop Time and String Value column values should be NULL.

## Post Statement

This function posts a single deferred statement transaction for an individual account. This transaction typically indicates the current balance amount for the account. This function performs the following steps:

1. The Billing module sets the transaction type to Statement (STMT).
2. The Billing module sets the Defer Balance flag (DEFERBALANCE) to True.
3. The Billing module triggers the Post Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.

The POST STATEMENT Rules Language statement can be used to post a statement as a single transaction. Some elements of the transaction are internally generated by the rate schedule when the POST STATEMENT Statement executes.

See **Appendix D: Financial Management Rules Language Statements** for details on the use of the POST STATEMENT Statement.

## Cancel Statement

This function cancels a previously posted statement transaction. This function performs the following steps:

1. The Billing module sets the transaction type to Statement (STMT).
2. The Billing module triggers the Cancel Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.

## Post Bill

This function posts a single bill (invoice) transaction for an individual account. This transaction not only indicates the actual invoice amount for the account, but also optionally triggers any necessary auto payment processing. This function performs the following steps:

1. The Billing module sets the transaction type to Bill (BILL) .
2. If not explicitly specified, the Billing module sets the application method to Immediate (IMMEDIATE).
3. If not explicitly specified, the Billing module sets the Defer Balance flag (DEFERBALANCE) to True.
4. The Billing module triggers the Post Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.
5. The Billing module determines the Bill Start Time (BILLSTARTTIME). This is either provided or is equal to the transaction time of the last transaction for this account with the same transaction id.
6. The Billing module determine the Bill Stop Time (BILLSTOPTIME). This is either provided or is equal to the transaction time of this transaction.

7. If the Invoice ID (INVOICEID) is provided, the Billing module updates the Invoice ID and Invoice Date (INVOICEID and INVOICEDATE) or the Cancel Invoice ID and Cancel Invoice Date (CANCELINVOICEID and CANCELINVOICEDATE) values for all previously posted or cancelled transactions that occurred within the Bill Start Time and Bill Stop Time (BILLSTARTTIME and BILLSTOPTIME) parameters whose Invoice ID or Cancel Invoice ID (INVOICEID or CANCELINVOICEID) is NULL.
8. If the Due Date (DUEDATE) is provided, the Billing module updates the Bill or Paid Date and Due Date (BILLEDORPAIDDATE and DUEDATE) values for all previously posted or cancelled charge transactions that occurred within the Bill Start Time and Bill Stop Time (BILLSTARTTIME and BILLSTOPTIME) parameters whose Due Date (DUEDATE) is NULL.
9. If the transaction is a charge and the amount is greater than zero, the Billing module triggers the Process AutoPayment function of the Remittance module (see **Chapter 7: Remittance**), using the bill calculation data when processing the transaction in the Remittance module, and processing is complete.
10. If the amount is less than zero, the Billing module checks whether the account is on an automatic payment plan and if Auto Refund flag is set to Yes ('Y'). If not, processing is complete.
11. The Billing module changes the Amount value to positive and triggers the Post Refund function of the Maintenance module (see **Chapter Six: Maintenance**). The Billing module uses the account data provided to the Post Bill function when triggering the PostRefund function.
12. The Billing module then triggers the Process AutoPayment function of the Remittance module (see **Chapter Five: Remittance**) and inserts a record in the AutoPayment table. The Billing module uses the account data provided to the Post Bill function when triggering the Process AutoPayment function.

The POST BILL Rules Language statement can be used to post a bill as a single transaction. Some elements of the transaction are internally generated by the rate schedule when the POST BILL Statement executes.

See **Appendix D: Financial Management Rules Language Statements** for details on the use of the POST BILL Statement.

## Cancel Bill

This function cancels a previously posted bill transaction. This function performs the following steps:

1. The Billing module sets the transaction type to Bill (BILL).
2. The Billing module triggers the Cancel Transaction function of the Financial Engine, and uses the bill calculation data (outlined above) when processing the transaction.

## Obtaining Information for Bill Printing

Each rate schedule for an account can forward charge details to the statementing interface as it calculates them. However, other financial transaction information that is not calculated by the rate schedule needs to be available so that it too can be forwarded to the statementing interface. This includes any payments, adjustments, transfers, etc. that occurred since the last bill.

Obtaining financial transaction data is performed using the List/Query functionality of Data Manager (see **Chapter 8: Working with Lists and Queries** in the *Data Manager User's Guide*). As transactions are posted from the Rules Language, the Bill Cycle Date and (unique) Rate Form columns are always set. Transactions from other sources may optionally provide a Bill Cycle Data value; however, the Rate Form value will always be null.

When accessed from a billing rate schedule, the following query returns all current transactions for this bill (that were not created by the Billing process):

```
"SELECT * FROM TRANSACTION
WHERE UIDACCOUNT = ?
AND
UIDRATEFORM IS NULL
AND
(BILLCYCLEDATE IS NULL OR BILLCYCLEDATE = ?) "
```

The LISTUPDATE Rules Language function can then be used to update the Bill Cycle Date (as well as the Billed Date or Due Date if necessary) of the resulting records to associate them with this bill.

## Obtaining Other Account Bill Information

The Get Bill Info function of the Billing module is used to return billing information about an account, including the account's current and past due balances. This function requires an XML string containing all of the information necessary to identify the account and the date for which to get the information. The only information that is required in this string is the (unique) Account ID and the Bill or Paid Date (which will default to the current date if not provided). The steps outlined below describe the process by which account bill information is obtained.

1. The Billing module sums the Balance of all charge Transaction records for the account with a Due Date value less than or equal to the provided (or defaulted) bill date, a Cancel Time value of NULL, and a Balance value greater than zero. This is the Past Due Balance.
2. The Billing module adds the Balance and Activity values of the account. This is the Current.
3. The Billing module creates and returns an XML string that contains the account's information. Both the Current Balance and Past Due Balance elements are added to this structure to support the return information.

The FMGETBILLINFO Rules Language function can be used to trigger the Get Bill Info function. The data necessary to identify the account is passed to the function as function parameters when the FMGETBILLINFO function is executed. The function returns the same data as outlined above.

See **Appendix D: Financial Management Rules Language Statements** for details on the use of the FMGETBILLINFO function.

## Canceling Transactions

Occasionally, cancellation of previously posted transactions created from a rate schedule is necessary. The Cancel Transactions function is used to cancel all transactions that were created by a rate schedule for an individual account. This function is triggered when the Bill Correction module of Oracle Utilities Billing Component is run in the CANCEL or CANCEL/REBILL mode (see the *Oracle Utilities Billing Component User's Guide* for more information). The steps outlined below describe the canceling transaction process.

1. The Billing module locates all Transaction records with unique Account ID and UIDBILLHISTORY (from an associated Bill History record) values that match those provided, and that have a non-Null Rate Form value and a Cancel Time value of Null.
2. The Billing module invokes the appropriate "Cancel" function on each Transaction found in Step 1 above, based on its Transaction Type value.

## Deprecated Functions

The following functions have been replaced by the Post Charge Or Credit and Cancel Charge Or Credit functions (as appropriate). Although these functions are still supported, using the Post Charge or Credit function with specified data as appropriate will produce the same results. Refer to **Deprecated Statements** on page D-36 for more information on using these statements via the Oracle Utilities Rules Language.

### **Post Service Charge and Cancel Service Charge.**

Use the Post Charge Or Credit function with an associated service plan to produce the same results as these functions.

### **Post Deferred Service Charge and Cancel Deferred Service Charge.**

Use the Post Charge Or Credit function with an associated service plan and DEFERBALANCE set to true to produce the same results as these functions.

### **Post Budget Service Charge and Cancel Budget Service Charge.**

Use the Post Charge Or Credit function with an associated budget plan and DEFERBALANCE set to true to produce the same results as these functions.

### **Post Budget Bill Charge and Cancel Budget Bill Charge.**

Use the Post Charge Or Credit function with an associated budget plan to produce the same results as these functions.

### **Post Budget Bill TrueUp and Cancel Budget Bill TrueUp.**

Use the Post Charge Or Credit function with an associated budget plan to produce the same results as these functions.

### **Post Installment Charge and Cancel Installment Charge.**

Use the Post Charge Or Credit function to produce the same results as these functions.



# Chapter 7

---

## Remittance

This chapter describes the Remittance functions of the Oracle Utilities Receivables Component, including an overview of some of the database tables used by the Remittance module, and explanations of each of the Remittance functions performed, including:

- **Payment Processing:** Processing of all incoming payments from all supported sources. It is equipped to handle many possible scenarios, including unpostable payments and transferred and written-off accounts.
- **Payment Exception Processing:** Processing of all payment exceptions, including NSF, closed accounts, and credit limits.
- **Automatic Payment Processing:** Processing of recurring direct debit and credit card charges and refunds.

## Remittance Database Tables

In addition to the tables described in **Chapter 3: The Oracle Utilities Receivables Component Database**, the Remittance module also uses a set of tables specifically designed to store data related to payments, including payment sources, automatic payments, batch payments, and payment assistance. Each of these tables is described below.

### Payment Tables

Payment tables store data associated with specific payments, batch payments, payment files, and automatic payments.

#### Note

Values in the Payment Table, the Batch Payment Table, and the Automatic Payment Table should be created **only** by the Financial Engine.

#### Payment Table

The Payment Table contains information about a payment for a specific account.

#### Payment Source Table

Records in the Payment Source Table represent the source of payments coming into the system, including automatic payments. This represents the vendor from whom the payment was sent such as Chase, Lockbox, etc.

#### Payment Method Table

Records in the Payment Method Table represent the different payment methods (Check, Visa, Direct Debit) for payments coming into the system, including automatic payments.

#### Auto Payment Plan Table

Records in the Auto Payment Plan Table represent automatic payment plans for a given account for a specified time period. The existence of a current automatic payment plan for an account triggers an automatic draft of the specified automatic payment vendor account when the account is billed.

#### Automatic Payment Table

Records in this table represent an instance of an automatic payment. Payments are collected into batches, grouped by vendor, and are typically transmitted to the vendor electronically at scheduled intervals.

Records in this table represent a batch payment from some payment source. Each record contains one or more payments on multiple accounts, and identifies the payment source.

### Payment Assistance Tables

Payment assistance tables store data associated with payment assistance agencies, assistance programs, assistance plans, and assistance pledges.

#### Assistance Agency Table

Records in this table represent payment assistance agencies in the system. Assistance agencies are agencies offering payment assistance to customers.

**Assistance Program Table**

Records in this table represent payment assistance programs in the system. Assistance programs are specific payment assistance programs offered by assistance agencies.

**Assistance Program Receivable Type Table**

Records in this table associate payment assistance programs with Receivable Types in the system.

**Assistance Plan Table**

Records in this table represent individual payment assistance plans in the system. Assistance plans are specific instances of assistance programs applied to an account.

**Assistance Pledge Table**

Records in this table represent individual payment assistance pledges in the system. Assistance pledges are pledges of specified amounts associated with specific assistance plans. These records should be created for an account when the plan is established.

# Remittance Functions

The core function of the Remittance module is to send payment transactions to the Financial Engine. These transactions are sent to the Remittance module from external systems through an interface.

## Payment Data

In addition to Data Source, Account, and Transaction data, the Remittance module also uses three specific types of payment data. These are Payments, Batch Payments, and Payment files. Payment files contain a number of batch payments, which in turn contain a number of payments.

### Payments

A payment represents a single payment posted to an account. A payment in the Remittance module includes the following:

- A unique ID for the payment,
- Application Method (indicates the credit application method for the payment),
- Default Account ID (the default account ID for the payment. This is used if the actual account ID cannot be determined, or if the payment cannot be posted to the intended account. This might occur in the case of an invalid account ID, if the account is out of balance, or if the account's Receivable Status is "UNCOLLECTIBLE"),
- Transaction ID (the transaction ID for the payment. If not provided, the default Transaction ID for the Payment Transaction Type will be used),
- Revenue Month (optional revenue month for post payment. If not provided, the current month will be used),
- Note (an optional note associated with posted payment),
- Account ID (the unique Account ID for the payment),
- Payment Source Code (the payment source code for the payment),
- Batch Payment (Unique ID of the batch payment from where this payment came),
- Batch Cancel (Unique ID of the batch payment that cancelled this payment),
- Payment ID (a unique ID for the payment within the batch or the payment source),
- Date (the date of the payment. If this is not provided, the current date is used),
- Amount (the amount of the payment, including the currency code for the payment),
- Payment Method Code (the payment method code for the payment),
- Institution (the institution from which the payment is drawn),
- Account Name (name on the account),
- Account No (a unique identifier of the account within the above institution),
- Account ZIP (ZIP code of account),
- Check No (the Check number for the above account),
- Expiration Date (expiration date for credit card account),
- RTN (routing transit number for direct debit account),
- Autopayment Time Stamp (transaction time that associated autopayment record was created),
- Autopayment Date (scheduled payment date of associated autopayment),
- Bill Date (bill date of associated bill transaction when autopayments are used),

- Invoice ID (associated invoice ID for the payment),
- Invoice Date (associated invoice date for the payment),
- Misc 1: Optional miscellaneous payment attribute,
- Misc 2: Optional miscellaneous payment attribute,
- Misc 3: Optional miscellaneous payment attribute,
- Cancel Revenue Month (the optional revenue month for a cancelled payment. If not provided, the current month will be used),
- Cancel Reason Code (an optional reason code for canceling payment),
- Cancel Note (an optional note associated with cancelled payment), and
- Post Penalty (indicates that a penalty should be posted for a cancelled payment).
- Assistance Program ID (associated Assistance Program for the payment)

## Batch Payments

A batch payment contains a number of payments. A batch payment in the Remittance module includes the following:

- A unique identifier for the batch payment,
- Batch No (a unique number of the batch payment within the payment file or payment source),
- Cancel (indicates that the batch should be processed in Cancel mode),
- Restart (indicates that the batch should be processed in Restart mode),
- Default Account ID (the optional default account ID for the payments in the batch). Forwarded to individual payments if not already provided,
- Transaction ID (the optional Transaction ID for the payments in the batch). Forwarded to individual payments if not already provided,
- Revenue Month (the optional revenue month for the payments in the batch). Forwarded to individual payments if not already provided,
- Payment Source Code (the payment source code for the batch). Required for processing. Forwarded to individual payments if not already provided,
- Payment File (the payment file from where the batch came). Optional for processing. This includes the unique file pathname of the payment file,
- Date (the date of the batch payment). Optional for processing. Forwarded to individual payments if not already provided,
- Payment Method Code (the payment method code for the batch). Optional for processing. Forwarded to individual payments if not already provided,
- Number of Payments (the total number of payments in the batch). Optional for processing,
- Amount (the total amount of all the payments in the batch, including the currency code for the payments in the batch). Optional for processing,

**Note:** All payments within a batch must use the same currency.

- Cancel Revenue Month (the optional revenue month for a cancelled batch. If not provided, the current month will be used),
- Cancel Reason Code (an optional reason code for canceling batch),
- Cancel Note (an optional note associated with cancelled batch), and

- Post Penalty (indicates that a penalty should be posted for payments associated with a cancelled batch).
- Max Errors (the maximum number of payment errors allowed prior to stopping the process). Optional for processing, and
- Payments (the individual payments in the batch). Required for processing.

## Payment Files

A payment file contains a number of batch payments. A payment file in the Remittance module includes the following:

- Name (the unique file path name of the payment file) Required for processing,
- Cancel (indicates that the file should be processed in Cancel mode),
- Restart (indicates that the file should be processed in Restart mode),
- Default Account ID (an optional default account ID for the batch payments in the payment file) Forwarded to individual batch payments if not already provided,
- Transaction ID (an optional Transaction ID for the batch payments in the payment file) Forwarded to individual batch payments if not already provided,
- Revenue Month (the optional revenue month for the batch payments in the payment file) Forwarded to individual batch payments if not already provided,
- Payment Source Code (the required payment source code for the payment file) Forwarded to individual batch payments if not already provided,
- Date (the optional date of the payment file) Forwarded to individual batch payments if not already provided,
- Payment Method Code (an optional payment method code for the payment file) Forwarded to individual batch payments if not already provided,
- Number of Batches (the total number of batches in the payment file) Optional for processing,
- Number of Payments (the total number of payments in the payment file) Optional for processing,
- Amount (the total amount of all payments (or batch payments) in the payment file, including the currency code for the payment file) Optional for processing,

**Note:** All batches within a payment file must use the same currency.

- Cancel Revenue Month (the optional revenue month for a cancelled file. If not provided, the current month will be used),
- Cancel Reason Code (an optional reason code for canceling file),
- Cancel Note (an optional note associated with cancelled file), and
- Post Penalty (indicates that a penalty should be posted for payments associated with a cancelled file).
- Max Errors (the maximum number of batch payment errors allowed prior to stopping the process) Optional for processing,
- Max Errors Per Batch (the maximum number of payment errors allowed per batch payment prior to stopping the batch payment process) Forwarded to individual batch payments if not already provided. Optional for processing, and
- Batch Payment (individual batch payments in the payment file) Required for processing.

## Remittance Function Processing

Each of the Remittance module functions is triggered from an external system via an interface. When triggered, each function sends transaction data in an XML string (referred to as Payments, Batch Payments, and Payment Files) to the Financial Engine, which in turn posts the transaction. When the Remittance functions are triggered, the Remittance module obtains all the information needed to process the file, and error messages are posted to the database if discrepancies are found. The Remittance module provides the following functions.

### Post Payment

This function processes individual payments at the account level. This function performs the following steps:

1. The Remittance module inserts a Payment record into the database. Payment records can have both positive and negative Amount values, for payments and refunds respectively. If there is an error, the Remittance module posts an “UNPOSTABLE\_PAYMENT” message that includes the Payment XML string and an appropriate error message, and processing ends.
2. The Remittance module finds the associated account record based on the Account ID. If the payment data includes a currency code, the Remittance module verifies that the Currency code for the account matches that provided in the payment data. If the account cannot be found or the currency codes don't match, the associated account is changed to the “default” account (which is provided in the payment data).
3. The Remittance module checks the Transfer To Account value of the account. If the account has been transferred to another, the Remittance module changes the associated account to the account that it was transferred to and repeats this step. If the transferred to account cannot be found, the associated account is changed to the “default” account (which is provided in the payment data).
4. If the account's Receivable Status value is “UNCOLLECTIBLE”, the associated account is changed to the “default” account (which is provided in the payment data).
5. If an Assistance Program ID is provided, the Application Method must be Receivable Type and no 'Related Transaction' should be attached.
6. If an Assistance Program ID is provided, the Remittance module determines the Receivable Types that the Payment may be applied to, creates 'Related Transactions' with these Receivable Types, and attaches them to the transaction.
7. The Remittance module triggers the Post Transaction function of the Financial Engine, and sets the following attributes for the transaction:
  - Account: Determined from steps 2, 3, and 4.
  - Transaction Type: Payment (PYMNT).
  - Transaction ID: Payment Transaction ID if provided or NULL.
  - RevenueMonth: Payment revenue month if provided or NULL.
  - Note: Payment note if provided or NULL.
  - Amount: Payment amount.
  - BilledOrPaidDate: Payment date if provided or NULL.
  - Invoice ID: Payment invoice ID if provided.
  - Invoice Date: Payment invoice date if provided.
  - Payment: Payment.
  - Application Method: Payment application method.

- Charge or Credit                      Credit
- Defer Balance                          False
- Related Transactions

The Remittance module uses the Payment data for the Post Transaction function.

8. If an Assistance Program ID is provided and the Post Transaction function succeeds, the Remittance module determines which Assistance Pledge (or Pledges) this Payment is associated with and updates the appropriate Pledge's Received Date and Received Amount fields.
9. If the Post Transaction function fails for any reason and the associated account is not the default account, the associated account is changed to the "default" account (which is provided in the payment data) and the Remittance module attempts to trigger the Post Transaction function again.
10. If the associated account is the default account, the Remittance module posts an "UNPOSTABLE\_PAYMENT" message that includes the Payment data and an appropriate error message.
11. If the associated account is not the default account and the account's Receivable Status value is "COLLECTIONS" the Remittance module posts a "COLLECTIONS\_PAYMENT" message that includes the Payment data.

The POST PAYMENT Rules Language statement can be used to post a payment as a single transaction. Some elements of the transaction are internally generated by the rate schedule when the POST PAYMENT Statement executes.

See **Appendix D: Financial Management Rules Language Statements** for details on the use of the POST PAYMENT Statement.

## Process Batch Payment

This function processes batch payments containing one or more individual payments. This function performs the following steps:

1. If the Restart flag is set, the Remittance module checks if the Batch already exists in database. If not, then it clears the Restart flag.
2. If the Received Date is not provided, the Remittance module defaults to current date.
3. If the default Account ID is provided, the Remittance module checks if the account exists. If not the Remittance module posts a "BATHPAYMENT\_ERROR" message that includes the Batch Payment data (with all individual payments) and an appropriate error message, and processing ends.
4. The Remittance module iterates through all the individual payments in the batch and determines the actual total number of payments and the total amount. It then compares these values with the number of payments and amount values (if provided) of the batch payment. If there is a discrepancy, the Remittance module posts a "BATHPAYMENT\_ERROR" message that includes the Batch Payment data (with all individual payments) and an appropriate error message, and processing ends.
5. The Remittance module inserts a Batch Payment record into database, if the Restart flag is not set. If there is an error, it posts a "BATHPAYMENT\_ERROR" message that includes the Batch Payment data (with all individual payments) and an appropriate error message, and processing ends.
6. If the Cancel flag is set to No ('N'), for each Payment in the Batch, if the Restart flag is set to Yes, the Remittance module checks if the Payment already exists in the database. If so, then it skips the payment. If not, the Remittance module triggers the Post Payment function (described above) for the payment..



7. If the Cancel flag is set to Yes ('Y'), for each Payment in the Batch, if the Restart flag is set to Yes, the Remittance module checks if the Payment has already been cancelled. If so, it skips the Payment. If not, the Remittance module triggers the Cancel Payment function for the Payment.
8. In either case, the Remittance module forwards the appropriate attributes to each payment, if not provided, prior to posting, and maintains a count of any payment errors. If the optionally provided max errors limit is surpassed, the Remittance module posts a "BATCHPAYMENT\_ERROR" message that includes the Batch Payment data (with all individual payments) and an appropriate error message, and processing ends.
9. The Remittance module posts a "BATCHPAYMENT\_PROCESSED" message that includes the Batch Payment data (without any individual payments) and the number of payment errors, if any.

### Processing Batch Payments in Cancel mode

The Remittance module can also cancel batch payments. Cancelled batch payments are either:

- **Existing batch payments:** Previously processed batch payments. Processing an existing batch payment in Cancel mode requires either:
  - setting the Cancel and Restart flags of the existing batch payment both set to Yes ('Y') and reprocessing the batch payment, or
  - processing a batch payment that contains the unique ID of the batch payment to be cancelled and has the Cancel and Restart flags both set to Yes ('Y').
- **Batches of individual existing payments:** Batches of payments collected into batches for cancellation. These payments may have been posted either as part of a previous batch payment or as individual payments. Processing a batch of individual existing payments in Cancel mode requires a batch payment that has the Cancel flag set to Yes ('Y'), and the individual payments to be cancelled.

### Process Payment File

This function processes payment files that contain one or more batch payments. This function performs the following steps:

1. If the default account ID is provided, the Remittance module checks if the account exists. If not, it posts a "PAYMENTFILE\_ERROR" message that includes the Payment File data (without any individual batch payments) and an appropriate error message, and processing ends.
2. The Remittance module iterates through all the individual batch payments and determines the actual total number of batch payments and the total amount. It then compares these values with the number of batches and amount values (if provided) of the payment file. If there is a discrepancy, the Remittance module posts a "PAYMENTFILE\_ERROR" message that includes the Payment File data (without any individual batch payments) and an appropriate error message, and processing ends.
3. The Remittance module triggers the Process Batch Payment function (described above) for each individual batch payment in the file. The Remittance module also forwards the appropriate attributes to each batch payment, if not provided, prior to processing. If the Cancel flag is set to Yes ('Y') for the Payment File, the Remittance module sets the Cancel flag to Yes for each Batch prior to processing. If the Restart flag is set for the Payment File, the Remittance module sets the Restart flag for each Batch prior to processing. The Remittance module maintains a count of any batch payment errors during processing, and if the optionally provided Max Errors limit is surpassed, the Remittance module posts a "PAYMENTFILE\_ERROR" message that includes the Payment File data (without any individual batch payments) and an appropriate error message, and processing ends.

4. The Remittance module posts a “PAYMENTFILE\_PROCESSED” message that includes the Payment File data (without any individual batch payments), the number of batch payment errors, if any, and the number of payments in error, if any.

### **Processing Payment Files in Cancel mode**

The Remittance module can also cancel payment files. Cancelled payment files are either:

- **Existing payment files:** Previously processed payment files. Processing an existing batch payment in Cancel mode requires processing a payment file that has the Cancel and Restart flags both set to Yes ('Y'), and contains the unique IDs of the batch payments to be cancelled.
- **Files containing a group of batch payments:** Payment files containing any number of valid batch payments collected for cancellation. These batch payments may have been posted either as part of a previously processed payment file, or may be comprised of individual payments. Processing a payment file containing batch payments in Cancel mode requires a payment file that has the Cancel flag set to Yes ('Y'), and the individual batch payments to be cancelled.

## Using the Process Payment File Command Line Program

The Process Payment file function can be performed using a command line program. The Process Payment file command line program (PROCPMNT.EXE) uses the syntax shown below. Parameter switches are case insensitive (i.e. you can enter them in either upper or lower case (-c or -C)). If a parameter includes a space, you must enclose it in quotes (for example, -s "11/01/1999 12:00:00"). The syntax is:

```
procpmnt.exe -d <connectstring> [-q <qualifier>] [-@ <xmlPaymentFile>]
[-lcfg logging configuration filename]
```

The command must be entered on one line. Also, you must either change to the directory in which the program is stored (typically, \LODESTAR\bin) before entering the command, or specify the path in the command. To view a list of all parameters on-screen, type **procpmnt -?** at the command prompt.

Parameter	Description
-d	<p><i>connectstring</i> is database connection information for the Oracle Utilities Data Repository. This parameter is <b>required</b> and must be in one of the following formats:</p> <p>For Oracle databases:</p> <pre>"Data Source=&lt;data_source&gt;;User ID=&lt;user_id&gt;;Password=&lt;password&gt;;LSProvider=ODP;"</pre> <p>where:</p> <ul style="list-style-type: none"> <li>• &lt;data_source&gt; is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\&lt;machine&gt;\oracle\network\admin directory)</li> <li>• &lt;user_id&gt; is the user ID for the database connection</li> <li>• &lt;password&gt; is the password for the supplied user ID.</li> </ul>
-q	<i>qualifier</i> is the optional qualifier for the data source.
-@	<i>xmlInputFile</i> is an XML file that contains the payment file.
-lcfg	<i>logging configuration filename</i> Name of an optional logging configuration file that specifies where error and log messages are sent. If you omit this parameter, the application creates a log file named PROCPMNT.LOG in the LOG directory.

## Cancel Payment

This function cancels individual payments. Based on the supplied Cancel Reason Code, this function also establishes a penalty installment against the account. This function performs the following steps:

1. The Remittance module finds the specified Payment record based on either a given unique ID or a combination of attributes. If the specified record cannot be found, then the Remittance module posts a “PAYMENT\_EXCEPTION\_ERROR” message that includes the Payment data, as well as an appropriate error message, and processing ends.
2. If an Assistance Program ID is provided, the Remittance module determines the Receivable Types that the Payment may be applied to, creates 'Related Transactions' with these Receivable Types, and attaches them to the transaction.
3. The Remittance module finds the associated Transaction record, triggers the Cancel Transaction function of Financial Engine, and sets the following additional attributes for the transaction:
  - Cancel Revenue Month: As provided or NULL.
  - Cancel Reason Code: As provided or NULL.
  - Cancel Note: As provided or NULL.

The Remittance module uses the Payment data for the Cancel Transaction function.

4. If an Assistance Program ID is provided and the Post Transaction function succeeds, the Remittance module determines which Assistance Pledge (or Pledges) this Payment is associated with and updates the appropriate Pledge's Received Date and Received Amount fields. If an error occurs, the Remittance module posts a “PAYMENT\_EXCEPTION\_ERROR” message that includes the Payment data, as well as an appropriate error message, and processing ends.
5. If the Post Penalty flag is set and a Cancel Reason Code is provided, the Remittance module triggers the Post Penalty function of the Billing module using the Cancel Reason Code as the Penalty Code and the Payment Amount as the Outstanding Amount.

## Process AutoPayment

This function creates an automatic payment and refund for an account, if set up to do so. This function performs the following steps:

1. The Remittance module locates the appropriate Account and determines if an automatic payment plan is currently in effect for the Account via the Auto Payment Plan Table. If not, processing is complete.
2. If the Payment Day value is not NULL, the Remittance module uses it to calculate the Payment Date as follows. If the day of the bill date provided (BilledOrPaidDate of BILL transaction or current time) is less than the Payment Day, advance the bill date's day to the Payment Day. If the day of the bill date provided is greater than the Payment Day, advance the month/year appropriately (if month is less than 12, add 1, if month is 12, add 1 to year and set month to 1) and set the day to the Payment Day.
3. If the Payment Day value is NULL, the Remittance module gets the Auto Payment Delay value from appropriate Payment Method record (zero if NULL), and adds this value (in days) to the bill date provided (BilledOrPaidDate of BILL transaction or current date). The result is the Payment Date value for the new Automatic Payment record (see Step 3 below).
4. The Remittance module creates and inserts a new Automatic Payment record as a child of the Auto Payment Plan record found in Step 1. This record has the following attributes:
  - The Payment Date value was calculated in Step 2.
  - The Transaction Time, unique Transaction ID (optional), and Amount values are provided via the Bill transaction. The Amount value is positive for payments and

negative for refunds. If there is an amount to be refunded, the Remittance module inserts a record into the LSRefund table with appropriate attributes and status set to “SENT”.

- The Batch No. and Cancel Time values should be NULL.
- The Status value should be set to “PENDING”.

## Cancel AutoPayment

If an automatic payment exists for an account, the Cancel AutoPayment function can be used to cancel it. This function performs the following steps:

1. The Remittance module looks for any Automatic Payment records that have either the same transaction unique ID (UIDTRANSACTION) as the provided transaction, or has both an Auto Payment Plan unique ID (UIDAUTOPAYPLAN) and a Transaction Time value based on the Account and Transaction Time of the provided transaction, and whose Batch No. and Cancel Time values are NULL (in other words, the payment has not yet been sent).
2. The Remittance module updates the Cancel Time field for each record found in Step 1 above to the Cancel Time value provided to the Cancel AutoPayment function (or the current time if not provided), and updates the Status field to “CANCELLED”.

## Batch AutoPayments

This function is a scheduled process that groups all pending automatic payments by payment source and forwards them to the appropriate outbound interface.

### Batch AutoPayments Processing

This function performs the following steps:

1. The Remittance module locates all the Automatic Payment records with a Batch No. and Cancel Time value of NULL, and a Payment Date value equal to or before the current date. It then gets each record, along with the attributes from its parent Auto Payment Plan and Payment Method records, and groups these records by Payment Method.
2. For each group (payment method), the Remittance module creates a Payment File data structure (defined above). This data structure consists of one or more Batch Payment data structures (also defined above), whose size (number of payments) should be specified by the Auto Batch Size parameter of the corresponding payment method. The batch number for the first batch starts at 1 in each Payment File and increments for each additional batch in the payment file. The Source Code, Method Code, Number of Batches, Number of Payments, Number of Refunds, Amount, and Refund Amount elements should all be set appropriately. The Financial Engine updates each Auto Payment record found in Step 1 so that its File Name and Batch No. values indicate what batch it is in, and its STATUS value is set to “SENT”. For each refund record found in the Auto Payment table, the Remittance module updates the Status in the LSRefund table with the supplied Refund Status value. If the Refund Status is not supplied, the Remittance module updates the status to “ISSUED”, and updates the Issue Date to the current date.
3. For each Payment File data structure created in Step 2 above, the Remittance module posts both a “PAYMENTFILEOUT” message. In addition, if the Post Delay column is greater than or equal to zero (i.e. not NULL), the Remittance module also posts a “PAYMENTFILEIN” message that includes the Payment File data. The “PAYMENTFILEIN” message should be scheduled to be handled *n* number of days in the future based on the Post Delay parameter of the corresponding payment method.
4. For each Payment File data structure created in Step 2 above, the Remittance module posts an “AUTOPAYMENTS\_PROCESSED” message that include the Payment File data (not including the individual batch payments).

## Using the Batch AutoPayments Command Line Program

The Batch AutoPayments function can be performed using a command line program. The Batch AutoPayments command line program (BAUTOPAY.EXE) uses the syntax shown below. Parameter switches are case insensitive (i.e. you can enter them in either upper or lower case (-c or -C)). If a parameter includes a space, you must enclose it in quotes (for example, -s "11/01/1999 12:00:00").

```
bautopay.exe -d <connectstring> [-q <qualifier>] [-s <refundStatus>]
[-lcfg logging configuration filename]
```

The command must be entered on one line. Also, you must either change to the directory in which the program is stored (typically, \LODESTAR\bin) before entering the command, or specify the path in the command. To view a list of all parameters on-screen, type **bautopay -?** at the command prompt.

Parameter	Description
-d	<p><i>connectstring</i> is database connection information for the Oracle Utilities Data Repository. This parameter is <b>required</b> and must be in one of the following formats:</p> <p>For Oracle databases:</p> <pre>"Data Source=&lt;data_source&gt;;User ID=&lt;user_id&gt;;Password=&lt;password&gt;;LSProvider=ODP;"</pre> <p>where:</p> <ul style="list-style-type: none"> <li>• &lt;data_source&gt; is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\&lt;machine&gt;\oracle\network\admin directory)</li> <li>• &lt;user_id&gt; is the user ID for the database connection</li> <li>• &lt;password&gt; is the password for the supplied user ID.</li> </ul>
-q	<i>qualifier</i> is the optional qualifier for the data source.
-s	<i>refundStatus</i> is an optional Status for refunds processed in the batch autopayment.
-lcfg	<i>logging configuration filename</i> Name of an optional logging configuration file that specifies where error and log messages are sent. If you omit this parameter, the application creates a log file named BAUTOPAY.LOG in the LOG directory.

# Chapter 8

---

## Maintenance

This chapter describes the Maintenance functions of the Oracle Utilities Receivables Component, including an overview of the data and processing performed by the Maintenance module, and explanations of each of the Maintenance functions performed, including:

- Post Payment Transfer,
- Cancel Payment Transfer,
- Post Adjustment,
- Cancel Adjustment,
- Post Refund,
- Cancel Refund,
- Post Pending Refunds,
- Process Pending Refunds,
- Process Time Voided Refunds,
- Update Refund Status
- Post Write-Off,
- Cancel Write-Off,
- Write Off Account,
- Write On Account,
- Cancel Transactions,
- Post Balance Transfer,
- Cancel Balance Transfer,
- Transfer Account Balance, and
- Return Account Balance.

## Maintenance Functions

The core function of the Maintenance module is to send maintenance transactions, such as transfers, adjustments, refunds, and write-offs to the Financial Engine. These transactions are sent to the Maintenance module from either the Oracle Utilities Receivables Component user interface or external systems through an interface.

## Maintenance Data

The functions of the Maintenance module require data source and transaction data that contains the information necessary to process the specific function being performed (i.e. post a payment transfer, cancel an adjustment, etc.). When the functions are triggered through an interface, the transaction data is sent in an XML string to the Maintenance module and then to the Financial Engine. When the functions are triggered through the Oracle Utilities Receivables Component user interface, the transaction data is generated by the user interface based on data entered by the user, and then is sent to the Maintenance module.

### Batch Refunds

Several of the refund functions use batch refund data. A batch refund in the Maintenance module includes the following:

- Number of refunds (the total number of refunds processed in the file. This is an output attribute),
- Total Amount (the total amount of all refunds processed in the file, including the currency code for the total amount. This is an output attribute),
- Number of Errors (the total number of errors occurred while refunds processing. This is an output attribute).
- Refund Wait Days (the number of days an unapplied outstanding credit has to remain on a closed account. This is a required input attribute for the Post Pending Refunds function),
- Refund Reason Code (the reason code for which the refunds are being posted to the LSRefund table. This is an optional input attribute for the Post Pending Refunds function),
- File Name (a user-specified fully qualified file name in which the batch refund data is stored. This is a required input attribute for Process Pending Refunds function),
- Account Number (the financial institution's account no. This is a required input attribute for Process Pending Refunds function),
- Void Offset Days (the number of days after which an issued check would become void. This is a required input attribute for Process Time Voided Refunds function),
- Writeoff Reason Code (the reason code for writing off the account. This is a required input attribute for Process Time Voided Refunds function),
- Country (the optional country name for the address. The default is USA),
- Account ID (from the Account table),
- Unique ID of the refund (from the LSRefund table),
- Unique ID for the refund transaction (from the LSTransaction table),
- Unique ID of the account (from the Account table),
- Unique ID for checking account (from the Checking Account table)
- Amount (the amount being refunded, including the currency code for the amount. This is a required input field when updating the status),
- Check Number (the check number in the checking account),



- Issued Date (the date on which the check was issued),
- Cashed Date (the date on which the check was cashed),
- Reason (description of the refund reason),
- Status (the status of the refund. The allowed states are ISSUED and CASHED. This is a required input when updating the status),
- Customer Name (the name of the customer to whom the amount is being refunded),
- Address1, Address2, Address 3 (the house number, apartment number, and street name of the customer),
- City (the name of the town where the customer lives),
- State (the state in which the city is located), and
- County (the county in which the city is located).

## Maintenance Function Processing

Each of the Maintenance module functions can be triggered either from the Oracle Utilities Receivables Component user interface (see the *Oracle Utilities Billing Component User's Guide*) or from an external system via an interface. When triggered, each function sends transaction data in an XML string to the Financial Engine, which posts the transaction. When the Maintenance functions are triggered, the Maintenance module obtains all the information needed to process the file, and error messages are posted to the database if discrepancies are found. The Maintenance module provides the following functions.

### Post Payment Transfer

This function transfers individual payments between accounts. This function performs the following steps:

1. The Maintenance module sets the transaction's Transaction Type to PYMNT (Payment).
2. If not explicitly specified, the Maintenance module sets the application method to Immediate (IMMEDIATE).
3. The Maintenance module sets CHARGEORCREDIT (Charge or Credit) to Credit (CR).
4. The Maintenance module triggers the Transfer Transaction Amount function of the Financial Engine and uses the data source and transaction data (outlined above) when processing the transaction.

### Cancel Payment Transfer

This function cancels a previous payment transfer. When triggered, this function performs the following steps:

1. The Maintenance module sets the transaction's Transaction Type to PYMNT (Payment).
2. The Maintenance module triggers the Cancel Transaction function of the Financial Engine, and uses the data source and transaction data (outlined above) when processing the transaction.

### Post Adjustment

This function posts an adjustment transaction credit or charge against an account. This function performs the following steps:

1. The Maintenance module sets the transaction's Transaction Type to ADJ (Adjustment).
2. If a Related Transaction is provided (it is an error if more than one is provided), the Maintenance module uses the Amount (if not specifically provided for the transaction), Charge or Credit (the opposite), Receivable Type, Charge Type, Operating Company and

Jurisdiction (if not specifically provided for the transaction), the unique ID of the associated Service Plan, and the unique ID of the associated Budget Plan from the related transaction for the adjustment transaction.

3. If both the Amount and a Related Transaction are specified, then the Maintenance module verifies that the amount is not greater than the Related Transaction amount. If so, it is an error.
4. If a Related Transaction is provided, the Maintenance module sets the application method to Specified (SPECIFIED). Otherwise, if not explicitly specified and the adjustment is a credit, the Maintenance module sets the application method to Immediate (IMMEDIATE).
5. The Maintenance module then triggers the Post Transaction function of the Financial Engine, using the data source and transaction data (with the additions as described above).
6. If a Related Transaction is provided and it is a taxed transaction, the Maintenance module posts the appropriate tax adjustment by triggering the Post Tax function of the Billing module for each associated tax, relating the tax to this transaction.

The POST ADJUSTMENT Rules Language statement can be used to post a charge or credit adjustment as a single transaction. Some elements of the transaction are internally generated by the rate schedule when the POST ADJUSTMENT Statement executes.

See **Appendix D: Financial Management Rules Language Statements** for details on the use of the POST ADJUSTMENT Statement.

## Cancel Adjustment

This function cancels a previous adjustment. This function performs the following steps:

1. The Maintenance module sets the transaction's Transaction Type to ADJ (Adjustment).
2. The Maintenance module triggers the Cancel Transaction function of the Financial Engine, and uses the data source and transaction data (outlined above) when processing the transaction.

## Post Refund

This function posts a refund transaction charge against an account. This function performs the following steps:

1. The Maintenance module sets the transaction's Transaction Type to RFND (Refund).
2. The Maintenance module sets the transaction's Charge or Credit value to Charge (CH).
3. If any Related Transactions are provided, the Maintenance module verifies that they are all credits. If not, it is an error.
4. If the Amount is not provided and Related Transactions are provided, then the Maintenance module uses the sum of the Balance values of the Related Transactions as the amount of the transaction.
5. If both the Amount and Related Transactions are specified, then the Maintenance module verifies that the amount is not greater than the sum of the Balance values of the Related Transactions. If so, it is an error.
6. If the Amount is specified, the Maintenance module verifies that the currency code for the amount matches the Currency code for the account. If not, it is an error.
7. If the Amount is provided without any Related Transactions, then the Maintenance module verifies that the Amount is not greater than the account's current credit balance. If so, it is an error.
8. The Maintenance module sets the transaction's Billed or Paid Date value to NULL.

9. If a Related Transaction is provided, the Maintenance module sets the application method to Specified (SPECIFIED). Otherwise, if not explicitly specified, the Maintenance module sets the application method to Immediate (IMMEDIATE).
10. The Maintenance module triggers the Post Transaction function of the Financial Engine using the data source and transaction data (with the additions as described above).
11. If the account is on an AutoPayPlan and the Use For Refund flag in the AutoPayment Plan table is set to Yes ('Y'), the Maintenance module triggers the Process Auto Payment function of the Remittance module, with a negative Amount value, and creates a Refund record with the Status set to SENT.
12. Otherwise, the Maintenance module creates a Refund record with the Status set to PENDING.
13. The Maintenance module inserts the above Refund record into the LSRefund table with the following details:
  - UIDTRANSACTION = Transaction UID
  - UIDACCOUNT = Account UID
  - AMOUNT = Transaction Amount
  - UIDCHECKINGACCOUNT = Checking Account UID
  - ISSUEDATE, CASHEDDATE, VOIDDATE = NULL

The POST REFUND Rules Language statement can be used to post a refund as a single transaction. Some elements of the transaction are internally generated by the rate schedule when the POST REFUND Statement executes.

See **Appendix D: Financial Management Rules Language Statements** for details on the use of the POST REFUND Statement.

## Cancel Refund

This function cancels a previous refund. This function performs the following steps:

1. The Maintenance module checks if the Refund has been cashed already (ISSUED status for auto refunds and CASHED status for refunds via check). If it has been cashed, it is an error.
2. The Maintenance module sets the transaction's Transaction Type to RFND (Refund).
3. If the account is on an AutoPayment Plan, the Maintenance module triggers the Cancel AutoPayment function of the Remittance module.
4. The Maintenance module triggers the Cancel Transaction function of the Financial Engine and uses the data source and transaction data (outlined above) when processing the transaction.
5. If the Cancel Transaction succeeds, the Maintenance module updates the status in the LSRefund table to VOIDED, and updates the Void Date with the current date.

## Post Pending Refunds

This function scans the Oracle Utilities Data Repository for all closed accounts with an unapplied credit, determines if the credit has been active on the account for the specified time period, and creates a Refund record in the database. This function performs the following steps:

1. The Refund Wait Days attribute must be provided.
2. The Maintenance module determines if the account is closed, by looking for records in the Account table with Stop Time less than the current date.
3. The Maintenance module check the Allow Refund flag for the account in the Account FME table. If this flag is set to No ('N'), the Maintenance module skip this account.

4. The Maintenance module determines if the account has an unapplied credit by checking that the total of the Last Balance and New Activity is less than zero.
5. The Maintenance module determines if the credit balance has remained on the account for the specified period by checking that the current date minus the Stop Time is greater than the Refund Wait Days attribute.
6. The Maintenance module determines if the account is not on an assistance plan by checking the Assistance Plan table.
7. The Maintenance module verifies that all other accounts for the customer don't have any outstanding balances by checking that the total of the Last Balance and New Activity is less than or equal to zero for the remaining accounts of a customer.  
  
If this is not the case (if there is an outstanding balance on any of the account for the customer), the Maintenance module sends a REFUND\_OUTSTANDING\_BALANCE\_ERROR message to the appropriate work queue.
8. The Maintenance module triggers the Post Refund function (p. 8-4) with the following attributes
  - Transaction Account UIDACCOUNT = Account UID
  - Transaction AMOUNT = positive value of Last Balance plus New Activity.

## Using the Post Pending Refunds Command Line Program

The Post Pending Refunds function can be performed using a command line program using the syntax shown below. Parameter switches are case insensitive (i.e. you can enter them in either upper or lower case (-c or -C)). If a parameter includes a space, you must enclose it in quotes (for example, -s "11/01/1999 12:00:00"). The syntax is:

```
pendrfd.exe -d <connectstring> [-q <qualifier>] -@ <xmlBatchRefundFilename>
[-lcfg <logging configuration filename>]
```

The command must be entered on one line. Also, you must either change to the directory in which the program is stored (typically, \LODESTAR\bin) before entering the command, or specify the path in the command. To view a list of all parameters on-screen, type **pendrfd -?** at the command prompt.

Parameter	Description
-d	<p><i>connectstring</i> is database connection information for the Oracle Utilities Data Repository. This parameter is <b>required</b> and must be in one of the following formats:</p> <p>For Oracle databases:</p> <pre>"Data Source=&lt;data_source&gt;;User ID=&lt;user_id&gt;;Password=&lt;password&gt;;LSProvider=ODP;"</pre> <p>where:</p> <ul style="list-style-type: none"> <li>• &lt;data_source&gt; is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\&lt;machine&gt;\oracle\network\admin directory)</li> <li>• &lt;user_id&gt; is the user ID for the database connection</li> <li>• &lt;password&gt; is the password for the supplied user ID.</li> </ul>
-q	<i>qualifier</i> is the optional qualifier for the data source.
-@	<i>xmlBatchRefundFilename</i> is the name of the Batch Refund (see <b>Batch Refunds</b> , p. 8-2) file to be used for processing.
-lcfg	<i>logging configuration filename</i> Name of an optional logging configuration file that specifies where error and log messages are sent. If you omit this parameter, the application creates a log file named PENDRFND.LOG in the LOG directory.

## Process Pending Refunds

This function scans the Oracle Utilities Data Repository for all Refund records with a Status of PENDING, converts them into the Batch Refund data type (see **Batch Refunds**, p. 8-2), and stores them in a user-defined file to be used for issuing the refunds via checks.

A Refund record's Status should be one of the following at any time:

- PENDING - Refund details have not yet been sent to the Check Writing System. This is the initial state of refunds issued via checks in the LSRefund table.
- SENT - Refund details have been sent to the Check Writing system for writing a check (for refunds issued via checks). The Refund record was created both in the Automatic Payment and Refund tables. This is the initial state of refunds issued via auto payments.
- ISSUED - Check has been issued (in case of refunds issued via checks), and refund details have been sent to credit card and debit card companies.
- CASHED - Check has been cashed by the customer.
- VOIDED - Check has been either cancelled or not cashed before the void date (Expired).

This function performs the following steps:

1. The Maintenance module checks whether the required Account Number and Filename elements (see **Batch Refunds**, p. 8-2) have been provided, If not, processing is done.
2. The Maintenance module gets all the pending refund accounts with a status of PENDING.
3. The Maintenance module fills in the following data elements with their corresponding database fields:
  - Account ID
  - Unique ID of Associated Refund,
  - Unique ID of Transaction,
  - Amount,
  - Check Number (incremented by 1),
  - Unique ID of associated checking account,
  - RTN,
  - Institution (obtained from the database based on the Account No.)
4. The Maintenance module gets the customer information along with the address and fills in the respective elements in the Batch Refund XML file.
5. The Maintenance module fill in the Number of Refunds and Amount with the total number of refunds and total amount.
6. The Maintenance module saves the converted Refunds data into the provided file name.
7. The Maintenance module updates the Status in the LSRefund table to SENT and updates the Send Date with current time.
8. The Maintenance module updates the unique ID of the associated checking account, and Check Number in the LSRefund table, and updates the Last Check Number in the Checking Account table.

## Using the Process Pending Refunds Command Line Program

The Process Pending Refunds function can be performed using a command line program using the syntax shown below. Parameter switches are case insensitive (i.e. you can enter them in either upper or lower case (-c or -C)). If a parameter includes a space, you must enclose it in quotes (for example, -s "11/01/1999 12:00:00"). The syntax is:

```
procrfnd.exe -d <connectstring> [-q <qualifier>]-@ <xmlBatchRefundFilename>
[-lcfg <logging configuration filename>]
```

The command must be entered on one line. Also, you must either change to the directory in which the program is stored (typically, \LODESTAR\bin) before entering the command, or specify the path in the command. To view a list of all parameters on-screen, type **procrfnd -?** at the command prompt.

Parameter	Description
-d	<p><i>connectstring</i> is database connection information for the Oracle Utilities Data Repository. This parameter is <b>required</b> and must be in one of the following formats:</p> <p>For Oracle databases:</p> <pre>"Data Source=&lt;data_source&gt;;User ID=&lt;user_id&gt;;Password=&lt;password&gt;;LSProvider=ODP;"</pre> <p>where:</p> <ul style="list-style-type: none"> <li>• &lt;data_source&gt; is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\&lt;machine&gt;\oracle\network\admin directory)</li> <li>• &lt;user_id&gt; is the user ID for the database connection</li> <li>• &lt;password&gt; is the password for the supplied user ID.</li> </ul>
-q	<i>qualifier</i> is the optional qualifier for the data source.
-@	<i>xmlBatchRefundFilename</i> is the name of the Batch Refund (see <b>Batch Refunds</b> , p. 8-2) file to be used for processing.
-lcfg	<i>logging configuration filename</i> Name of an optional logging configuration file that specifies where error and log messages are sent. If you omit this parameter, the application creates a log file named PROCREFND.LOG in the LOG directory.

## Process Time Voided Refunds

This function scans the Oracle Utilities Data Repository for all Refund records with a Status of ISSUED and determines whether the checks have been cashed before the void date. This function performs the following steps:

1. The Maintenance module determines whether the required Void Offset Days, Writeoff Reason Code elements (see **Batch Refunds**, p. 8-2) have been provided. If not, processing is done.
2. The Maintenance module get all the Refund records with a status of SENT or ISSUED and whose Check Number is not null.
3. The Maintenance module checks that the current date minus the Issued Date or Send Date Today's date is greater than the Void Offset Days.
4. The Maintenance module sets Write Off Reason Code for the account with the provided Write Off Reason Code.
5. The Maintenance module triggers the Cancel Refund function (p. 8-5) of the Maintenance module.
6. If the account is closed, the Maintenance module triggers the Write Off Account function (p. 8-14) of the Maintenance module



## Using the Process Time Voided Refunds Command Line Program

The Process Time Voided Refunds function can be performed using a command line program using the syntax shown below. Parameter switches are case insensitive (i.e. you can enter them in either upper or lower case (-c or -C)). If a parameter includes a space, you must enclose it in quotes (for example, -s "11/01/1999 12:00:00"). The syntax is:

```
voidrfnd.exe -d <connectstring> [-q <qualifier>] -@ <xmlBatchRefundFilename>
[-lcfg <logging configuration filename>]
```

The command must be entered on one line. Also, you must either change to the directory in which the program is stored (typically, \LODESTAR\bin) before entering the command, or specify the path in the command. To view a list of all parameters on-screen, type **voidrfnd -?** at the command prompt.

Parameter	Description
-d	<p><i>connectstring</i> is database connection information for the Oracle Utilities Data Repository. This parameter is <b>required</b> and must be in one of the following formats:</p> <p>For Oracle databases:</p> <pre>"Data Source=&lt;data_source&gt;;User ID=&lt;user_id&gt;;Password=&lt;password&gt;;LSPProvider=ODP;"</pre> <p>where:</p> <ul style="list-style-type: none"> <li>• &lt;data_source&gt; is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\&lt;machine&gt;\oracle\network\admin directory)</li> <li>• &lt;user_id&gt; is the user ID for the database connection</li> <li>• &lt;password&gt; is the password for the supplied user ID.</li> </ul>
-q	<i>qualifier</i> is the optional qualifier for the data source.
-@	<i>xmlBatchRefundFilename</i> is the name of the Batch Refund (see <b>Batch Refunds</b> , p. 8-2) file to be used for processing.
-lcfg	<i>logging configuration filename</i> Name of an optional logging configuration file that specifies where error and log messages are sent. If you omit this parameter, the application creates a log file named VOIDRFND.LOG in the LOG directory.

## Update Refund Status

This function updates records in the Oracle Utilities Data Repository with refund status information. This function performs the following steps:

1. The Maintenance module gets the account status details from the provided Batch Refund data. Either the unique ID of the refund, the unique ID of the transaction, or the unique ID of the associated checking account, and the Account Number and Check Number must be provided. The Maintenance module checks them against the values stored in the database.
2. The Maintenance module checks the amounts. If they don't match, the Maintenance module writes a REFUND\_UPDATE\_ERROR error message to the work queue.
3. The Maintenance module checks the Status. A Refund's Status can change from one state to another as follows: SENT to ISSUED, ISSUED to CASHED, SENT to CASHED. The system does not update a refund with the same state. If there is an error, the Maintenance module sends an error message to the work queue.

A Refund's Issue Date should not be less than its Send Date. A Refund's Cashed Date should not be less than either the Send Date or the Issued Date based on the status. If this doesn't match, the Maintenance module sends a REFUND\_UPDATE\_ERROR error message to the work queue. If the Issued Date or Cashed Date are not provided, the Maintenance module updates the record with the current date.

4. The Maintenance module updates the LSRefund table with appropriate valid values.
5. If the Maximum No. Errors attribute is provided, the Maintenance module checks if the error count exceeds the Maximum No. Errors. If so, the Maintenance modules sends a BATCHREFUND\_UPDATE\_ERROR error message that includes the Batch Refund xml string.
6. The Maintenance module posts a BATCHREFUND\_UPDATE\_PROCESSED message that includes Batch Refund xml string and the number of errors, if any.

## Using the Update Refund Status Command Line Program

The Update Refund Status function can be performed using a command line program using the syntax shown below. Parameter switches are case insensitive (i.e. you can enter them in either upper or lower case (-c or -C)). If a parameter includes a space, you must enclose it in quotes (for example, -s "11/01/1999 12:00:00"). The syntax is:

```
updtrfnd.exe -d <connectstring> [-q <qualifier>] -@ <xmlBatchRefundFilename>
[-lcfg <logging configuration filename>]
```

The command must be entered on one line. Also, you must either change to the directory in which the program is stored (typically, \LODESTAR\bin) before entering the command, or specify the path in the command. To view a list of all parameters on-screen, type **updtrfnd -?** at the command prompt.

Parameter	Description
-d	<p><i>connectstring</i> is database connection information for the Oracle Utilities Data Repository. This parameter is <b>required</b> and must be in one of the following formats:</p> <p>For Oracle databases:</p> <pre>"Data Source=&lt;data_source&gt;;User ID=&lt;user_id&gt;;Password=&lt;password&gt;;LSProvider=ODP;"</pre> <p>where:</p> <ul style="list-style-type: none"> <li>• &lt;data_source&gt; is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\&lt;machine&gt;\oracle\network\admin directory)</li> <li>• &lt;user_id&gt; is the user ID for the database connection</li> <li>• &lt;password&gt; is the password for the supplied user ID.</li> </ul>
-q	<i>qualifier</i> is the optional qualifier for the data source.
-@	<i>xmlBatchRefundFilename</i> is the name of the Batch Refund (see <b>Batch Refunds</b> , p. 8-2) file to be used for processing.
-lcfg	<i>logging configuration filename</i> Name of an optional logging configuration file that specifies where error and log messages are sent. If you omit this parameter, the application creates a log file named UPDTRFND.LOG in the LOG directory.

## Post Write-Off

This function posts write-off transaction credits or charges against an account. This function performs the following steps:

1. The Maintenance module sets the transaction's Transaction Type to WRTOFF (Write-Off).
2. The Maintenance module verifies that the single related transaction is provided and that its balance is greater than zero. If not it is an error.
3. The Maintenance module sets the CHARGEORCREDIT value to opposite that of related transaction.
4. The Maintenance module sets the AMOUNT equal to the BALANCE of related transaction.
5. The Maintenance module sets the application method to Specified (SPECIFIED).
6. The Maintenance module sets the DEFERBALANCE (Defer Balance) to false.
7. The Maintenance module triggers the Post Transaction function of the Financial Engine using the data source and transaction data (with the additions as described above).

## Cancel Write-Off

This function cancels a previous write-off transaction. This function performs the following steps:

1. The Maintenance module sets the transaction's Transaction Type to WRTOFF (Write-Off).
2. The Maintenance module triggers the Cancel Transaction function of the Financial Engine and uses the data source and transaction data (outlined above) when processing the transaction.

## Write Off Account

This function posts a write-off transaction for each outstanding (balance > 0) transaction that has been previously posted for the account. This function performs the following steps:

1. The Maintenance module verifies that the current account balance is not equal to zero. If so it is an error.
2. The Maintenance module triggers the Post Write Off function (p. 8-14) for each outstanding transaction (balance > 0) for the account.
3. If the account's current balance is a net charge, the Maintenance module updates the account's RECEIVABLESTATUS to "UNCOLLECTIBLE". If a net credit the Maintenance module updates the account's RECEIVABLESTATUS to "UNREFUNDABLE".
4. If a WRITEOFFREASONCODE was provided, the Maintenance module updates the account accordingly.

The POST WRITEROFF Rules Language statement can be used to write off an account. Some elements of the transaction are internally generated by the rate schedule when the POST WRITEROFF Statement executes.

See **Appendix D: Financial Management Rules Language Statements** for details on the use of the POST WRITEROFF Statement.

## Write On Account

This function cancels all previous write-off transactions for the account. This function performs the following steps:

1. The Maintenance module triggers the Cancel Write Off function (p. 8-14) each previously posted Write-Off transaction for the account.
2. If a RECEIVABLESTATUS is provided, the Maintenance module updates the account accordingly.

3. The Maintenance module updates the account's WRITEOFFREASONCODE to NULL.

### Post Transaction

This function posts a transaction of any type. This function performs the following steps:

1. The Maintenance module determines the transaction type of the transaction.
2. The Maintenance module triggers the appropriate post method based on the transaction type determined in step 1 above.

### Cancel Transaction

This function cancels a transaction of any type. This function performs the following steps:

1. The Maintenance module determines the transaction type of the transaction.
2. The Maintenance module triggers the appropriate Cancel function based on the transaction type determined in Step 1 above.

### Process Batch Transaction

This function processes batch transactions containing one or more individual transactions. This function performs the following steps:

1. If the Restart flag is set, the Maintenance module checks if the Batch already exists in database. If not, then it clears the Restart flag.
2. If the Received Date is not provided, the Maintenance module defaults to current date.
3. The Maintenance module iterates through all the individual transactions in the batch and determines the actual total number of transactions and the total amount. It then compares these values with the number of transactions and amount values (if provided) of the batch transaction. If there is a discrepancy, the Maintenance module posts a "BATCHTRANSACTION\_ERROR" message that includes the Batch Transaction data (with all individual transactions) and an appropriate error message, and processing ends.
4. The Maintenance module inserts a Batch Transaction record into database, if the Restart flag is not set. If there is an error, it posts a "BATCHTRANSACTION\_ERROR" message that includes the Batch Transaction data (with all individual payments) and an appropriate error message, and processing ends.
5. If the Cancel flag is set to No ('N'), the Maintenance module triggers the Post Transaction function (described above) for each transaction.
6. If the Cancel flag is set to Yes ('Y'), for each Transaction in the Batch, if the Restart flag is set to Yes, the Maintenance module checks if the transaction has already been cancelled. If so, it skips the transaction. If not, the Maintenance module triggers the Cancel Transaction function for the transaction.
7. In either case, the Maintenance module forwards the appropriate attributes to each transaction, if not provided, prior to posting, and maintains a count of any transaction errors. If the optionally provided Max Errors Limit is surpassed, the Maintenance module posts a "BATCHTRANSACTION\_ERROR" message that includes the Batch Transaction data (with all individual transactions) and an appropriate error message, and processing ends.
8. The Maintenance module posts a "BATCHTRANSACTION\_PROCESSED" message that includes the Batch Transaction data (without any individual transactions) and the number of transaction errors, if any.

### Processing Batch Transactions in Cancel mode

The Maintenance module can also cancel batch transactions. Cancelled batch transactions are either:

- **Existing batch transactions:** Previously processed batch transactions. Processing an existing batch transaction in Cancel mode requires either:
  - setting the Cancel and Restart flags of the existing batch transaction both set to Yes (‘Y’) and reprocessing the batch transaction, or
  - processing a batch transaction that contains the unique ID of the batch transaction to be cancelled and has the Cancel and Restart flags both set to Yes (‘Y’).
- **Batches of individual existing transactions:** Batches of transactions collected into batches for cancellation. These transactions may have been posted either as part of a previous batch transaction or as individual transactions. Processing a batch of individual existing transactions in Cancel mode requires a batch transaction that has the Cancel flag set to Yes (‘Y’), and the individual transactions to be cancelled.

### Process Transaction File

This function processes transaction files containing one or more batch transactions. This function performs the following steps:

1. The Maintenance module iterates through all the individual batch transactions and determines the actual total number of batch transactions and the total amount. It then compares these values with the number of batches and amount values (if provided) of the transaction file. If there is a discrepancy, the Maintenance module posts a “TRANSACTIONFILE\_ERROR” message that includes the Transaction File data (without any individual batch transactions) and an appropriate error message, and processing ends.
2. The Maintenance module triggers the Process Batch Transaction function (described above) for each individual batch transaction in the file. The Maintenance module also forwards the appropriate attributes to each batch transaction, if not provided, prior to processing. If the Cancel flag is set to Yes (‘Y’) for the Transaction File, the Maintenance module sets the Cancel flag to Yes for each Batch prior to processing. If the Restart flag is set for the Transaction File, the Maintenance module sets the Restart flag for each Batch prior to processing. The Maintenance module maintains a count of any batch transaction errors during processing, and if the optionally provided Max Errors limit is surpassed, the Maintenance module posts a “TRANSACTIONFILE\_ERROR” message that includes the Transaction File data (without any individual batch transactions) and an appropriate error message, and processing ends.
3. The Maintenance module posts a “TRANSACTIONFILE\_PROCESSED” message that includes the Transaction File data (without any individual batch transactions), the number of batch transaction errors, if any, and the number of transactions in error, if any.

### Processing Transaction Files in Cancel mode

The Maintenance module can also cancel transaction files. Cancelled transaction files are either:

- **Existing transaction files:** Previously processed transaction files. Processing an existing batch transaction in Cancel mode requires processing a transaction file that has the Cancel and Restart flags both set to Yes (‘Y’), and contains the unique IDs of the batch transactions to be cancelled.
- **Files containing a group of batch transactions:** Transaction files containing any number of valid batch transactions collected for cancellation. These batch transactions may have been posted either as part of a previously processed transaction file, or may be comprised of individual transactions. Processing a transaction file containing batch transactions in Cancel mode requires a transaction file that has the Cancel flag set to Yes (‘Y’), and the individual batch transactions to be cancelled.

## Using the Process Transaction File Command Line Program

The Process Transaction File function can be performed using a command line program using the syntax shown below. Parameter switches are case insensitive (i.e. you can enter them in either upper or lower case (-c or -C)). If a parameter includes a space, you must enclose it in quotes (for example, -s "11/01/1999 12:00:00"). The syntax is:

```
proctrns.exe -d <connectstring> [-q <qualifier>] -@ <xmlTransactionFile>
[-lcfg <logging configuration filename>]
```

The command must be entered on one line. Also, you must either change to the directory in which the program is stored (typically, \LODESTAR\bin) before entering the command, or specify the path in the command. To view a list of all parameters on-screen, type **proctrns -?** at the command prompt.

Parameter	Description
-d	<p><i>connectstring</i> is database connection information for the Oracle Utilities Data Repository. This parameter is <b>required</b> and must be in one of the following formats:</p> <p>For Oracle databases:</p> <pre>"Data Source=&lt;data_source&gt;;User ID=&lt;user_id&gt;;Password=&lt;password&gt;;LSProvider=ODP;"</pre> <p>where:</p> <ul style="list-style-type: none"> <li>• &lt;data_source&gt; is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\&lt;machine&gt;\oracle\network\admin directory)</li> <li>• &lt;user_id&gt; is the user ID for the database connection</li> <li>• &lt;password&gt; is the password for the supplied user ID.</li> </ul>
-q	<i>qualifier</i> is the optional qualifier for the data source.
-@	<i>xmlTransactionFile</i> is the name of the transaction file (see <b>Transaction Files</b> , p. 5-7) file to be used for processing.
-lcfg	<i>logging configuration filename</i> Name of an optional logging configuration file that specifies where error and log messages are sent. If you omit this parameter, the application creates a log file named PROCTRNS.LOG in the LOG directory.

## Post Balance Transfer

This function transfers the balance of a single previously posted transaction from one account to another. This function performs the following steps:

1. The Maintenance module verifies that the related transaction has an outstanding balance and has not been cancelled. If not it is an error.
2. The Maintenance module initializes the “from” transaction as follows:
  - a. Set the TRANSACTIONTYPE to BALTXFR (Balance Transfer).
  - b. Set the Account equal to the related transaction account.
  - c. Set CHARGEORCREDIT to opposite of related transaction.
  - d. Set the Amount to the balance of the related transaction.
  - e. Set DEFERBALANCE to false.
  - f. Set related transaction to related transaction.
  - g. Set remaining attributes equal to corresponding attributes of input transaction.
3. The Maintenance module triggers the Post Transaction function of the Financial Engine using the “from” transaction above.
4. Initialize the “to” transaction as follows:
  - a. Set TRANSACTIONTYPE to BALTXFR (Balance Transfer).
  - b. Set CHARGEORCREDIT equal to that of related transaction.
  - c. Set the Amount to the balance of the related transaction
  - d. Set DEFERBALANCE to false.
  - e. Set BILLEDORPAIDDATE and DUE DATE equal to that of related transaction if not explicitly provided.
  - f. Set the UIDTXFRFROMTRANS to UID of “from” transaction posted above.
5. The Maintenance module triggers the Post Transaction function of the Financial Engine using the “to” transaction above.
6. The Maintenance module updates the UIDTXFRTOTRANS value of the “from” transaction to the UID of the “to” transaction posted above.



**Cancel Balance Transfer**

This function cancels a previous balance transfer transaction. This function performs the following steps:

1. The Maintenance module sets the TRANSACTIONTYPE to BALTXFR (Balance Transfer).
2. The Maintenance module triggers the Cancel Transaction function of the Financial Engine and uses the data source and transaction data (outlined above) when processing the transaction.
3. The Maintenance module finds the other (“to” or “from”) transaction and repeat steps 1 and 2 for it.

**Transfer Account Balance**

This function posts a balance transfer transaction for each outstanding transaction (balance > 0) that has been previously posted for the account. This function triggers the Post Balance Transfer function (p.8-17) and uses the data source and transaction data (outlined above) when processing the transaction.

**Return Account Balance**

This function cancels all previous balance transfer transactions for the account. This function triggers the Cancel Balance Transfer function (p.8-19) and uses the data source and transaction data (outlined above) when processing the transaction.



# Chapter 9

---

## Collections

This chapter describes the Collections functions of the Oracle Utilities Receivables Component, including:

- **Collections Database Tables**
- **Collections Arrangements**
- **Collection Exemptions**
- **Collections Processing and Activities**

## Collections Database Tables

In addition to the tables described in **Chapter 3: The Oracle Utilities Receivables Component Database**, the Collections module also uses a set of tables specifically designed to store data related to collection agencies and arrangements. These tables are described below.

### Collections Agency Tables

Collection agency tables store data associated with specific collection agencies. Collection agencies are agencies employed to collect outstanding payments from customers.

#### Collection Agency Table

Records in this table represent individual collection agencies in the system.

#### Collection Agency Program Table

Records in this table represent individual collection programs (each provided by a collection agency) in the system.

#### Collection Agency X Directory Table

Records in this table associate collection agencies with contact information (from the Directory table) in the system.

#### Collection Program Type Table

Records in this table represent different types of collection programs in the system.

#### Collection Program X Account Table

Records in this table associate collection programs with individual accounts in the system.

### Collection Arrangements Tables

Collection arrangement tables store data associated with specific collection arrangements. Collection arrangements are arrangements made with a customer regarding collection of outstanding payments.

#### Collection Arrangement Type Table

Records in this table represent specific types of collection arrangements.

#### Collection Arrangement Table

Records in this table represent specific collection arrangements with individual accounts.

#### Collection Arrangement Payment Table

Records in this table represent scheduled payments associated with collection arrangements. Note that records in this table do not represent actual or received payments.

## Collection Exemptions Tables

Collection exemptions tables store data associated with specific collection exemptions. Collection exemptions are exemptions to collections processing made with a customer regarding.

### Collection Exemption Type Table

Records in this table represent specific types of collection exemptions.

### Collection Exemptions Table

Records in this table represent specific collection exemptions with individual accounts.

## Collection Message Tables

Collection message tables store messages created by collections processes run by Oracle Utilities Receivables Component.

### Agency Message Table

Records in this table represent notifications sent to collection agencies regarding an account's outstanding balance. These messages are created by the **Collection Agency Notification - Enter** and **Collection Agency Notification - Update** collections functions.

### Phone Message Table

Records in this table represent telephone contact made with a customer regarding their outstanding balance. These messages are created by the **Phone Contact** collections function.

### Shut Off Message Table

Records in this table represent requests for discontinuation of services, based on the customer's their outstanding balance. These messages are created by the **Service Discontinuation Request** and **Service Discontinuation Request Cancellation** collections functions.

### Letter Message Table

Records in this table represent letters and/or dunning notices sent to a customer regarding their outstanding balance. These messages are created by the **Letter Generation/Dunning Notice** collections function.

### Dynamic Message Table

Records in this table define dynamic message payloads that can be included in collections messages. Records in this table contain the following information:

- **Dynamic Message Code:** A code that designates the dynamic message payload.
- **Variable Name:** The name of the variable that will contain the message payload in XML format.

The following example demonstrates how to define contact information for a collection agency and a message type:

Dynamic Message Code	Variable Name
AGENCYINFO	AGENCYCONTACTINFO
AGENCYINFO	OPCODE
AGENCYINFO	METERHISTORY
AGENCYINFO	MESSAGETYPE

To include the message payload, pass a valid Dynamic Message Code within the activity context of the letter activity. For example, in the input map of the activity context you would include the following:

```
"AGENCYINFO" to /CONTEXT/DYNAMICMSGCODE
```

### Dynamic Message Value Table

Records in this table define the values contained in dynamic message payloads defined in the Dynamic Message table. Records in this table contain the following information:

- **Variable Name:** The name of the variable that will contain the message payload in XML format, from the Dynamic Message table.
- **Variable Source Code:** A code that designates the type of variable. Can be “L” (literal), “Q” (query), “OC” (optional context), or “RC” (required context).
- **Variable Value:** The value of the variable.
  - If the Variable Source Code is “L”, the Variable Value should be a string literal that is directly output to the message payload.
  - If the Variable Source Code is “Q”, the Variable Value should be a query that is run with all parameters (surrounded by %) replaced by values that correspond to the name in the context. More than one record can be returned from the query. In this case, each record is the child of the Variable Name element.
  - If the Variable Source Code is “OC”, the Variable Value should be a valid XPath expression that specifies a value to be included in the payload.
  - If the Variable Source Code is “RC”, the Variable Value should be a valid XPath expression that specifies a value to be included in the payload
- **Parent Variable Name:** Optional name of a previously defined variable that will serve as parent to the variable. For example, when defining the “METER” variable, entering “METERHISTORY” in this column would result in the “METERHISTORY” variable being the parent of “METER” in the context.
- **Exclude Columns:** A comma-separated list of columns names which are return values needed from the parent to the child query, but that are not to be part of the output of the payload.

The following example demonstrates how to define the values of the message payloads defined in the above example:

Variable Name	Variable Source Code	Variable Value	Parent Variable Name	Exclude Columns
AGENCYCONTACTINFO	Q	Select NAME, PHONE From colagency where uidaccount = %%UIDACCOUNT%%		
MESSAGETYPE	L	COLLECTION AGENCY ENTER		
METERHISTORY	Q	SELECT SERVICETYPECODE, UIDMETER FROM METERHISTORY WHERE UIDACCOUNT = %%UIDACCOUNT%%		UIDMETER

Variable Name	Variable Source Code	Variable Value	Parent Variable Name	Exclude Columns
METER	Q	SELECT METERID FROM METER WHERE UIDMETER= %%UIDMETER%%	METERHISTORY	
OPCOCODE	RC	OPCOCODE		

A possible output based on the above example could be:

```
<AGENCYINFO>
  <AGENCYCONTACTINFO>
    <NAME>Acme Collections</NAME>
    <PHONE>365-555-1212</PHONE>
  </AGENCYCONTACTINFO>
  <AGENCYCONTACTINFO>
    <NAME>AAA Collections Agency</NAME>
    <PHONE>456-741-2583</PHONE>
  </AGENCYCONTACTINFO>
<MESSAGE TYPE>COLLECTION AGENCY ENTER</MESSAGE TYPE>
<METERHISTORY>
  <SERVICETYPECODE>EL</SERVICETYPECODE>
  <METER>
    <METERID>A456I2</METERID>
  </METER>
</METERHISTORY>
<OPCOCODE>LODESTAR</OPCOCODE>
</AGENCYINFO>
```

## Other Collections Tables

Other collections tables store data used by the collections functions of Oracle Utilities Receivables Component.

### Collection History Table

Records in this table represent individual collections activities performed for an account.

### Credit Score History Table

Records in this table store the Credit Score for a customer over time. Each record contains a start and stop time that define the period during which a particular credit score applies.

### Factor and Factor Value Tables

Though not used solely by the collections functions of Oracle Utilities Receivables Component, the Factor and Factor Value tables are used to define the day ranges displayed in the Aging pane on the Collections tab. Each day range is defined by a factor in the Factor and Factor Value tables as follows:

#### Factor Table

- **Operating Company:** The operating company associated with accounts that use this day range.
- **Jurisdiction:** The jurisdiction associated with accounts that use this day range.
- **Code:** Must be "BUCKET\_" followed by a number (ex BUCKET\_01, BUCKET\_02, etc.).
- **Name:** Same as Code.
- **Unit of Measure:** NULL

**Factor Value Table**

- **Factor:** The factor from the Factor Table.
- **Effective Date:** The date on which the aging day range in effect.
- **Value:** The maximum number of days in the aging day range, starting from the value of the previous day range.
- **Prorate Flag:** NULL

For example, to define aging day ranges of

<b>0-15</b>	<b>16-30</b>	<b>31-45</b>	<b>46-60</b>	<b>61-75</b>	<b>75+</b>
-------------	--------------	--------------	--------------	--------------	------------

you would define the following factors in the Factor and Factor Value tables

<b>Operating Company</b>	<b>Jurisdiction</b>	<b>Code/Name</b>	<b>Value</b>
Per Account	Per Account	BUCKET_01	15
Per Account	Per Account	BUCKET_02	30
Per Account	Per Account	BUCKET_03	45
Per Account	Per Account	BUCKET_04	60
Per Account	Per Account	BUCKET_05	75



# Collections Arrangements

The Collections arrangements functions are used to create, maintain, and review collection arrangements.

## Creating Collections Arrangements

Collections arrangements are created using the Collection Arrangements function of the Oracle Utilities Receivables Component user interface. This function allows users to view, edit, add, and delete collection arrangements and related payment arrangements for individual accounts.

When a collection arrangement is created, the function automatically creates a collection exemption of type 'Arrangement Exemption' for the total amount of the arrangement. When the arrangement is completed or defaulted, the status of the exemption is set to 'EXPIRED'. Deleting a collection arrangement also deletes the associated collection exemption for that arrangement.

In addition, when a collection arrangement is created, if the amount of the arrangement is greater than the past due amount for the account, the status of any collections processes running for the account are set to SUSPENDED.

## Collection Arrangement Data

Collection arrangement data includes data about individual collection arrangements as well as payments associated with collection arrangements. Collection arrangement data includes the following:

- Unique ID for collection arrangement
- Unique ID of the Account related to the collection arrangement
- Start Date for collection arrangement
- Stop Date for collection arrangement
- Number of payments associated with this collection arrangement.
- Total Amount of all collection arrangement payments associated with this Collection Arrangement.
- Status Code for the collection arrangement. Valid values include CURRENT, DEFAULT, and COMPLETE
- Modification Date for the collection arrangement
- Documentation Date for the collection arrangement (optional)
- Compliance Date (used by the Review Collection Arrangement function.
- Collection Arrangement Payments associated with the arrangement. Each includes:
  - Unique ID of the associated collection arrangement
  - Due Date for the collection arrangement payment
  - Amount Due for the collection arrangement payment

## Collections Arrangements Processing

The Collections arrangements functions can be triggered from an external system via an interface or using a command line program. When triggered, these function send the collection arrangement data (including all collection arrangement payments associated with the collection arrangement) in an XML string to the Financial Engine, which in turn evaluates the data to determine the status of the arrangement. When the functions are triggered, all the information needed to process the file is obtained, and error messages are posted to the database if discrepancies are found.

## Review Collection Arrangement

This function periodically reviews the terms of a collection arrangement to ensure the arrangement is not in Default. The Review Collection Arrangement function performs the following steps:

1. The Collections module checks the Status code of the arrangement. If the Status Code is not CURRENT, it returns the collection arrangement record as is.
2. The Collections module determines the outstanding payments balance by aggregating all collection arrangement payments for all CURRENT collection arrangements related to the Account whose due date is greater than the compliance date. The total is what remains due after the compliance date.
3. If the outstanding payments balance is zero, the status codes of all CURRENT collection arrangements related to the account are changed to COMPLETE.
4. The Collections module determines the outstanding balance by aggregating all posted transactions with a balance whose 'date' is less than or equal to the compliance date.
5. If the outstanding balance is greater than the outstanding payments balance, the status codes of all CURRENT collection arrangements related to the account are changed to DEFAULT.
6. The Collections module returns the updated collection arrangement records.

## Using the Arrangement Review Command Line Program

The Review Collection Arrangement function can be performed using a command line program called **LSArrRev.exe**. In addition to the steps performed by the Review Collection Arrangement function (outlined above), the Arrangement Review command line program also determines if any accounts need to be taken out of or put back into collections, and sets the status of collections processes for any accounts to either Suspended, Terminated, or Resumed, as appropriate.

## Configuring the Arrangement Review Command Line Program

The Arrangement Review command line program must be configured before it can be run, using the syntax shown below. Once configured, the program does not require any parameters to be run (parameters are stored in the Windows Registry). The configuration syntax is:

**LSArrRev.exe** -Configure

The command must be entered on one line. Also, you must either change to the directory in which the program is stored (typically, C:\LODESTAR\CMEBatch) before entering the command, or specify the path in the command.

When you run the command line with the above syntax, the Collection Arrangement Review screen opens.

Enter batch process parameters for the program as follows:

Parameter	Description
Username	The userid for the data source.
Password	The password for userid above.
Qualifier	The optional qualifier for the data source.

Parameter	Description
DSN	<p>Database connection information for the Oracle Utilities Data Repository that contains the rate form record. This parameter is <b>required</b> and must be in one of the following formats:</p> <p>For Oracle databases:</p> <pre>"Data Source=&lt;data_source&gt;;User ID=&lt;user_id&gt;;Password=&lt;password&gt;;LSProvider=ODP;"</pre> <p>where:</p> <ul style="list-style-type: none"> <li>• &lt;data_source&gt; is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\&lt;machine&gt;\oracle\network\admin directory)</li> <li>• &lt;user_id&gt; is the user ID for the database connection</li> <li>• &lt;password&gt; is the password for the supplied user ID.</li> </ul>
Days to Review	<p>How many days in the past an arrangement should be reviewed. For example, if the Days to Review parameter is set to 5, the process would review all arrangements that had at least one payment with a payment due date between today and 5 days in the past.</p>

### Running the Arrangement Review Command Line Program

When the Arrangement Review program runs, it calls the Review Collection Arrangement function. However, depending on the status of the arrangement after review, several steps specific to Collections can be taken, as outlined below:

1. Get all accounts that have arrangements within the specified time range (Days to Review parameter).
2. For each account with an arrangement that meets these parameters, the **Review Collection Arrangement** function is called.
3. If the arrangement is in default (the total amount of the remaining payments is less than the current amount past due), a message is logged in the **Collection History Table** indicating an arrangement has been set to default. Additionally, the associated Collection Exemption that was automatically created when the Arrangement was created will have its status set to EXPIRED. Finally any existing Collections processes that were suspended when the Arrangement was created will be Resumed.
4. If the Arrangement is reviewed and has been successfully completed, a message is logged in the **Collection History Table** indicating an arrangement has been completed. The associated Exemption will be set to EXPIRED, and the past due status will be checked on the account. If the arrangement was successfully completed but the account is still past due (the past due balance is greater than the remaining amount of the arrangement) any collections process suspended when the arrangement was created will be restarted. If the account is no longer past due and there are not any additional exemptions on the account that could have suspended a collections process for the account (such as exemptions for the account that have a status of Current, and the associate Exemption type has a terminate or suspend flag value of "S"), any previously suspended processes will be Terminated and the accounts receivable status and collections status will be reset (set to empty values to indicate account is not currently in collections).
5. If an arrangement was created with a start date in the future, when the arrangement was originally created none of the existing collections processes would have been Suspended or Terminated automatically. Once the arrangement review process is run and it is determined that this arrangement is no longer in the future (the arrangement's Start Date = today's date),

if the account is no longer past due, the process will determine if collections processes should be Terminated or Suspended (terminate or suspend flag on the Exemption Type table for the Collection Arrangement Exemption exemption type) and will do so. However, if the account is still past due (the past due balance is greater than the remaining amount of the arrangement), any suspended processes will be Resumed.

---

## Collection Exemptions

The Collections exemptions functions are used to create, maintain, and review collection exemptions.

### Creating Collections Exemptions

Collections exemptions are created using the Collection Exemptions function of the Oracle Utilities Receivables Component user interface. This function allows users to view, edit, add, and delete collection exemptions for individual accounts.

When creating a collection exemption, leaving the amount blank exempts the entire account. In this case, regardless of what the past due balance is on the account, the account will never go into collections.

When creating a collection exemption, if the Stop Date is not entered, the collection exemption remains in effect indefinitely. The only exception to this rule is in the case of collection exemptions that are created automatically by creating a collection arrangement. These exemptions do not have a Stop Date when they are initially created, but after the arrangement is completed or expired, the Stop Date will be populated with the date that the arrangement was completed or expired.

### Collection Exemption Data

Collection exemption data includes data about individual collection exemptions including the following:

- Unique ID for collection exemption
- Unique ID of the Account related to the collection arrangement
- The Amount of the collection exemption.
- Start Date for collection exemption
- Stop Date for collection exemption
- The type of collection exemption from the **Collection Exemption Type Table**
- An optional note about the collection exemption
- The current status of the exemption. Valid values include CURRENT, and EXPIRED.
- The collection arrangement associated with the exemption

## Reviewing Collections Exemptions

The exemption review process is intended to be executed as a daily batch process, that reviews all collection exemptions to verify they are still current (an exemption with a Stop Date in the future). Additionally the program will start or stop collections processes for the account depending on the past due status for the account after the status of the account's exemptions have been reviewed.

### Using the Exemption Review Command Line Program

The Exemption Review function is performed using a command line program called **LSExempRev.exe**.

### Configuring the Exemption Review Command Line Program

The Exemption Review command line program must be configured before it can be run, using the syntax shown below. Once configured, the program does not require any parameters to be run (parameters are stored in the Windows Registry). The configuration syntax is:

**LSExempRev.exe -Configure**

The command must be entered on one line. Also, you must either change to the directory in which the program is stored (typically, C:\LODESTAR\CMEBatch) before entering the command, or specify the path in the command.

When you run the command line with the above syntax, the Database Connection screen opens.

Enter batch process parameters for the program as follows:

Parameter	Description
Username	The userid for the data source.
Password	The password for userid above.
Qualifier	The optional qualifier for the data source.
DSN	<p>Database connection information for the Oracle Utilities Data Repository that contains the rate form record. This parameter is <b>required</b> and must be in one of the following formats:</p> <p>For Oracle databases:</p> <pre>"Data Source=&lt;data_source&gt;;User ID=&lt;user_id&gt;;Password=&lt;password&gt;;LSProvider=ODP;"</pre> <p>where:</p> <ul style="list-style-type: none"> <li>• &lt;data_source&gt; is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\&lt;machine&gt;\oracle\network\admin directory)</li> <li>• &lt;user_id&gt; is the user ID for the database connection</li> <li>• &lt;password&gt; is the password for the supplied user ID.</li> </ul>

**Running the Exemption Review Command Line Program**

When the Exemption Review program runs, it performs the following steps:

1. Review exemptions for those where the current date is greater than the Stop Date of the exception. For each of these exemptions, set the status to EXPIRED.
2. If an account that is associated with an expired exemption is still past due, resume any collections processes that had previously been Suspended.
3. If an account is no longer past due and if no additional CURRENT exemptions exist on the account that may have suspended collections processes, Terminate any existing collections processes.
4. Review any exemptions that may have been originally created to start at a future date (indicated as any exemptions that have a status of CURRENT and whose Start Date is less than or equal to the current date). For these processes, if once the exemption is applied the account is no longer past due, Suspended or Terminate any collections processes (depending on the Terminate or Suspend flag value for the exemption type).

## Collections Processing and Activities

When a customer's outstanding balance is either past due by a specified period of time or exceeds a specified amount, the customer is put into collections. When this occurs, a series of activities related to collecting the outstanding balance is performed. This series of activities is known as a collections process.

Collections processing requires that the workflow management functionality of Oracle Utilities Billing Component be installed and configured. See the *Part Three: Workflow Management Configuration* for more information.

### Selecting Accounts for Collections

The Account Selection process searches the accounts within the Oracle Utilities Data Repository and selects those that are eligible for Collections, including those that have past due balances and are not currently in Collections. This process performs user-defined business logic to select the accounts to enter Collections. This routine should normally be scheduled as part of nightly batch processing and is most often executed after both Remittance and Billing have successfully completed.

The Account Selection process should also interrogate the age of each past due account and ensure it is placed in Collections at the appropriate point within the process. This ensures that accounts that have been suspended from Collections processing are returned at the appropriate step in the process.

The Manual Submission function of the Oracle Utilities Receivables Component user interface also provides the ability to override the selection process and manually place an account at a certain point within the Collections process.

### Creating an Account Selection Trigger

Because the specific criteria used to select accounts for collections can vary greatly between implementations, creating an Account Selection Trigger (AST) is considered an implementation task. Oracle Utilities Receivables Component doesn't include an AST, but this section provides examples and guidelines for creating an AST that meets your specific business needs.

The business logic used for selecting accounts can use parameters such as Amount Past Due, Number of Days Past Due, or a grace period beyond the due date. These types of parameters can be stored as Factors in the Oracle Utilities Data Repository, allowing business analysts to modify the criteria without requiring changes to the underlying application code. This also allows business analysts to modify some aspects of each strategy without requiring IT support.

The basic steps for any AST are as follows:

1. Review all accounts to ensure that they are still current.
2. Compare the past due amount and the age of any past due balances to Factors stored in the Oracle Utilities Data Repository, and factor in any exemptions or arrangements that have been created for the account
3. Determine if the account should start at step one of a collections process or at an accelerated step.
4. Start the appropriate Collections process based on this information. Oracle Utilities Receivables Component includes processes for both Active and Inactive accounts. In addition, select accounts might use a collections process created during implementation.

After an account is selected to enter Collections, the account is placed within an appropriate Collections process. The Collections process maps out the activities that are required to occur to the account as part of the process, along with any events that may affect the account's path through the process. The Oracle Utilities Billing Component - Workflow Management is the



backbone behind the processing, ensuring that the activities are processed at the appropriate time and are completed successfully.

### Recommended Detailed Steps

This section outlines several recommended steps for creating an AST for use with Oracle Utilities Receivables Component. This section references several factors described in **Account Selection Factors** on page 9-17.

#### Step One: Gather list of working accounts

1. For active accounts, the working set of accounts will be those accounts that have a receivable status not equal to "COLLECTIONS" and that have a balance (new activity + balance) greater than the Factor Value for the ACT\_COLLECTION\_ENTRY\_BALANCE factor for the account's operating company and jurisdiction.
2. For inactive accounts, this initial account list is where receivable status is not equal to "INACTIVE COLLECTIONS" and that have a balance (new activity + balance) greater than the Factor Value for the INACT\_COLLECTION\_ENTRY\_BALANCE factor for the account's operating company and jurisdiction. Additionally the account status code must equal "FINAL" and the account Stop Time cannot be null.

If you use different Account Status codes (stored in the Account Status table) than ACTIVE and FINAL you can modify these steps accordingly.

#### Step Two: Determine Collectible Balance

After gathering a working set of accounts, the next step is to determine the collectible balance for each account. This can be done manually if the formula is very involved and specific to your business needs or you can use the functionality that Oracle Utilities provides in the Collection Information function of the Collections interface.

This method calculates the collectible balance as follows:

Collectible Balance = Past Due Amount - Exempt Amount

where:

- **Past Due Amount:** is the sum of all Transactions in the Transaction table where the Cancel Time is Null, the Charge Type is 'CH' and the Due Date is in the past. The Credits (Transaction records where the Charge Type is 'CR') are then subtracted from this number to get the final Past Due Amount.
- **Exempt Amount:** is the sum of all "CURRENT" exemptions for this account. If an Exemption exists on this account that has a Null amount then the entire account is exempt and the Past Due Balance is 0.

#### Step Three: Determine Age of Oldest Past Due Transaction

The next step is to determine the age of oldest past due transaction. This can be calculated against the invoice date in the Transaction table or any other date as dictated by specific business needs.

#### Step Four: Compare Collectible Balance and Age of Oldest Past Due Transaction

The next step is to compare the Collectible Balance and the Age of the oldest past due transaction to Factors stored in the Oracle Utilities Data Repository. Oracle Utilities provides pre-supplied factors for this purpose, but other factors can be used as well. See **Account Selection Factors** on page 9-17 for more information about these factors.

#### Step Five: Determine Starting Point for Collections Processes

The next (optional) step is to determine the account should be placed in collections at an accelerated step in the process. Two specific factors are provided for this step. See **Account Selection Factors** on page 9-17 for more information about these factors.

**Step Six: Obtain Collections Process to Start**

The next step is to get the name of the process you wish start. This information can be stored in a \*.ini file, the system registry, or an input file that is passed to your program upon execution. Items that should be included are:

- Process to start for Active Collections
- Process to start for Inactive Collections
- Active Collections accelerated step (if applicable)
- Inactive Collections accelerated step (if applicable)

**Step Seven: Compile Data Needed to Start Collections Processing**

The last step is to compile the data needed to start a collections process for the account and put that data in the proper format used by the Oracle Utilities Billing Component - Workflow Management.

At this point in the process you have determined if the account is active or inactive (or another specific status code that you have defined), you've determined if the account should be in collections or not, and you've determined if the account should start at step one or an accelerated step. Now you have to build the proper process context XML document required by the Start Process function of the Oracle Utilities Billing Component - Workflow Management.

You can use the Create Process Context function of the Collections interface to create the Process Context using the Context Value and Process Context Value tables.

See **Chapter 12: Setting Up Workflow Management Database Tables** in the *Oracle Utilities Billing Component - Workflow Management Installation and Configuration Guide* for more information about setting up and configuring the **Variable Source**, **Context Value** and **Process Context Value** tables.

An example of how this function could be called from a Visual Basic application is below:

```
Set oColObject = CreateObject("CollectionObjects.ProcessXML")
xml = oColObject.createProcessContext(xmlDataSource, uidProcess,
uidAccount, "UIDACCOUNT=" & uidAccount & ";")
```

Once the process context is properly generated, the Start method for the appropriate collections process can be called. Note that you will need to retrieve the process you wish to start from the \*.ini file, the system registry, or the input file that was used when starting the program. (see Step Six, above).

An example of how the Start function can be called follows:

```
Set wrkflw = CreateObject("lswrkflw.ProcessInstance")
retXML = wrkflw.Start(xmlDataSource, xml)
If InStr(retXML, "<STATE>RUNNING</STATE>") > 0 Then
Call CommitTransaction
Else
Call RollBackTransaction
End If
```

## Account Selection Factors

The recommended Account Selection process described above references the following factors, which must be set up in the Factor and Factor Value tables:

**ACT\_COLLECTION\_ENTRY\_BALANCE:** If an active account's balance exceeds the value of this factor, the account becomes eligible to be put into collections.

**INACT\_COLLECTION\_ENTRY\_BALANCE:** If an inactive account's balance exceeds the value of this factor, the account becomes eligible to be put into collections.

**ACT\_COLLECTION\_ENTRY\_AGE:** If the oldest past due transaction (based on Invoice Date) for an active account is older than the value of this factor, the account becomes eligible to be put into collections.

**INACT\_COLLECTION\_ENTRY\_AGE:** If the oldest past due transaction (based on Invoice Date) for an inactive account is older than the value of this factor, the account becomes eligible to be put into collections.

**ACT\_COLLECTION\_ACCELERATE\_BALANCE:** If an active account's balance exceeds the value of this factor, the account is eligible to be put into collections starting at an accelerated step.

**INACT\_COLLECTION\_ACCELERATE\_BALANCE:** If an inactive account's balance exceeds the value of this factor, the account is eligible to be put into collections starting at an accelerated step.

**ACT\_COLLECTION\_ACCELERATE\_AGE:** If the oldest past due transaction (based on Invoice Date) for an active account is older than the value of this factor, the account is eligible to be put into collections starting at an accelerated step.

**INACT\_COLLECTION\_ACCELERATE\_AGE:** If the oldest past due transaction (based on Invoice Date) for an inactive account is older than the value of this factor, the account is eligible to be put into collections starting at an accelerated step.

## Initiating User-Defined Collections Processes

The Account Selection Trigger program can be used to start collections processes based on two pre-defined workflow processes defined using the Oracle Utilities Billing Component - Workflow Management application. These processes are the Active Collections process, and the Inactive Collections process, and are supplied by Oracle Utilities as part of the Collections functionality of Oracle Utilities Receivables Component.

To configure the Account Selection Trigger program to start a user-defined collections process, use the following procedure:

1. Configure the context variables for the user-defined process in the **Variable Source** and **Context Value** tables.
2. Associate the context variables to the user-defined process in the **Process Context Value** table.
3. Specify the Process Name of the user-defined process as you would for the Active Collections or Inactive Collections processes. In the \*.ini file, the system registry, or the input file used by the AST.

See **Chapter 12: Setting Up Workflow Management Database Tables** in the *Oracle Utilities Billing Component - Workflow Management Installation and Configuration Guide* for more information about setting up and configuring the **Variable Source**, **Context Value** and **Process Context Value** tables.

## Collections Process Activities

Collections processes are made up of a series of activities, linked together in a specific sequence that can be used to perform the Collections activities of an organization. These processes and activities are controlled via the workflow management functionality of Oracle Utilities Billing Component. Workflow Management can be used to configure these activities into specific collections processes. The available collections activities include:

- **Compute Internal Performance Score**
- **Update Status**
- **Late Payment Fee Calculation**
- **Letter Generation/Dunning Notice**
- **Phone Contact**
- **Service Discontinuation Request**
- **Service Discontinuation Request Cancellation**
- **Collection Agency Notification - Enter**
- **Collection Agency Notification - Update**
- **Write Off**
- **Collection Information**
- **Insert Collection History Record**
- **Update Account Statuses**
- **Post Activity Event**

### Compute Internal Performance Score

Each customer has a performance score that provides an indication of the payment performance history for the customer. The performance score is calculated based on two factors assigned to Transaction IDs, collection arrangement types, and collections process activities. These factors are weight (measuring how heavily a transaction, collection arrangement, or activity should be weighed in the calculation) and duration (measuring how long a transaction, collection arrangement, or activity should be used in the calculation). For example, a Late Payment Change (LPC) might have a Weight of 10 and a Duration of three months, while a service discontinuation might have a Weight of 50 and a Duration of 12 months. For transactions, the invoice date is used to determine duration. Collections process activities use the activity's execution time, and arrangements use the arrangement start date.

Default arrangements are also included a customer's performance score. Any time a collection arrangement is defaulted and a record entered in the **Collection History Table** for the defaulted arrangement, the weight for the defaulted arrangement (based on the Collection Arrangement Type) will be included in the customer's performance score. As with the other entities, this only occurs if the date the arrangement was defaulted falls within the duration value for the default arrangement. By default, the Weight and Duration values of defaulted collection arrangements are set to 50, but can be modified.

The internal performance score is a numeric value and can be interpreted so that each customer then falls into one of X number of categories (such as A, B, or C or Good, Fair, or Poor) in which customers can be evaluated. Weight and Durations for Transaction IDs, Collection Arrangement Types, and Collections Process Activities are user-defined.

Performance scores are stored in the **Credit Score History Table**. This table stores all instances when the performance score was calculated. Calculation of performance scores is automatically performed on a periodic basis or upon request.

The Compute Performance Score function is performed using a command line program called **LSPerfScore.exe**.

### Configuring the Calculate Performance Score Command Line Program

The Calculate Performance Score command line program must be configured before it can be run, using the syntax shown below. Once configured, the program does not require any parameters to be run (parameters are stored in the Windows Registry). The configuration syntax is:

**LSPerfScore.exe** -Configure

The command must be entered on one line. Also, you must either change to the directory in which the program is stored (typically, C:\LODESTAR\CMEBatch) before entering the command, or specify the path in the command.

When you run the command line with the above syntax, the Database Connection Information screen opens.

Enter batch process parameters for the program as follows:

Parameter	Description
Username	The userid for the data source.
Password	The password for userid above.
Qualifier	The optional qualifier for the data source.
DSN	<p>Database connection information for the Oracle Utilities Data Repository that contains the rate form record. This parameter is <b>required</b> and must be in one of the following formats:</p> <p>For Oracle databases:</p> <pre>"Data Source=&lt;data_source&gt;;User ID=&lt;user_id&gt;;Password=&lt;password&gt;;LSProvider=ODP;"</pre> <p>where:</p> <ul style="list-style-type: none"> <li>• &lt;data_source&gt; is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\&lt;machine&gt;\oracle\network\admin directory)</li> <li>• &lt;user_id&gt; is the user ID for the database connection</li> <li>• &lt;password&gt; is the password for the supplied user ID.</li> </ul>
SQL for Customers to be Reviewed	SQL statements that define the customer list to be reviewed. For example, to calculate the performance scores for all customers, the SQL "SELECT DISTINCT(UIDCUSTOMER) FROM ACCOUNT WHERE STATUS <> 'FINAL'" This would review all customers that have active accounts in the Oracle Utilities Data Repository.

## Running the Calculate Performance Score Command Line Program

When the Calculate Performance Score program runs, it performs the following steps:

1. For each customer, track each transaction, collection arrangement, and collections process activity executed for each of the customer's accounts.
2. Determine which of these is to be included in the performance score based on the Duration of the transactions (based on Transaction ID), collection arrangements (based on Collection Arrangement Type), and collections process activities executed for the customer's accounts.
3. Total the Weight of all transactions, collections arrangements, and collections process activities to be included in the performance score for each account, and then combine the account totals into a customer total.
4. The new total for each customer is stored in the **Credit Score History Table** table. If a previous record exists in the table, set the Stop Date for the previous record as today's date and add the new record. Each customer can only have one valid record in the **Credit Score History Table** for any given time period.

## Update Status

The Update Status activity determines if an account should remain in a collections process or be dropped. This activity is triggered by events occurring within the system that could change an account's status. This activity can be configured to occur 'real time' when these events occur on an account, and include a change in the account's balance, Collection Exemptions, and Collection Arrangements.

- This activity updates several fields in the database to account for the customer's collection's status, including the Collection Status and Receivable Status fields in the Account FME table.
- The activity scans account information and determines if the account is in collections or not. The Update Status activity determines if there are transactions that have a positive balance past the due date.

If not, then the following updates occur:

- The Receivable Status is updated to "Current"
- The Collections Status is updated to "Not-in-Collections"
- The account is removed from the collections process

## Input Parameters

The Update Status activity uses the following input parameters:

- **UIDACCOUNT** (required): The UID of the account.
- **UIDCOLEXEMPTION** (optional): The UID of the exemption that will be used to update the account's collection status. If provided, this variable is written to the xml payload of the Collection Object History record that corresponds to the executed Updated Status activity.
- **COLLECTIONSTATUS** (optional): The value to update the Collection Status field in Account FME table.
- **TERMINATEORSUSPEND** (optional): A flag that indicates whether to Terminate ("T") or Suspend ("S") collections processes for the account, based on the Exemption type. If no value is passed in and an account is determined to no longer be in Collections, then the collections processes will be suspended. Generally the value for this variable is provided by the application that posts the event to indicate that an exemption was created to the workflow engine.

## Output Values

The Update Status activity returns the following output values:

- **TERMINATE:** A flag that indicates if collections processes were terminated. This flag will either be set to “T”, “S”, or “N”. “T” indicates that processes were terminated. “S” indicates that processes were suspended. “N” indicates that when the Update Status active was executed, the account was still considered past due so no action was taken.
- **ACCOUNTPASTDUE:** “Y” or “N”, indicating if the account was past due or not.

This activity can be only be triggered from a collections process.

## Late Payment Fee Calculation

When an account is considered in arrears, a late payment charge can be calculated and applied to the account.

Late Payments can be calculated in two ways, either as a one-time charge (based on a percentage applied to the total Past Due Amount for each invoice), or a flat fee charge. Both types of calculations are based on factors stored in the Factor and Factor Value tables. These factors is associated to the account through the Factory History table or by Operating Company and Jurisdiction.

## Input Parameters

The Late Payment Fee Calculation activity uses the following input parameters:

- **UIDACCOUNT:** The UID of the account. Used only when triggered using the CALCULATE\_LATEPAYMENT Rules Language function.
- **DATETYPE** (required): Indicates which date field in the LSTransaction table is used to sum up all charge transactions as part of the past due calculation. Valid values are TRANSACTIONTIME, STATEMENTDATE, INVOICEDATE, and DUEDATE.
- **COLLECTIONSTATUS:** The value to update Collection Status field in the AccountFME table.

## Output Values

The Late Payment Fee Calculation activity returns the following output values:

- **LATEPAYMENTFEE:** The late payment fee that has been calculated.
- **LIMITAMOUNT:** The maximum amount the late payment should be for the account.

## Factors

This activity uses the following factors, which must be set up in the Factor and Factor Value tables:

- **Grace Period:** This represents the number of days after the due date that should be allowed for an Oracle Utilities Receivables Component transaction (LSTRANSACTION table) before a transaction should be included in the total amount past due function. The value stored in the Factor Value table should be the number of days for the grace period.
  - **Name:** GRACEPERIOD
  - **Code:** GRACEPERIOD
- **Late Fee Charge Type:** Indicates if the late payment fee calculation function should calculate a flat charge (LFC), as a percentage of the overdue amount (LFP), or both a flat charge and a percentage of the overdue amount. (BOTH)
  - **Name:** LATEFEECHARGETYPE
  - **Code:** LFC, LFP, or BOTH, as appropriate
- **Late Fee Charge:** The amount of late payment charge returned from the function if the Late Fee Charge Type code is set to LFC.

- **Name:** LFCHARGE
- **Code:** LFCHARGE
- **Late Fee Percentage:** The amount of late payment charge returned from the function if the Late Fee Charge Type code is set to LFP.
  - **Name:** LFCHARGE
  - **Code:** LFCHARGE
- **Late Fee Limit Amount:** Represents a ceiling for the amount a customer can be charged if an account is overdue. If the amount calculated by the function exceeds the value for this factor, an error is returned as well as the late fee charge amount.
  - **Name:** LFLIMITAMOUNT
  - **Code:** LFLIMITAMOUNT

The late payment fee can be one of the following:

- A one-time charge based on a percentage applied to the total Past Due Amount for each invoice
- A flat fee charge
- A flat fee **and** a one-time percentage charge
- A flat fee **or** a one time percentage

This activity can be triggered from either a collections process or from the Oracle Utilities Rules Language. See the **CALCULATE\_LATEPAYMENT Function** on page D-33 for more information about triggering this activity from the Rules Language.

## Letter Generation/Dunning Notice

At specified times in a collections process, letters/dunning notices are generated and sent to the customer. These are created by the Letter Generation activity, which generates a message to a target letter queue or output file.

### Input Parameters

The Letter Generation/Dunning Notice activity uses the following input parameters:

- **DELIVERYTYPE** (required): Indicates the manner in which the letter is delivered. Valid values are FILE or QUEUE. If set to FILE, the letter is output to the location detailed in the File Path parameter, with name given in the File Name parameter. This file will continue to be appended to with each letter output. If set to QUEUE, the letter is posted to the Letter Message table.
- **OPCODE** (required): Valid operating company code.
- **JURISCODE** (required): Valid jurisdiction code.
- **FILEPATH**: The path to the file to which the letter is output. Only used if the DELIVERYTYPE parameter is set to FILE. Example: "C:\LetterGeneration\".
- **FILENAME**: The name of the file to which the letter is output. Only used if the DELIVERYTYPE parameter is set to FILE. Example: "Letter.xml".
- **MESSAGESUBTYPE**: The value to be output in the letter. This value is not used for any processing.
- **COLLECTIONSTATUS**: The value to update Collection Status field in Account FME table.
- **CALCSHUTOFF**: Set to 'Y' if needed. This outputs the calculated shut off date on the Account node of the letter.



- **USEBUSINESSDAY:** Set only if CALCSHUTOFF is set to 'Y'. Finds the next business day that the shutoff should occur on.
- **SHUTOFFDATE:** The non-calculated shut off date value to be output in the letter.

### Output Values

The Letter Generation/Dunning Notice activity returns the following output values:

- **TOTALAMOUNTDUE:** The total amount due for the account, calculated based on the Last Balance and New Activity fields in the Account FME table.
- **PASTDUE:** The total past amount from LSTransaction table
- **SHUTOFFDATE:** The date calculated for shut off, or the date passed in on the SHUTOFFDATE input parameter.

### Defining the Letter Generation Activity

Defining the letter generation activity requires three steps:

1. Select a letter queue or output file
2. Specify the header data for the message. The header contains information about the type of dunning notice to send, and other instructions for the letter generation application to generate the letter. This information must be set-up during configuration.
3. The Letter Generation activity sends a message to the selected target queue or file that consists of the Collection Activities Payload Information in XML format. See **Collection Activities Payload Information** on page 9-33 for more information.

After sending the letter message to the letter generation queue, the activity waits for a response from the messaging system that the letter has been sent. When the letter generation activity receives this reply message, it creates a record in the **Letter Message Table** with the date sent, ACCOUNT ID, type of letter sent and the payload of the letter.

This activity can be only be triggered from a collections process.

### Phone Contact

At specified times in the process, a request for customer phone contact can be generated. The request can be placed into a collections work queue, sent via a transaction to the Customer Relationship Management (CRM) system for a call by a Customer Service Representative (CSR) or Account Executive, or output to a file.

### Input Parameters

The Phone Contact activity uses the following input parameters:

- **DELIVERYTYPE** (required): Indicates the manner in which the message is delivered. Valid values are FILE or QUEUE. If set to FILE, the message is output to the location detailed in the File Path parameter, with name given in the File Name parameter. This file will continue to be appended to with each message output. If set to QUEUE, the notice is posted to the Phone Message table.
- **OPCODE** (required): Valid operating company code.
- **JURISCODE** (required): Valid jurisdiction code.
- **FILEPATH:** The path to the file to which the message is output. Only used if the DELIVERYTYPE parameter is set to FILE. Example: "C:\PhoneContact\".
- **FILENAME:** The name of the file to which the message is output. Only used if the DELIVERYTYPE parameter is set to FILE. Example: "Phone.xml".
- **MESSAGESUBTYPE:** The value to be output in the message. This value is not used for any processing.

- **COLLECTIONSTATUS:** The value to update Collection Status field in Account FME table.
- **CALCSHUTOFF:** Set to 'Y' if needed. This outputs the calculated shut off date on the Account node of the message.
- **USEBUSINESSDAY:** Set only if CALCSHUTOFF is set to 'Y'. Finds the next business day that the shutoff should occur on.
- **SHUTOFFDATE:** The non-calculated shut off date value to be output in the message.

### Output Values

The Phone Contact activity returns the following output values:

- **TOTALAMOUNTDUE:** The total amount due for the account, calculated based on the Last Balance and New Activity fields in the Account FME table.
- **PASTDUE:** The total past amount from LSTransaction table
- **SHUTOFFDATE:** The date calculated for shut off, or the date passed in on the SHUTOFFDATE input parameter.

### Defining the Phone Contact Activity

Defining the phone contact activity requires three steps:

1. Select a phone call work queue, CRM message queue or output file for a CRM system.
2. Specify the header data for the message. The header contains information about the type of phone call, priority, # of repeat attempts and other required instructions (if any). This information must be set-up during configuration.
3. The activity sends a message to the selected target queue or file that consists of the Collection Activities Payload Information in XML format. See **Collection Activities Payload Information** on page 9-33 for more information.

Optionally, after the phone call activity sends the phone call request it can be configured to wait do the following activities:

1. Receive a message from the phone call message queue that the phone call has been attempted.
2. Receive information about the disposition of that attempt and a note. Dispositions can be “Attempted”, “Completed” or “Invalid”. Notes contain information about the disposition.
3. Post that information to the **Phone Message Table** including date attempted, account, type of call, disposition and any notes.

The phone call activity repeats these three steps for the specified # of attempts or until it receives a completed disposition.

This activity can be only be triggered from a collections process.

## Service Discontinuation Request

If the customer does not respond to gentle reminders to pay their overdue balance, a service discontinuation request is generated and sent. This activity sends a message to a specified service discontinuation queue.

### Input Parameters

The Service Discontinuation Request activity uses the following input parameters:

- **DELIVERYTYPE** (required): Indicates the manner in which the message is delivered. Valid values are FILE or QUEUE. If set to FILE, the message is output to the location detailed in the File Path parameter, with name given in the File Name parameter. This file will continue to be appended to with each message output. If set to QUEUE, the message is posted to the Shut Off Message table.
- **OPCODE** (required): Valid operating company code.
- **JURISCODE** (required): Valid jurisdiction code.
- **FILEPATH**: The path to the file to which the message is output. Only used if the DELIVERYTYPE parameter is set to FILE. Example: "C:\ShutOff\".
- **FILENAME**: The name of the file to which the message is output. Only used if the DELIVERYTYPE parameter is set to FILE. Example: "ShutOff.xml".
- **MESSAGESUBTYPE**: The value to be output in the message. This value is not used for any processing.
- **COLLECTIONSTATUS**: The value to update Collection Status field in Account FME table.
- **CALCSHUTOFF**: Set to 'Y' if needed. This outputs the calculated shut off date on the Account node of the message.
- **USEBUSINESSDAY**: Set only if CALCSHUTOFF is set to 'Y'. Finds the next business day that the shutoff should occur on.
- **SHUTOFFDATE**: The non-calculated shut off date value to be output in the message.

### Output Values

The Service Discontinuation Request activity returns the following output values:

- **TOTALAMOUNTDUE**: The total amount due for the account, calculated based on the Last Balance and New Activity fields in the Account FME table.
- **PASTDUE**: The total past amount from LSTransaction table
- **SHUTOFFDATE**: The date calculated for shut off, or the date passed in on the SHUTOFFDATE input parameter.

### Processing

The service discontinuation request activity sends a message to the selected target queue or file that consists of the Collection Activities Payload Information in XML format. See **Collection Activities Payload Information** on page 9-33 for more information.

After the message has been sent, the activity should do the following:

- Wait for a response that from the message queue that the shut-off request has been delivered.
- Create a record in the **Shut Off Message Table** indicating that the shut-off was sent, time delivered, and a note about to whom.
- Wait for a response from the message queue that the shut-off has been completed or not.

If the shut-off was complete, the activity creates a record in the **Collection History Table** indicating the date that the customer was shut-off.

If the customer was not shut-off, a work queue item is created with the reason that it was not shut-off.

If the activity waits more than specified period of time for a confirmation that the customer has been shut off and no response has been received, then it posts work queue item with a note that shut-off was not complete.

This activity can be only be triggered from a collections process.

## Service Discontinuation Request Cancellation

This activity cancels a previous request for a service discontinuation. This activity should be invoked when a payment or credit adjustment comes in that brings an account current or out of collections and there is a pending service discontinuation that has yet to be completed. This activity also applies for reconnections. The number of days after a service disconnection is completed that a reconnect can be issue is stored in the Factor and Factor Value tables. If a payment, exemption or arrangement comes in during that window, a cancellation of the Service Discontinuation request is issued.

The Service Discontinuation Request Cancellation activity sends out a Service Disconnect Cancel message if the original Service Disconnection Request (shut off) was issued within a configurable grace period AND if the account is no longer past due. This activity can be configured to be run whenever a collection Exemption or Arrangement is created for an account, or when a payment or credit is applied to the account. The number of days for this grace period is stored in the Factor Value table for the DISCONNECTCANCELDAYS factor. This factor value is used to calculate the last date of the grace period for a shut off to be cancelled. This calculation is done by either adding the factor value to the stop time of the most recent shut off activity, or adding this factor to a shut off date that is stored in a context variable within the most recent shut off activity. If the date of execution for the Service Discontinuation Request Cancellation (SDC) activity is within this calculated grace period date, then an SDC message will be generated. An SDC message can also be generated by not calculating a grace period at all, but by simply sending out an SDC if a previous shut off activity has been executed and the account is no longer past due. An SDC message will contain the same message payload as the original shut off message, however the MESSAGECODE in the SHUTOFFMSG table will be SERVICEDISCONNECTCANCEL as opposed to SERVICEDISCONNECT.

### Input Parameters

The Service Discontinuation Request Cancellation activity uses the following input parameters:

- **DELIVERYTYPE** (required): Indicates the manner in which the message is delivered. Valid values are FILE or QUEUE. If set to FILE, the message is output to the location detailed in the File Path parameter, with name given in the File Name parameter. This file will continue to be appended to with each message output. If set to QUEUE, the message is posted to the Shut Off Message table.
- **OPCODE** (required): Valid operating company code.
- **JURISCODE** (required): Valid jurisdiction code.
- **GRACEPERIODTYPE** (required): Indicates way in which a disconnect cancel grace period can be calculated. Valid values are OVERRIDE, SHUTOFFCONTEXTVARIABLE, and ACTIVITYSTOPDATE. OVERRIDE generates a disconnect cancel message if a previous shut off activity that was either completed or is currently running is present. ACTIVITYSTOPDATE causes a disconnect cancel message to be generated if the system date is less than or equal to ( $\leq$ ) the most recent shut off activity's stop date, plus the value of the DISCONNECTCANCELDAYS factor. SHUTOFFCONTEXTVARIABLE performs the same calculation as the ACTIVITYSTOPDATE but instead of the activity stop date, the date specified in the SHUTOFFVARIABLE that was passed into the SHUTOFF activity will be used.

- **SHUTOFFVARIABLE** (required if GRACEPERIODTYPE = SHUTOFFCONTEXTVARIABLE): The name of context variable where the shut off date will be stored in the Shut Off activity
- **FILEPATH**: The path to the file to which the message is output. Only used if the DELIVERYTYPE parameter is set to FILE. Example: "C:\ShutOff\".
- **FILENAME**: The name of the file to which the message is output. Only used if the DELIVERYTYPE parameter is set to FILE. Example: "ShutOff.xml".
- **MESSAGESUBTYPE**: The value to be output in the message. This value is not used for any processing.
- **COLLECTIONSTATUS**: The value to update Collection Status field in Account FME table.
- **CALCSHUTOFF**: Set to 'Y' if needed. This outputs the calculated shut off date on the Account node of the message.
- **USEBUSINESSDAY**: Set only if CALCSHUTOFF is set to 'Y'. Finds the next business day that the shutoff should occur on.
- **SHUTOFFDATE**: The non-calculated shut off date value to be output in the message.
- **DYNAMICMSGCODE**: The value matching the Dynamicmsgcode field in the Dynamic Message table.

### Output Values

The Service Discontinuation Request Cancellation activity returns the following output values:

- **TOTALAMOUNTDUE**: The total amount due for the account, calculated based on the Last Balance and New Activity fields in the Account FME table.
- **PASTDUE**: The total past amount from LSTransaction table
- **SHUTOFFDATE**: The date calculated for shut off, or the date passed in on the SHUTOFFDATE input parameter.
- **CANCELSENT**: 'Y' is cancellation was sent, 'N' if cancellation was not sent, and "No Previous Shutoff Found" if a Service Discontinuation Request was not found.

### Processing

The activity creates a record in the **Collection History Table** that contains the same payload information as the service discontinuation request, but with a cancellation message type.

This activity can be only be triggered from a collections process.

### Collection Agency Notification - Enter

Sometimes attempts to collect the receivable balance need stronger actions. In this case, an account can be sent to a collection agency (either first party or third party agency) for further action.

Determining the appropriate Collection Agency to send the account to is based on a number of factors, including Jurisdiction, Revenue Code, Operating Company, Collection Agency Type, and Dollar threshold. Based on these criteria, an account will be directed to the appropriate agency.

### Collection Agency Selection Process

First, the system searches for an exact match on Jurisdiction (JURISCODE), Revenue Code (REVENUECODE), and Operating Company (OPCOCODE).

If no exact matches are found, then the system searches for a partial match on Jurisdiction (JURISCODE), Revenue Code (REVENUECODE), and Operating Company (OPCOCODE)  
For a partial match, one of the following is true:

- The JURISCODE is the same as the account and the OPCOCODE and REVENUECODE are Null, or
- The OPCOCODE is the same as the account and the JURISCODE and REVENUECODE are Null, or
- The REVENUECODE is the same as the account and the OPCOCODE and JURISCODE are Null, or
- The JURISCODE and OPCOCODE are the same as the account and the REVENUECODE is Null, or
- The JURISCODE and REVENUECODE are the same as the account and the OPCOCODE is Null, or
- The OPCOCODE and REVENUECODE are the same as the account and the JURISCODE is Null.

If none of the above partial matches are found, and the JURISCODE, REVENUECODE and OPCOCODE are all Null, the system uses other fields to find a match.

In addition to Jurisdiction, Revenue Code, and Operating Company fields above, the required fields Upper Amount (UPPERAMT), Lower Amount (LOWERAMT), Collection Agency Type (COLAGENCYTYPE), Start Date (STARTDATE), and Stop Date (STOPDATE) are also included in the checks. For example, in the first case above where Jurisdiction, Revenue Code, and Operating Company are an exact match, the system also assumes an exact match on Collection Agency Type, the system date is between the agency's Start Date and Stop Date, and the past due amount for the account is between the Upper Amount and the Lower Amount.

If more than one agency or no agency is found to match for any three checks, an error is raised.

### Input Parameters

The Collection Agency Notification - Enter activity uses the following input parameters:

- **DELIVERYTYPE** (required): Indicates the manner in which the message is delivered. Valid values are FILE or QUEUE. If set to FILE, the message is output to the location detailed in the File Path parameter, with name given in the File Name parameter. This file will continue to be appended to with each message output. If set to QUEUE, the message is posted to the Shut Off Message table.
- **OPCOCODE** (required): Valid operating company code.
- **JURISCODE** (required): Valid jurisdiction code.
- **FILEPATH**: The path to the file to which the message is output. Only used if the DELIVERYTYPE parameter is set to FILE. Example: "C:\AgencyNotify\".
- **FILENAME**: The name of the file to which the message is output. Only used if the DELIVERYTYPE parameter is set to FILE. Example: "AgencyNotify.xml".
- **MESSAGESUBTYPE**: The collection agency to be output in the message. This value is used to determine which collection agency should be notified.
- **COLLECTIONSTATUS**: The value to update Collection Status field in Account FME table.
- **CALCSHUTOFF**: Set to 'Y' if needed. This outputs the calculated shut off date on the Account node of the message.
- **USEBUSINESSDAY**: Set only if CALCSHUTOFF is set to 'Y'. Finds the next business day that the shutoff should occur on.
- **SHUTOFFDATE**: The non-calculated shut off date value to be output in the message.

## Output Values

The Collection Agency Notification - Enter activity returns the following output values:

- **TOTALAMOUNTDUE:** The total amount due for the account, calculated based on the Last Balance and New Activity fields in the Account FME table.
- **PASTDUE:** The total past amount from LSTransaction table
- **SHUTOFFDATE:** The date calculated for shut off, or the date passed in on the SHUTOFFDATE input parameter.

## Processing

The Collection Agency Notification - Enter, activity generates a message to the Collection Agency Notification queue with a payload of the account details required to enter a 3rd party collections process.

The activity waits for a response from the agency noting the customer has been successfully entered into third party collections.

The Collection Agency Notification Activity performs the following steps.

1. Gets the header fields for the collection agency from the **Collection Agency Table**. The header contains information to route the request to the appropriate place and other required instructions for the delivery of the notification.
2. Sends a message to the selected target queue or file that consists of the Collection Activities Payload Information in XML format. See **Collection Activities Payload Information** on page 9-33 for more information.

When this activity receives a message from the message queue that the third party has begun collections, the activity enters a Start Date in the **Collection Program X Account Table** associating the account to a specific collection agency with a specific start time, and creates a record in the **Agency Message Table** indicating that the message has been sent to the collection agency.

This activity can be only be triggered from a collections process.

## Collection Agency Notification - Update

The Collection Notification - Update activity generates a message to the Collection Agency Notification queue with a payload of the account details required to tell the agency that a balance change has occurred within the account. This balance change provides the agency with updated financial information on the account. Each time the financial balance changes on an account, a transaction is generated giving the Collection Agency updated financial information. The agency can use the Last Payment and Last Payment Date to determine if this transaction was generated by a payment event. Since Collection agencies only get compensated for payment collected on accounts, this field allows the agency to identify if a payment was made.

## Input Parameters

The Collection Agency Notification - Update activity uses the following input parameters:

- **DELIVERYTYPE** (required): Indicates the manner in which the message is delivered. Valid values are FILE or QUEUE. If set to FILE, the message is output to the location detailed in the File Path parameter, with name given in the File Name parameter. This file will continue to be appended to with each message output. If set to QUEUE, the message is posted to the Shut Off Message table.
- **OPCOCODE** (required): Valid operating company code.
- **JURISCODE** (required): Valid jurisdiction code.
- **FILEPATH:** The path to the file to which the message is output. Only used if the DELIVERYTYPE parameter is set to FILE. Example: "C:\AgencyNotify\".

- **FILENAME:** The name of the file to which the message is output. Only used if the DELIVERYTYPE parameter is set to FILE. Example: "AgencyNotify.xml".
- **MESSAGESUBTYPE:** The value to be output in the message. This value is not used for any processing.
- **COLLECTIONSTATUS:** The value to update Collection Status field in Account FME table.
- **CALCSHUTOFF:** Set to 'Y' if needed. This outputs the calculated shut off date on the Account node of the message.
- **USEBUSINESSDAY:** Set only if CALCSHUTOFF is set to 'Y'. Finds the next business day that the shutoff should occur on.
- **SHUTOFFDATE:** The non-calculated shut off date value to be output in the message.

### Output Values

The Collection Agency Notification - Update activity returns the following output values:

- **TOTALAMOUNTDUE:** The total amount due for the account, calculated based on the Last Balance and New Activity fields in the Account FME table.
- **PASTDUE:** The total past amount from LSTransaction table
- **SHUTOFFDATE:** The date calculated for shut off, or the date passed in on the SHUTOFFDATE input parameter.

### Processing

The Collection Agency Notification Activity - Update performs the following steps.

1. Looks up the Agency that the account is currently assigned to
2. Gets the header fields for the collection agency from the **Collection Agency Table**. The header contains information to route the request to the appropriate place and other required instructions for the delivery of the notification.
3. Sends a message to the selected target queue or file that consists of the Collection Activities Payload Information in XML format. See **Collection Activities Payload Information** on page 9-33 for more information.

The activity waits for a response from the agency noting the customer has been successfully entered into third party collections, and creates a record in the **Agency Message Table** indicating that the update has been sent to the collection agency.

This activity can be only be triggered from a collections process.

### Write Off

If all avenues of collections have failed, then the outstanding amount is written off. The Write Off activity creates a transaction to write-off the amount of un-collectable revenue. This amount of un-collectable revenue is all the revenue for that customer is past due. The receivable status is updated to "UNCOLLECTIBLE".

### Input Parameters

The Write Off activity uses the following input parameters:

- **WRITEOFFREASON:** The value inserted into the Account FME table by the Write Off Account function. See **Write Off Account** on page 8-14 for more information.
- **COLLECTIONSTATUS:** The value to update Collection Status field in the Account FME table.



## Output Values

The Write Off activity returns the following output values:

- **WRITEOFFAMOUNT:** The amount written off of the account by the Write Off Account function. See **Write Off Account** on page 8-14 for more information.
- **WRITEOFFDATE:** The date the amount was written off.

## Processing

This activity also makes a note in the **Collection History Table** with the amount of write-off, date and reason. This data is stored in the database and viewed by clicking the **Details** link on the **Collection History** pane of the **Viewing Collections Data** on the Oracle Utilities Receivables Component user interface.

This activity can be triggered from either a collections process or from the Oracle Utilities Rules Language. See the **Post Writeoff Statement** on page D-23 for more information about triggering this activity from the Rules Language.

## Collection Information

The Collection Information activity gathers collections-related information for a specified account.

## Input Parameters

The Collection Information activity uses the following input parameters:

- **UIDACCOUNT:** The UID for the account for which you wish to gather information.

## Output Values

The Collection Information activity returns the following output values:

- **COLLECTIONSTATUS:** The account's current Collection Status, from the Account FME table.
- **RECEIVABLESTATUS:** The account's current Receivable Status, from the Account FME table.
- **TOTALAMOUNTDUE:** The account's current Total Amount Due, from the Account FME table.
- **PASTDUEAMOUNT:** The account's current Past Due Amount, from the Transaction table (charges - credits).
- **TOTALEXEMPTAMOUNT:** The sum of all collections exemptions, from the Collections Exemption table, or "FULLYEXEMPT" if the account is fully exempt.
- **LASTPAYMENTDATE:** The date of the last payment posted to the Payment table.
- **LASTPAYMENTAMOUNT:** The amount of the last payment posted to the Payment table.
- **COLLECTABLEBALANCE:** The Past Due balance minus the TOTALEXEMPTAMOUNT.

## Processing

This activity can be only be triggered from a collections process.

## Insert Collection History Record

The Insert Collection History Record activity inserts a record in the Collection History Table.

## Input Parameters

The Insert Collection History activity uses the following input parameters:

- **UIDACCOUNT:** The UID for the account used to gather account information. Required if ACCOUNTID is not provided.
- **ACCOUNTID:** The ID for the account used to gather account information. Required if UIDACCOUNT not provided.
- **TYPE:** The type of activity being recorded.
- **NAME:** The name of the activity being recorded.
- **SOURCE:** The source of the activity's execution.
- **MESSAGE:** Data provided by the activity's execution.

### Output Values

The Insert Collection History activity returns the following output values:

- **STATUS:** The status of the activity (PASS or ERROR with error details).

### Processing

This activity can be only be triggered from a collections process.

### Update Account Statuses

The Update Account Statuses activity updates the Collections Status and/or Receivable Status fields on the Account FME Table.

### Input Parameters

The Update Account Statuses activity uses the following input parameters:

- **UIDACCOUNT:** The UID for the account used to gather account information.
- **UPDATETYPE:** The type of update. Valid values are COLLECTIONS, RECEIVABLE, or BOTH.
- **RECEIVABLESTATUS:** The new Receivable Status for the account.
- **COLLECTIONSTATUS:** The new Collections Status for the account.

### Output Values

The Update Account Statuses activity returns the following output values:

- **STATUS:** The status of the update activity (PASS or ERROR with error details).

### Processing

**Note:** If the UPDATETYPE is BOTH but only one element is provided, the second element will be set to Null. Also, if the UPDATETYPE is COLLECTIONSTATUS but no COLLECTIONSTATUS element is present, the Collection Status column in the Account FME table will be set to Null. Likewise, if the UPDATETYPE is RECEIVABLESTATUS but no RECEIVABLESTATUS element is present, the Receivable Status column in the Account FME table will be set to Null.

This activity can be only be triggered from a collections process.

### Post Activity Event

The Post Activity Event activity posts an event to the Workflow engine. The event posted must be defined in the EVENTTYPE table. This activity can be used to force an event to be posted to the Workflow engine after another item (such as a workflow start, suspend or resume item) has been posted. This guarantees order of processing.

**Input Parameters**

The Post Activity Event activity uses the following input parameters:

- **UIDACCOUNT:** The UID for the account used to gather account information. (Required).
- **EVENT:** The EVENTTYPECODE to be posted to the Workflow engine (Required).

**Output Values**

The Post Activity Event activity returns the following output values:

- **STATUS:** The status of the activity (PASS or ERROR with error details).

**Processing**

This activity can be triggered from either a collections process or from the Oracle Utilities Rules Language. See the **Process Event Statement** on page E-11 in the *Oracle Utilities Billing Component - Workflow Management Installation and Configuration Guide* for more information about triggering this activity from the Rules Language.

**Collection Activities Payload Information**

Several of the collections activities that involve contacting customers, such as Letter Generation, Phone Contact, Collection Agency Notifications, and Service Discontinuation Requests create a message payload (in XML format) that contains customer, account, and contact information required for the message. Unless otherwise noted, all data is provided for all activities, and includes the following:

- COLLECTIONTRANSACTION (the root element that contains the transaction)
- UIDPROCESSINSTANCE (the UID of the process instance that generated the message)
- UIDACTIVITYINSTANCE (the UID of the activity instance that generated the message)
- MESSAGESUBTYPE (the type of message created)
- CUSTOMER (customer information, including the following):
  - CUSTOMERID
  - NAME
  - CUSTOMERCONTACT (customer contact information, including the following):
    - TYPE
    - NAME
    - ADDRESS1
    - ADDRESS2
    - ADDRESS3
    - CITY
    - STATE
    - ZIP
    - COUNTY
    - COUNTRY
    - FAX
    - PHONE
    - EMAIL
    - PAGER

- ACCOUNT (account information, including the following):
  - ACCOUNTID
  - NAME
  - OPCOCODE (operating company)
  - JURIS (jurisdiction)
  - ACCOUNTCONTACT (account contact information, including the following):
    - TYPE
    - NAME
    - ADDRESS1
    - ADDRESS2
    - ADDRESS3
    - CITY
    - STATE
    - ZIP
    - COUNTY
    - COUNTRY
    - FAX
    - PHONE
    - EMAIL
    - PAGER
  - ACCOUNTMANAGER (account manager information, including the following):
    - NAME
    - EMAIL
    - PHONE
    - FAX
  - SHUTOFFDATE
  - COLLECTIONAGENCY (collection agency)
  - PROGRAMNAME (collection arrangement information (Letter Generation only), including the following):
    - STATUS
    - STARTDATE
    - STOPDATE
    - NUMPAYMENTS (including dates and amounts for each payment)
    - TOTALAMOUNT
- INVOICEDetails (invoice information (Letter Generation only), including the following):
  - INVOICE (individual account invoices, including the following):
    - INVOICEID
    - INVOICEDATE

- 
- INVOICEDUEDATE
  - STATEMENTDATE
  - TRANSACTIONID
  - CHARGEORCREDIT
  - AMOUNT
  - BALANCE
  - DUEDATE
  - SERVICEPLANINFO (service plan information (Letter Generation and Service Discontinuation Request only), including the following):
    - SERVICETYPECODE
    - LDCACCOUNTNO
    - STARTDATE
    - STOPDATE
    - ADDRESS1
    - ADDRESS2
    - ADDRESS3
    - CITY
    - STATE
    - ZIP
    - COUNTY
    - COUNTRY
    - METERINFO (meter information, including the following):
      - METERID
      - NUMBEROFDIALS
      - OFFSET
      - MULTIPLIER
      - SERIALNUMBER
      - UNINUMBER
      - MANUFACTURER
  - PAYMENT (payment information (Letter Generation only), including the following):
    - RECEIVEDATE (last payment date)
    - AMOUNT (last payment amount)
  - DEPOSITINFO (deposit information (Letter Generation only), including the following):
    - DEPOSITAMOUNT
  - BALANCEINFO (balance information, including the following):
    - TOTALAMOUNTDUE
    - PASTDUEBALANCE
    - COLLECTIBLEBALANCE (Past Due - Exemptions, etc.)

In addition to the above information, activity payloads can also contain dynamic payload information. See descriptions of the **Dynamic Message Table** and the **Dynamic Message Value Table** on page 9-4 for more information.

### Example Activity Payload

```
<COLLECTIONTRANSACTION UIDPROCESSINSTANCE="822" UIDACTIVITYINSTANCE="4186"
MESSAGESTYPE="">
  <CUSTOMER>
    <CUSTOMERID>SCENARIO_2</CUSTOMERID>
    <NAME>Scenario 2</NAME>
  </CUSTOMER>
  <ACCOUNT>
    <ACCOUNTID>SCENARIO_2</ACCOUNTID>
    <NAME>Scenario 2</NAME>
    <OPCODE>SESCO</OPCODE>
    <JURISCODE>AGL</JURISCODE>
    <SHUTOFFDATE>1/20/2004</SHUTOFFDATE>
    <COLLECTIONSAGENCY>CARLOS' AGENCY</COLLECTIONSAGENCY>
    <PROGRAMNAME>AGL</PROGRAMNAME>
  </ACCOUNT>
  <INVOICEDetails>
    <INVOICE>
      <INVOICEID>SCENARIO_2</INVOICEID>
      <INVOICEDATE>3/1/2003</INVOICEDATE>
      <INVOICEDUEDATE>6/1/2003</INVOICEDUEDATE>
      <STATEMENTDATE>3/1/2003</STATEMENTDATE>
      <TRANSACTIONID>BILL</TRANSACTIONID>
      <CHARGEORCREDIT>CH</CHARGEORCREDIT>
      <AMOUNT>100</AMOUNT>
      <BALANCE>100</BALANCE>
      <DUEDATE>6/1/2003</DUEDATE>
    </INVOICE>
    <INVOICE>
      <INVOICEID>SCENARIO_2</INVOICEID>
      <INVOICEDATE>3/1/2003</INVOICEDATE>
      <INVOICEDUEDATE>6/1/2003</INVOICEDUEDATE>
      <STATEMENTDATE>3/1/2003</STATEMENTDATE>
      <TRANSACTIONID>BILL</TRANSACTIONID>
      <CHARGEORCREDIT>CH</CHARGEORCREDIT>
      <AMOUNT>125</AMOUNT>
      <BALANCE>125</BALANCE>
      <DUEDATE>6/1/2003</DUEDATE>
    </INVOICE>
  </INVOICEDetails>
  <SERVICEPLANINFO>
    <SERVICETYPECODE>GAS</SERVICETYPECODE>
    <LDCACCOUNTNO></LDCACCOUNTNO>
    <STARTDATE>1/1/2008</STARTDATE>
    <STOPDATE>1/1/2009</STOPDATE>
    <ADDRESS1>123 Main Street</ADDRESS1>
    <ADDRESS2></ADDRESS2>
    <ADDRESS3></ADDRESS3>
    <CITY>Houston</CITY>
    <STATE>TX</STATE>
    <ZIP>77003</ZIP>
    <COUNTY></COUNTY>
    <COUNTRY></COUNTRY>
  <METERINFO>
    <METERID>2</METERID>
    <NUMBEROFDIALS>1</NUMBEROFDIALS>
    <OFFSET>1</OFFSET>
    <MULTIPLIER>1</MULTIPLIER>
    <SERIALNUMBER></SERIALNUMBER>
    <UNINUMBER></UNINUMBER>
    <MANUFACTURER></MANUFACTURER>
  </METERINFO>
</SERVICEPLANINFO>
```

```
<PAYMENT>
  <RECEIVEDDATE>1/3/2003</RECEIVEDDATE>
  <AMOUNT>100</AMOUNT></PAYMENT>
<DEPOSITINFO>
  <DEPOSITAMOUNT>100</DEPOSITAMOUNT>
</DEPOSITINFO>
<BALANCEINFO>
  <TOTALAMOUNTDUE>225</TOTALAMOUNTDUE>
  <PASTDUEBALANCE>225</PASTDUEBALANCE>
  <COLLECTIBLEBALANCE>225</COLLECTIBLEBALANCE>
</BALANCEINFO>
</COLLECTIONTRANSACTION>
```





# Chapter 10

---

## Reports

Reporting is an integral part of the overall functionality of an accounting system, and the Oracle Utilities Receivables Component is no exception. Reporting provides for the confirmation of successful processing and allows the business community the ability to have visibility into the overall performance of operations. This visibility allows the organization to monitor processing, track revenue and receivables, and apply work effort or resources to areas that require additional attention.

This chapter provides an overview of the reporting functionality of Oracle Utilities Receivables Component, including a description of a number of specific reports available through Oracle Utilities Receivables Component, and how those reports are generated. Information on viewing reports is provided in **Chapter 20: Viewing Oracle Utilities Receivables Component Reports** in the *Oracle Utilities Billing Component User's Guide*.

## Report Database Tables

The Reporting functionality of Oracle Utilities Receivables Component use two specific tables, each described below.

### Report List Table

Records in the Report List Table represent the different types of reports available in the system. For each report type, the Report List Table stores the following information:

- **Report ID:** Used to identify the report type,
- **Report Name:** The name of the report type (Payment Posting, Account Balance, etc.),
- **Description:** A description of the report, and
- **Category:** The category of reports to which the report is associated.

### Report Table

Records in the Report Table represent individual reports run within the system. For each report processed by Oracle Utilities Receivables Component, the Report Table stores the following information:

- **UID Report:** A unique ID for the report,
- **Report ID:** The appropriate report ID, from the Report Lists Table,
- **User ID:** ID of user running the report,
- **Start/Stop Time:** Start and stop time of report,
- **Rpt Start Date:** Start date of the report,
- **Rpt Stop Date:** Stop date of the report,
- **Status:** Status of the report (successful/unsuccessful), and
- **Filename:** The file name of the report.

## Oracle Utilities Receivables Component Reports

The Oracle Utilities Receivables Component includes the ability to process and view a number of reports, including:

- Payment Posting,
- Issue Refund,
- Account Balance,
- AR Aging,
- G/L Activity, and
- System Balance.

### Report Format

Each of the reports described below are outlined in the following manner:

- **Description:** A brief description of the purpose of the report.
- **Database:** A listing of the specific database tables and columns used when generating the report, the selection criteria used by each report, and the XML document types (if applicable) and data elements returned by the report.
- **Sample Report:** A sample of the report.

## Payment Posting Report

### Description

The Payment Posting report tracks the number of payments processed each night. This report is broken down by payment file processed, and it is assumed that each source of payment will be processed in separate files.


### Database

**Tables and Columns:** The Payment Posting report is based on data in the following tables and columns and selection criteria.

Table	Columns
Batch Payment Table	Payment File
	Payment Source
	Payment Method
	Batch No.
	Number of Payments
	Amount

**Selection Criteria:** Where the Received Date on the Batch Payment record equals a given date.

### Sample Report - Payment Posting Report

		<b>Lodestar Corp</b> <i>Financial Management Extension</i>		
<b>Report ID:</b> FME-001		<b>Run Date:</b> 6/12/2002 12:17:01 PM		
<b>Report Name:</b> Payment Posting		<b>Report Period:</b> 5/29/2002		
<b>File</b> K:\Shell\ShellQAData\And_Cons\Before_Data\Remittance\RM-ASST-PYMNT-FILE-EUR				
<b>Source</b>	<b>Method</b>	<b>Batch #</b>	<b>Number of Payments</b>	<b>Total Amount</b>
CHASE	PAYMENTMETHODCODE	RM-ASST-PYMNT-BAT CH-EUR	7	€930.00
			<hr/>	<hr/>
			7	€930.00
<hr/>				
<b>File</b> K:\Shell\ShellQAData\And_Cons\Before_Data\Remittance\RM-ASST-PYMNT-FILE-GBP				
<b>Source</b>	<b>Method</b>	<b>Batch #</b>	<b>Number of Payments</b>	<b>Total Amount</b>
CHASE	PAYMENTMETHODCODE	RM-ASST-PYMNT-BAT CH-GBP	5	£635.50
			<hr/>	<hr/>
			5	£635.50
<hr/>				

## Issue Refund Report

### Description

The Issue Refund report details the refunds issued by the system by refund type. This report lists the number of refunds and the total amount of refunds processed by the Issue Refund function of the Oracle Utilities Receivables Component user interface.

### Database


**Tables and Columns:** The Issue Refund report is based on data in the following tables and columns and selection criteria.

Table	Columns
Account Table	Account ID
Address Table	Name
	Address1
	Address2
	Address3
	City
	State
	ZIP Code
Transaction Table	Amount

**Selection Criteria:** Where the Transaction Type Code on the Transaction record equals 'RFND', the Transaction Time on the Transaction record is a given date, and the Cancel Time is Null.

**XML Data Elements:** Account ID, Name, Address1, Address2, Address3, City, State, ZIP Code, and Amount

### Sample Report - Issue Refund Report

		<b>Lodestar Corp</b> <i>Financial Management Extension</i>		
<b>Report ID:</b> FME-002	<b>Run Date:</b> 6/12/2002 12:21:08 PM			
<b>Report Name:</b> Issue Refund	<b>Report Period:</b> 5/30/2002			
Account ID	Account Name	Address	UserID	Amount
Currency Code: EUR				
MN-RFND-3103-82631	Pat Hutchins	101 Oak Ave	dvitale	C28.12
		Pittsburgh, PA 15215		
MN-RFND-3103-82634	Mark Savoy		dvitale	C110.00
MN-RFND-3103-82638	Lionel Vaughn		dvitale	C250.00
MN-RFND-3103-82640	Isabel Almeida	28 Oak Ave	dvitale	C306.05
		Stoughton, MA 02048		
MN-RFND-3103-82642	Barbara Lowry		dvitale	C50.99
MN-RFND-3103-82645	Irving Matassa		dvitale	C156.00
MN-RFND-3103-82655	Kenneth Langley		dvitale	C30.00
MN-RFND-3103-82657	Ed Seltzer		dvitale	C37.50
				<b>C968.66</b>
Currency Code: GBP				
MN-RFND-3103-82632	Loretta Nolan	103 Oak Ave	dvitale	£28.00
		Clinton, IN 47842		
MN-RFND-3103-82633	Maureen Archibald		dvitale	£107.50
MN-RFND-3103-82639			dvitale	£110.75
MN-RFND-3103-82641	WILLIAM KEADY	3872 Morganza Road	dvitale	£98.00
		Bridgeville, PA 15017		
MN-RFND-3103-82646	Martha English		dvitale	£124.80
MN-RFND-3103-82658	Ronald Fitzgerald		dvitale	£107.50

## Account Balance Report

### Description

Each night, a particular segment of the bill accounts are checked to ensure they are “in balance” (the term “balance” refers to the charges, credits, and payments being captured on the database). The Account Balance report details the number of accounts balanced and the number of accounts, along with necessary details, that were “out-of-balance”.

### Database

**Tables and Columns:** The Account Balance report is based on data in the following tables and columns and selection criteria.

Table	Columns
Control Message Table	N/A
Control Message Data Table	XML Data Chunk
Work Queue Message Table	N/A
Work Queue Message Data Table	XML Data Chunk

**Selection Criteria:** Where the Posted Time on the Control Message record equals a given date (not including the time) and the Message Type Code equals ‘ACCOUNTS\_BALANCED’.


**Sub-Selection Criteria:** Where the Posted Time on the Work Queue Message record equals the Posted Time from the Control Message record and the Message Type Code from the Work Queue Message record equals ‘ACCOUNT\_BALANCE\_ERROR’.

**XML Document Types:** ACCOUNTS\_BALANCED and BALANCE\_ACCOUNTS\_ERROR

**XML Data Elements:** Number of Accounts Balanced, Number of Account Balance Errors, and Account ID



# Sample Report - Account Balance Report



**Lodestar Corp**  
Financial Management Extension

**Report ID:** FME-003      **Run Date:** 6/12/2002 12:23:06 PM  
**Report Name:** Bill Account Control Balancing      **Report Period:** 5/29/2002

**Accounts Balanced** 769      **In Balance** 741  
**Out of Balance** 28

**Accounts Out of Balance**

Currency Code: GBP

Account #	Message
MN-RFND-3103-82639	LSACCT6045: Account is out of balance: LSACCT6043: Calculated last balance of 0 GBP is not equal to saved last balance of 22 GBP.

Currency Code:

Account #	Message
999999999	LSACCT6045: Account is out of balance: LSACCT6042: Account activity is out of balance: LSACCT6040: Calculated activity of 0 USD is not equal to saved activity of -4393.615 USD.
GUI-ADJ-3037-81488	LSACCT6045: Account is out of balance: LSACCT6044: Calculated current balance of 1.25 USD is not equal to saved current balance of -1.25 USD.
GUI-ADJ-3037-81736	LSACCT6045: Account is out of balance: LSACCT6042: Account activity is out of balance: LSACCT6040: Calculated activity of -390 USD is not equal to saved activity of 476.25 USD.
GUI-ADJ-3037-81749	LSACCT6045: Account is out of balance: LSACCT6042: Account activity is out of balance: LSACCT6040: Calculated activity of -390 USD is not equal to saved activity of 406.25 USD.
GUI-DEP-3072-81881	LSACCT6045: Account is out of balance: LSACCT6042: Account activity is out of balance: LSACCT6040: Calculated activity of 466.25 USD is not equal to saved activity of 476.25 USD.

## AR Aging Report

### Description

The AR Aging report details aging for accounts receivables in the database. Accounts receivables represents those sales that have not yet been collected, and aging refers to the classification of receivable items into different groups by the amount of time that the item is outstanding. The AR Aging report format breaks down AR items by Market, then by Rate.


### Database

**Tables and Columns:** The AR Aging report is based on data in the following tables and columns and selection criteria.

Table	Columns
Account Table	Balance
Revenue Code Table	Revenue Name
Transaction Table	Operating Company
	Rate Code

**Selection Criteria:** Where the Balance on the Account is not zero and the Charge or Credit value on the related Transaction record equals 'CH'

# Sample Report - AR Aging Report

		<b>Lodestar Corp</b> <i>Financial Management Extension</i>					
<b>Report ID:</b> FME-004 <b>Report Name:</b> A/R Aging		<b>Run Date:</b> 6/12/2002 12:25:30 PM <b>Report Period:</b> 6/12/2002					
Currency Code: EUR Jurisdiction Code: <b>Camden</b>							
<b>Industrial</b>							
<b>Rate</b>	<b>&lt;=30</b>	<b>30-45</b>	<b>45-60</b>	<b>60-90</b>	<b>90-120</b>	<b>120-180</b>	<b>180+</b>
	€0.00	€0.00	€0.00	€0.00	€0.00	€0.00	€320.00
<b>Revenue Total</b>	€0.00	€0.00	€0.00	€0.00	€0.00	€0.00	€320.00
<b>Market Total</b>	€0.00	€0.00	€0.00	€0.00	€0.00	€0.00	€320.00
Jurisdiction Code: <b>FL</b>							
<b>Industrial</b>							
<b>Rate</b>	<b>&lt;=30</b>	<b>30-45</b>	<b>45-60</b>	<b>60-90</b>	<b>90-120</b>	<b>120-180</b>	<b>180+</b>
	€0.00	€0.00	€0.00	€0.00	€0.00	€0.00	€705.00
<b>Revenue Total</b>	€0.00	€0.00	€0.00	€0.00	€0.00	€0.00	€1,025.00
<b>Market Total</b>	€0.00	€0.00	€0.00	€0.00	€0.00	€0.00	€1,345.00
Jurisdiction Code: <b>GA</b>							
<b>Industrial</b>							
<b>Rate</b>	<b>&lt;=30</b>	<b>30-45</b>	<b>45-60</b>	<b>60-90</b>	<b>90-120</b>	<b>120-180</b>	<b>180+</b>
	€0.00	€0.00	€0.00	€0.00	€0.00	€0.00	€1,780.00
<b>Revenue Total</b>	€0.00	€0.00	€0.00	€0.00	€0.00	€0.00	€2,805.00
<b>Market Total</b>	€0.00	€0.00	€0.00	€0.00	€0.00	€0.00	€4,150.00

## G/L Activity Report

### Description

At scheduled intervals, the records in the Journal Transaction Table (p. 2-10) are balanced against records in the Transaction Table (p. 2-5) and the Credit Application Table (p. 2-8). The G/L Activity report details this process. This report displays the total number of financial records balanced, and sum of the amounts for these transactions. It also displays the total number of journal transactions that resulted, and the sum of the amounts for these transactions.

### Database

**Tables and Columns:** The G/L Activity report is based on data in the following tables and columns and selection criteria.


Table	Columns
Control Message Table	N/A
Control Message Data Table	XML Data Chunk

**Selection Criteria:** Where the Posted Time in the Control Message record equals a given date (not including the time) and the Message Type Code in the Control Message record equals 'JOURNAL\_BALANCED'.

**XML Document Types:** JOURNAL\_BALANCED

**XML Data Elements:** Journal Account ID, Cost Center, Revenue Month, Transaction Count, Transaction Amount, Journal Count, Journal Amount

Sample Report - G/L Activity Report

		<p style="text-align: right;"><b>Company Name</b> <i>Financial Management Extension</i></p>					
<b>Report ID:</b> FME-005		<b>Run Date:</b> 6/12/2002 5:00:27 PM					
<b>Report Name:</b> G/L Activity		<b>Report Period:</b> 1/1/2002 - 6/12/2002					
BalanceJournal Processed @ 5/30/2002 12:04:37 PM							
Journal Account ID	Journal Account Name	Cost Center	Revenue Month	Transaction Amount	Credit App. Amount	Journal Amount	Difference
100.01E	Treasury Cash	13000	2000-04-01	200.00	0.00	200.00	0.00
100.01E	Treasury Cash	13000	2000-05-01	530.00	0.00	530.00	0.00
100.01E	Treasury Cash	13000	2002-05-01	-125.00	0.00	-125.00	0.00
100.01P	Treasury Cash	13000	2000-04-01	245.50	0.00	245.50	0.00
100.01P	Treasury Cash	13000	2000-05-01	120.00	0.00	120.00	0.00
100.01P	Treasury Cash	13000	2000-06-01	270.00	0.00	270.00	0.00
100.02	Unapplied Credits		2001-01-01	75.00	958.10	1,033.10	0.00
100.02	Unapplied Credits		2001-02-01	0.00	11.50	11.50	0.00
100.02E	Unapplied Credits		2001-01-01	80.00	0.00	80.00	0.00
100.02P	Unapplied Credits		2001-01-01	-65.00	0.00	-65.00	0.00
200.11	Budget Bill Used		2001-01-01	8.75	0.00	8.75	0.00
200.11	Budget Bill Used		2001-02-01	1.75	0.00	1.75	0.00
500.01	Accounts Receivable	1000	2002-05-01	-28.13	103.14	75.01	0.00
500.01	Accounts Receivable	10000	2001-01-01	80.00	0.00	80.00	0.00
500.01	Accounts Receivable	10000	2002-05-01	-93.71	-318.24	-411.95	0.00
500.01	Accounts Receivable	13000	2002-05-01	161.74	-221.50	-59.76	0.00
500.01	Accounts Receivable	15000	2000-12-01	195.00	0.00	195.00	0.00
500.01	Accounts Receivable	15000	2001-04-01	1,545.00	0.00	1,545.00	0.00
500.01	Accounts Receivable	15000	2001-05-01	787.50	0.00	787.50	0.00
500.01	Accounts Receivable	15000	2001-06-01	0.00	0.00	0.00	0.00
500.01	Accounts Receivable		2000-12-01	500.96	0.00	500.96	0.00
500.01	Accounts Receivable		2001-01-01	79,777.42	-958.10	78,819.32	0.00
500.01	Accounts Receivable		2001-02-01	527.00	-11.50	515.50	0.00
500.01	Accounts Receivable		2001-03-01	0.00	0.00	0.00	0.00

## System Balance Report

### Description

The System Balance report compares the Subledger Balances for those Journal Accounts that have a Type of either A/R or DEFERRED A/R with the net of the sum of the debit and credit balances posted to the LSTransaction table. There is a separate line on the report for each Type/ Journal Account ID/Cost Center combination. The Subledger Balance, Transaction Balance and any difference between the two is reported.

### Database

**Tables and Columns:** The System Balance report is based on data in the following tables and columns and selection criteria.

Table	Columns
Journal Account Table	Journal Account ID Journal Account Name
Cost Center Table	Cost Center Name
Subledger Account Table	Running Balance
Transaction Table	Amount
Budget Plan Table	Amount


**Selection Criteria for A/R:** Where the Journal Account Type in the Journal Account record equals 'A/R' or 'DEFERRED A/R'.

**Database Summary for A/R:** From the Transaction Table, where the Cancel Time is null and the Balance is greater than zero, subtract the sum of all credit transactions from the sum of all debit transactions.

**Selection Criteria for Deferred A/R:** Where the Journal Account Type in the Journal Account record equals 'DEFERRED A/R'.

**Database Summary for DEFERRED A/R:** Sum the Variance from the Budget Plan Table.

# Sample Report - System Balance Report

		<b>Lodestar Corp</b> <i>Financial Management Extension</i>			
<b>Report ID:</b> FME-006 <b>Report Name:</b> System Balance		<b>Run Date:</b> 6/12/2002 12:29:37 PM <b>Report Period:</b> 6/12/2002			
Currency Code: EUR					
<b>A/R</b>					
<b>Journal Acct ID</b>	<b>Description</b>	<b>Cost Center</b>	<b>Subledger Balance</b>	<b>Transaction Balance</b>	<b>Difference</b>
500.01E	Accounts Receivable	Cost Center 1	€28.12	€0.00	€28.12
500.01E	Accounts Receivable	Cost Center 10	€556.05	€0.00	€556.05
500.01E	Accounts Receivable	Cost Center 13	€50.99	€0.00	€50.99
500.01E	Accounts Receivable	Cost Center 15	€262.50	€262.50	€0.00
			€897.66	€262.50	
Currency Code: GBP					
<b>A/R</b>					
<b>Journal Acct ID</b>	<b>Description</b>	<b>Cost Center</b>	<b>Subledger Balance</b>	<b>Transaction Balance</b>	<b>Difference</b>
500.01P	Accounts Receivable	Cost Center 1	£28.13	£0.00	£28.13
500.01P	Accounts Receivable	Cost Center 10	£98.00	£0.00	£98.00
500.01P	Accounts Receivable	Cost Center 13	£110.75	£0.00	£110.75
500.01P	Accounts Receivable	Cost Center 15	£525.00	£525.00	£0.00
			£705.63	£525.00	
Currency Code: USD					
<b>A/R</b>					
<b>Journal Acct ID</b>	<b>Description</b>	<b>Cost Center</b>	<b>Subledger Balance</b>	<b>Transaction Balance</b>	<b>Difference</b>
500.01	Accounts Receivable	Cost Center 1	\$8,523.13	\$0.00	\$8,523.13
500.01	Accounts Receivable	Cost Center 10	\$1,409,173.69	\$0.00	\$1,409,173.69

## Market Participant Aging Report

### Description

The Market Participant Aging report details aging for accounts receivables in the database based on individual accounts. Accounts receivables represents those sales that have not yet been collected, and aging refers to the classification of receivable items into different groups by the amount of time that the item is outstanding.

### Database

**Tables and Columns:** The Market Participant Aging report is based on data in the following tables and columns and selection criteria.

Table	Columns
Account Table	Account ID Account Name
Transaction Table	Balance

**Selection Criteria:** Where the balance on the transaction record is greater than zero (BALANCE > 0) and the Cancel Time is NULL

**Selection Criteria for Current Charges:** Where the Charge or Credit value on the transaction record equals 'CH' and the Transaction Time is less than or equal to (TRANSACTIONTIME <=) 30 days.


**Selection Criteria for Current Credits:** Where the Charge or Credit value on the transaction record equals 'CR' and the Transaction Time is less than or equal to (TRANSACTIONTIME <=) 30 days.

**Selection Criteria for Aged Charges:** Where the Charge or Credit value on the transaction record equals 'CH' and the Transaction Time is greater than (TRANSACTIONTIME >) 30 days.

**Selection Criteria for Aged Credits:** Where the Charge or Credit value on the transaction record equals 'CR' and the Transaction Time is greater than (TRANSACTIONTIME >) 30 days.



Sample Report - Market Participant Aging Report

		<b>Lodestar Corp</b> <i>Financial Management Extension</i>				
<b>Report ID:</b> FME-007 <b>Report Name:</b> Market Participant Aging		<b>Run Date:</b> 6/12/2002 12:30:38 PM <b>Report Period:</b> 6/12/2002				
Currency Code: EUR						
Account ID	Account Name	Current Charges Due	Current Credits	Aged ( >30) Charges Due	Aged ( >30) Credits	Net Due
BL-AUTOPAY-3103-82691	Lisa Jones	€0.00	€0.00	€506.25	€0.00	€506.25
BL-AUTOPAY-3103-82692	Michael Campo	€262.50	€0.00	€243.75	€0.00	€506.25
BL-BILL-3103-82607	Eric Fleming	€51.25	€0.00	€0.00	€0.00	€51.25
BL-BILL-3103-82608	Angela Tanner	€51.25	€0.00	€0.00	€0.00	€51.25
BL-CHRGCRDT-3103-82615	Byron Wheeler	€80.00	€0.00	€0.00	€0.00	€80.00
BL-CHRGCRDT-3103-82618	Irvin Seltzer	€100.00	€0.00	€0.00	€0.00	€100.00
BL-CHRGCRDT-3103-82624	Lisa Fabrizio	€50.00	€0.00	€0.00	€0.00	€50.00
BL-CHRGCRDT-3103-82627	Warren Tate	€75.00	€0.00	€0.00	€80.00	(€5.00)
BL-CHRGCRDT-3103-82628	Maria Candido	€65.00	€0.00	€7.00	€0.00	€72.00
BL-STMT-3103-82602	Mildred Gilmore	€76.30	€0.00	€0.00	€25.50	€50.80
CL-3103-82675	Martin Kowalski	€0.00	€0.00	€40.00	€0.00	€40.00
MN-RFND-3103-82635	Frank Brown	€0.00	€0.00	€0.00	€56.25	(€56.25)
MN-RFND-3103-82644	David Gilmore	€0.00	€0.00	€0.00	€23.75	(€23.75)
MN-RFND-3103-82647	Phil Urbano	€0.00	€0.00	€0.00	€2.50	(€2.50)
MN-RFND-3103-82649	Marshall Wexler	€0.00	€0.00	€0.00	€0.13	(€0.13)
MN-RFND-3103-82651	Duncan Montero	€0.00	€0.00	€0.00	€3.09	(€3.09)
MN-RFND-3103-82653	Allan Hannigan	€0.00	€0.00	€0.00	€110.00	(€110.00)
RM-ASST-3103-82659	Robin Trembly	€0.00	€110.00	€310.00	€0.00	€200.00
RM-ASST-3103-82661	Rosalind Jennings	€0.00	€0.00	€215.00	€0.00	€215.00

## Running Oracle Utilities Receivables Component Reports

Oracle Utilities Receivables Component reports are run using the Report Generator batch file (LSREPORTS.EXE). Some of the parameters used by this program are contained in a Configuration File, and others are passed to the program when the batch file executes.

### Configuration File

The Report Generator requires the LSReports.ini configuration file. This configuration file must be in the same directory as the executable, and contains the following information:

- **DSN** - ODBC Data source Name for connecting to database
- **USER** - User Name string for connecting to database
- **PWD** - Password string for connecting to database
- **Qualifier** - the database qualifier used to connect to the database
- **LSRFConnectString** - The connect string for the Oracle Utilities Data Repository, in one of the following formats:

For Oracle databases:

```
Data Source=<data_source>;User
ID=<user_id>;Password=<password>;LSProvider=ODP;
```

where:

<data\_source> is the Oracle TNS Name for the data source, from the TNS\_NAMES.ora file (typically located in the \\<machine>\oracle\network\admin directory)

<user\_id> is the user ID for the database connection

<password> is the password for the supplied user ID.

- **LOGFILE** - filename (including path) for storing logging messages
- **ENHANCEDLOGGING** - turns on informational messages; 0 for False; 1 for True
- **RPTDIR** - directory for temporary storage of reports
- **WaitingTime** - The waiting time in seconds used by LSReports.exe to wait for the LSReportMonitor service to pick up the requested report. If the report is not picked up by the service within the specified waiting time, the LSReports.exe will terminate its process. (Default: 10 seconds).

### Example

An example of the LSReports.ini file is shown below:

```
[General]
DSN=pwrline
USER=userid
PWD=userpassword
Qualifier=pwrline
LSRFConnectString=Data Source=EIPDB;User
ID=pwrline;Password=userpassword;LSProvider=ODP;
LogFile=\\server\reports\error.log
EnhancedLogging=0
RptDir=\\server\reports\reports\
WaitingTime=10
```

## Logging

All informational and error messages generated during report generation are sent to the Windows NT Event log and a text file. The NT Event log can be viewed by using Windows Event Viewer. The text file is the file specified in the **LogFile** parameter in the Configuration File. Informational messages are only posted if the **EnhancedLogging** parameter is set to 1 in the configuration file.

## Report Storage

All reports generated by the Energy Information Platform Reporting Framework are stored in the Oracle Utilities Data Repository. As the reports are being run, temporary copies of the reports are stored in a directory on a file server. This directory is defined by the **RptDir** parameter in the configuration file. Each filename will be saved in the Report Table, and the unique ID for the report is part of the filename.

## Report Generator Syntax

LSREPORTS uses the syntax shown below.

**lsreports** *report start stop*

In actual use, the command must be entered on one line. Also, you must either change to the directory in which the program is stored (typically, \LODESTAR\bin) before entering the command, or specify the path in the command. To view a list of all parameters on-screen, type **lsreports -?** at the command prompt.

Parameter	Description
<i>report</i>	<p>A number indicating the report to run. Must be one of the following:</p> <ul style="list-style-type: none"> <li>• FME-001 - Payment Posting</li> <li>• FME-002 - Issue Refund</li> <li>• FME-003 - Account Balance</li> <li>• FME-004 - AR Aging</li> <li>• FME-005 - G/L Activity</li> <li>• FME-006 - System Balance</li> <li>• FME-007 - Market Participant Aging.</li> </ul> <p>Only the number of the report is needed. For example, to run the Issue Refund report, enter 2.</p>
<i>start</i>	<p>The start date for the report. Can be either an offset or an explicit date.</p> <p>To specify an offset, enter the number of days prior to the current (system) date that you wish to use as the start date of the report. To run the report with a start date equal to the current date, enter 0.</p> <p>To specify an explicit date, enter the start date for the report in the following format:</p> <p>yyyy-mm-dd</p>

---

Parameter	Description
<i>stop</i>	<p>The stop date for the report. Can be either an offset or an explicit date.</p> <p>To specify an offset, enter the number of days prior to the current (system) date that you wish to use as the stop date of the report. To run the report with a stop date equal to the current date, enter 0.</p> <p>To specify an explicit date, enter the stop date for the report in the following format:</p> <p>yyyy-mm-dd</p>

---

### Examples

The following command will run the Account Balance report for the current date:

```
LSREPORTS 3 0 0
```

The following command will run the G/L Activity report for a start date of January 1, 2009 and stop date of January 31, 2009

```
LSREPORTS 5 2009-01-01 2009-01-31
```

## Viewing Oracle Utilities Receivables Component Reports

Viewing Oracle Utilities Receivables Component reports is performed through the Oracle Utilities Billing Component user interface. See **Chapter 20: Viewing Oracle Utilities Receivables Component Reports** in the *Oracle Utilities Billing Component User's Guide* for more information about viewing Oracle Utilities Receivables Component reports.

---

## Creating Custom Reports

The reports described on previous pages of this chapter represent pre-configured reports available through the Oracle Utilities Receivables Component. In addition to these reports, you can also create and integrate custom reports into Oracle Utilities Receivables Component. The steps involved in creating new reports include designing the report, and adding the report to the system. Each of these is described in more detail below.

### Design Reports

The first step is to design the report to be added. Custom reports are created using Oracle BI Publisher. See **Designing Oracle BI Publisher Reports for use with the Reporting Framework** on page 13-21 in the Oracle Utilities Energy Information Platform Configuration Guide for more information about designing BI Publisher reports.

### Add Report to System

The new report is not available to the user until the report has been added to the system. **Adding Oracle BI Publisher Reports to the Energy Information Platform** on page 13-19 for more information about adding BI Publisher reports to the Reporting Framework.



# Chapter 11

---

## Configuring Oracle Utilities Receivables Component Security

This chapter describes how to configure security to work with the Oracle Utilities Receivables Component application, including:

- **Oracle Utilities Receivables Component Security**

# Oracle Utilities Receivables Component Security

This section describes the securable features of Oracle Utilities Receivables Component, including:

- **Financials Features** (includes Workflow Management)
- **Important Notes about Assigning Oracle Utilities Receivables Component Permissions**

## Financials Features

Financials features include the following:

- **Transactions:** Enables the Transactions menu option.
  - **Installment Plans:** Enables the Installment Plans functions and menu option.
  - **Deposits:** Enables the Deposits functions and menu option.
  - **Payments:** Enables the Payments functions and menu option.
  - **Adjustments:** Enables the Adjustments functions and menu option.
    - **Validate Invoice Allow:** Enables validation of invoices on the Post Adjustment screen.
  - **Refunds:** Enables the Refunds functions and menu option.
    - **Allow Transaction Selection:** Enables selection of transactions on the Post Refund screen.
  - **Write Offs:** Enables the Write Offs functions and menu option.
  - **Balance Transfers:** Enables the Balance Transfers functions and menu option.
- **Work Queue:** Enables the Work Queue function and menu option.
- **Controls:** Enables the Controls function and menu option.
- **Run Reports:** Enables the Run Reports menu option on the Financials menu.
- **View Reports List:** Enables the View Reports menu option on the Financials menu.
- **Collections:** Enables the Collections menu option.
  - **Collection Exemptions:** Enables the Collections Exemptions function and menu option.
  - **Collection Arrangements:** Enables the Collections Arrangements function and menu option.
  - **WQ:** Enables the Work Queue function and menu option.
  - **Manual Submission:** Enables the Manual Submission function and menu option.
- **Workflow:** Enables the Workflow menu option.
  - **Activity Types:** Enables the Activity Types menu item and function.
  - **Process Versions:** Enables the Process Versions menu item and function.
  - **Process Controls:** Enables the Process Controls menu item and function.
  - **Work Queue:** Enables the Work Queue menu item and function.



## **Important Notes about Assigning Oracle Utilities Receivables Component Permissions**

By default, the Oracle Utilities Receivables Component features restrict access to all Oracle Utilities Receivables Component functions and screens. To allow users access to Oracle Utilities Receivables Component functionality, create users and groups and enable appropriate permissions for each.



# Part Two

---

## Workflow Management Configuration

Part Two describes configuration of the workflow management functionality of Oracle Utilities Billing Component, and contains the following chapters:

- **Chapter 12: Setting Up Workflow Management Database Tables**
- **Chapter 13: The Workflow Engine**
- **Chapter 14: The Workflow Scheduler**
- **Chapter 15: The Rules Language Engine**

---

# Chapter 12

---

## Setting Up Workflow Management Database Tables

This chapter describes a number of specific tables in the Oracle Utilities Data Repository used by the workflow management functionality of Oracle Utilities Billing Component, including:

- **COM Object Tables**
- **Process Context Tables**

## COM Object Tables

COM object tables define COM objects used by the COMOBJECT activity implementation type, and include:

- **COM Object Type Table**
- **COM Object Table**

### COM Object Type Table

Records in this table define types of COM object used by Workflow Management, and contain the following data:

- **COM Object Type:** The type of COM object.

### COM Object Table

Records in this table define specific COM objects used by Workflow Management, and contain the following data:

- **DLL Name:** The name of the Dynamic Link Library (DLL) that contains the COM object.
- **Interface Name:** The name of the method (function)
- **Object Name:** The name of COM object.
- **Weight:** *Used with Oracle Utilities Receivables Component and Collections only.* The weight given to activities that use this COM object when calculating a customer's performance score.
- **Duration:** *Used with Oracle Utilities Receivables Component and Collections only.* The duration given to activities that use this COM object when calculating a customer's performance score.
- **Display Name:** The name of the COM object that appears on the Activity Implementation screen of the Workflow Management user interface.
- **COM Object Type:** The COM object type (from the **COM Object Type Table**).

## Process Context Tables

Process context tables define process context values required for processes run by Workflow Management. These are used when a process is initiated from an external system via the Process Context Population function of the Collections interface, and include:

- **Variable Source Table**
- **Context Value Table**
- **Process Context Value Table**

### Variable Source Table

Records in this table define variable sources used by context values, and contain the following data:

- **Variable Source Code:** The type of variable source.
- **Variable Source Code Description:** A description of the variable source.

### Context Value Table

Records in this table define individual context variables required by a process context, and contain the following data:

- **Name:** The name of the context variable.
- **Source:** The type of context variable (from the **Variable Source Table**).
- **Value:** The value of the context variable.

### Process Context Value Table

Records in this table define a process context and its related context variables, and contain the following data:

- **Process:** The name of the process context.
- **Variable Name:** The name of a context variable contained in the process context (from the **Context Value Table**).





# Chapter 13

---

## The Workflow Engine

This chapter describes the Workflow Engine used by the workflow management functionality of Oracle Utilities Billing Component, including:

- **Workflow Engine Functions**
- **Workflow Engine Components**
- **Workflow Engine Processing**
- **Workflow Function Activities**

## Workflow Engine Functions

The workflow management functionality of Oracle Utilities Billing Component provides a number of functions for working with process and activity instances. These functions are initiated from an interface of some sort, and performed by the Workflow Engine. They require the following data:

- An XML file that contains data source connection data (user id, password, datasource, and qualifier), and
- An XML file that specifies the process/activity instance data itself.

### **Start Process Instance**

This function is used to start a new process instance. See **Start Process** on page 13-6 for more information.

### **Suspend Process Instance**

This function is used to suspend an existing running process. See **Suspend Process** on page 13-8 for more information.

### **Resume Process Instance**

This function is used to resume an existing suspended process. See **Resume Process** on page 13-8 for more information.

### **Terminate Process Instance**

This function is used to terminate an existing running, suspended or in error process. See **Terminate Process** on page 13-8 for more information.

### **Activity Instance Finished**

This function is used to notify Workflow Management that a running activity is finished. See **Activity Finished** on page 13-8 for more information.

### **Activity Instance Expired**

This function is used to notify Workflow Management that a running activity is expired. See **Activity Expired** on page 13-8 for more information.

### **Activity Instance In Error**

This function is used to notify Workflow Management that a running activity is in error. See **Activity In Error** on page 13-9 for more information.

### **Post Activity Event**

This function posts an event related to zero or more existing activity instances.

# Workflow Engine Components

The Workflow Engine is made up of a number of components, including:

- **Workflow Engine API**
- **Workflow Engine Message Queue**
- **Workflow Engine Executable**

Each of these components is described below.

## Workflow Engine API

The Workflow Engine API provides access to the Workflow Engine functions (see **Workflow Engine Functions** on page 13-2) from external systems such as a Customer Relationship Management (CRM) application or other system. See **Chapter 24: Workflow Management Process Instance Interface** for more information about using the Workflow Engine API.

## Workflow Engine Message Queue

The Workflow Engine Message Queue is a message queue used by the Workflow Engine. As messages are posted to this queue, the Workflow Engine Executable (see **Workflow Engine Executable** on page 13-5) monitors the Workflow Engine Message Queue for incoming messages and handles each one at a time.

**Note:** The Workflow Engine Executable must be installed on the same machine as the Workflow Engine Message Queue.

In order for the Workflow Engine Message Queue to function properly, a number of records must be created in the Message Queue, Message Type, and Message Type Queue tables in the Oracle Utilities Data Repository. The specifics of these records are provided below.

### Message Queue Table

The single record in the Message Queue table specifies the code, type, and name of the Workflow Engine Message Queue. These should be as follows:

- **Code:** LSWFENG
- **Type:** MSMQ (for Microsoft Message Queue)
- **Name:** One of the following:
  - <SERVERNAME>\LSWFENG (when using Public Queues)

where:

<SERVERNAME> is the name of the server on which the actual message queue resides. For instance, if the server name is FME3, the message queue name would be FME3\LSWFENG.

  - .\private\$\LSWFENG (when using Private Queues)
- **Server Name:** the name of the server on which the public message queue resides. Note that this refers to the actual message queue, which is not necessarily the same machine where the Data Repository resides. This field is not required when using Private queues.

## Message Type Table

The records in the Message Type table specify message types used by the Workflow Engine. There are four specific message types, which should be as follows:

### Activity Finished

A message of this type is posted whenever the Activity Finished function is triggered.

- **Code:** LSWFENG\_ACTV\_FINISHED
- **Description:** *Optional*
- **File Path:** *Optional*

### Activity In Error

A message of this type is posted whenever the Activity InError function is triggered.

- **Code:** LSWFENG\_ACTV\_INERROR
- **Description:** *Optional*
- **File Path:** *Optional*

### Activity Expired

A message of this type is posted whenever the Activity Expired function is triggered.

- **Code:** LSWFENG\_ACTV\_EXPIRED
- **Description:** *Optional*
- **File Path:** *Optional*

### Navigate Process

A message of this type is posted whenever the Navigate Process function is triggered.

- **Code:** LSWFENG\_NAVIGATE\_PROC
- **Description:** *Optional*
- **File Path:** *Optional*

## Message Type Queue Table

The records in the Message Type Queue table associates specific message types used by the Workflow Engine with the Work Flow Engine Message Queue. These records should be as follows:

### Activity Finished

- **Message Type:** LSWFENG\_ACTV\_FINISHED
- **Message Queue:** LSWFENG

### Activity In Error

- **Message Type:** LSWFENG\_ACTV\_INERROR
- **Message Queue:** LSWFENG

### Activity Expired

- **Message Type:** LSWFENG\_ACTV\_EXPIRED
- **Message Queue:** LSWFENG

### Navigate Process

- **Message Type:** LSWFENG\_NAVIGATE\_PROC
- **Message Queue:** LSWFENG

## Workflow Engine Executable

The Workflow Engine Executable (LSWFENG.EXE) monitors the Workflow Engine message Queue(s) for incoming messages and handles each one at a time.

The Workflow Engine Executable is implemented as a Windows Service. The Workflow Engine Executable must be installed on the same machine as the Workflow Engine Message Queue. When installed, the executable can be configured to run automatically whenever the server is started.

The Workflow Engine Executable uses the following parameters, which must be set during installation or via Windows Service properties in order for the service to run automatically.

Parameter	Description
-d	<i>datasourcename</i> is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\<machine>\oracle\network\admin directory)
-u	<i>userid</i> The userid of the user running the Workflow Engine Executable (most often a System Administrator, but can be any user with access to the datasource).
-p	<i>password</i> The datasource password of the user running the Workflow Engine Executable (most often a System Administrator, but can be any user with access to the datasource).
-q	<i>qualifier</i> is an optional database qualifier.
-o	<i>data provider</i> The database provider, based on the database type. <ul style="list-style-type: none"> <li>Oracle: ODP</li> </ul>
-m	<i>multiple</i> Specifies if the service can support multiple providers (ODP for Oracle databases). Valid values are "True" and "False".

# Workflow Engine Processing

This section describes different processes performed by the Workflow Engine. These processes are triggered by corresponding functions (see **Workflow Engine Functions** on page 13-2).

## Start Process

The Start Process function performs the following steps:

1. The Workflow Engine creates a Process Instance record based on the specified Process Version, sets the Process State to **RUNNING**, and initializes the Process Context as specified. The Workflow Engine also creates a Process Archive record at this time.
2. For each Process Activity in the specified Process Version, the Workflow Engine creates an Activity Instance record, and initializes the Activity State to **INACTIVE**. The Workflow Engine also creates Activity Archive records at this time.
3. If the Activity Type is **EVENT**, the Workflow Engine triggers the **Register Activity Event** function.
4. If the Process is specified to be started at a later time, the Workflow Engine sets the Process State to **SUSPENDED** and triggers the **Schedule Message** function of the Scheduler.
5. If the process is not specified to be started at a later time, the Workflow Engine triggers the **Navigate Process** function.

## Navigate Process

The Navigate Process function performs the following steps.

1. The Workflow Engine locates the process' Start Activities. Start Activities are those activities that have no input path.
2. The Workflow Engine evaluates the Input Path of each activity whose state is **INACTIVE**.  
If the Source Activity is **COMPLETED**, the Workflow Engine evaluates the Input Path Transition Condition against the Process Context.  
If the Source Activity is **SKIPPED**, the Input Path evaluates to **FALSE**.  
If neither of the above are true, the Input Path cannot be evaluated.
3. If all the Input Paths could be evaluated and an activity's Start Condition is **TRUE**, the Workflow Engine triggers the **Start Activity** function.
4. If all the Input Paths could be evaluated and an activity's Start Condition is **FALSE**, the Workflow Engine sets the Activity state to **SKIPPED**.
5. If all the Input Paths cannot be evaluated, the Workflow Engine moves on to the next **INACTIVE** activity in the process.
6. The Workflow Engine locates all End Activities. End Activities are those activities that are not the Source Activity for any input paths in the Process. If all the End Activities' states are either **COMPLETED** or **SKIPPED**, the Workflow Engine sets the Process State to **COMPLETED** and updates the State and Stop Time of the Process Instance and Process Archive records.

## Start Activity

The Start Activity function performs the following steps:

1. The Workflow Engine sets the Activity State to RUNNING, and maps the Process Context to the Activity Context.
2. If the Activity Type is MESSAGE, the Workflow Engine triggers the Post Message function of the Message interface.
3. If the Activity Type is EVENT, the Workflow Engine triggers the **Wait Activity Event** function.
4. If the Activity Type is PROCESS, the Workflow Engine triggers the **Start Process Instance** function.
5. If the Activity Type is RATEFORM, the Workflow Engine posts a message to the Rules Language Execution Engine Message Queue.
6. The Workflow Engine updates the State and Start Time of the Activity Instance and Activity Archive records.

## Terminate Activity

The Terminate Activity function performs the following steps:

1. The Workflow Engine sets the Activity State to TERMINATED.
2. If the Activity Type is COMOBJECT, the Workflow Engine cancels the function called by the COM object.
3. If the Activity Type is EVENT, the Workflow Engine triggers the **Clear Activity Event** function.
4. If the Activity Type is MESSAGE, the Workflow Engine triggers the **Remove Message** function of the Message interface.
5. If the Activity Type is PROCESS, the Workflow Engine triggers the **Terminate Process Instance** function.
6. RATE FORM activities can only be terminated by stopping the Rules Language Execution Engine service.
7. If the Activity Type is RESUME, the Workflow Engine triggers the **Resume Process Instance** function.
8. If the Activity Type is SUSPEND, the Workflow Engine triggers the **Suspend Process Instance** function.
9. If the Activity Type is TERMINATE, the Workflow Engine triggers the **Terminate Process Instance** function.
10. If the Activity Type is WAIT, the Workflow Engine triggers the **Wait Activity Event** function.
11. If the Activity Type is END, the Workflow Engine triggers the **Activity Instance Finished** function.
12. The Workflow Engine updates the State and Stop Time of the Activity Instance and Activity Archive records.

## Suspend Process

The Suspend Activity function performs the following steps:

1. If the Process State is RUNNING, the Workflow Engine set the Process State to SUSPENDED.
2. The Workflow Engine updates the State and Stop Time of the Process Instance and Process Archive records.

## Resume Process

The Resume Activity function performs the following steps:

1. If the Process State is SUSPENDED, the Workflow Engine set the Process State to RUNNING and triggers the Navigate Process function.
2. The Workflow Engine updates the State and Start Time of the Process Instance and Process Archive records.

## Terminate Process

The Terminate Process function performs the following steps:

1. For each Activity Instance with an Activity State of RUNNING, the Workflow Engine triggers the Terminate Activity function, and sets the Process State to TERMINATED and updates the State and Stop Time of the Process Instance and Process Archive records.

## Activity Finished

The Activity Finished function performs the following steps:

1. If the Process State and Activity State are both RUNNING, the Workflow Engine sets the Activity State to FINISHED and updates the Activity Context as specified.
2. If an Expiration Time exists for the Activity Instance, the Workflow Engine triggers the **Cancel Message** function of the Scheduler.
3. The Workflow Engine evaluates the Activity Exit Condition against the Activity Context.
  - If Activity Exit Condition evaluates to TRUE, the Workflow Engine sets the Activity State to COMPLETED, maps the Activity Context to the Process Context, updates the State and Stop time of the Activity Instance and Activity Archive records, and triggers the Navigate Process function
  - If the Activity Exit Condition evaluates to FALSE, the Workflow Engine triggers the **Start Activity** function.

## Activity Expired

The Activity Expired function performs the following steps:

1. If the Process State and Activity State are both RUNNING, the Workflow Engine sets the Activity State to EXPIRED.
2. If the Activity Type is MESSAGE, the Workflow Engine triggers the **Remove Message** function of the Message interface.
3. If the Activity Type is EVENT, the Workflow Engine triggers the **Clear Activity Event** function.
4. If the Activity Type is PROCESS, the Workflow Engine triggers the **Terminate Process Instance** function.



5. RATE FORM activities can only be terminated by stopping the Rules Language Execution Engine service.
6. The Workflow Engine evaluates the Exit/Error condition against the Activity Context.
  - If the Exit/Error condition is EXIT, the Workflow Engine sets the Activity State to COMPLETED, sets the Stop Time on the Activity Instance and Activity Archive records, and triggers the Navigate Process function.
  - If Exit/Error condition is ERROR, the Workflow Engine sets the Activity State to INERROR sets the Process State to INERROR, and sets the Stop Time on the Activity Instance and Activity Archive records.

## Activity In Error

The Activity In Error function performs the following steps:

1. If the Process State and Activity State are both RUNNING, the Workflow Engine sets the Activity State to INERROR and sets the Stop Time on the Process Instance and Activity Instance records.
2. If an Expiration Time exists, the Workflow Engine triggers the **Cancel Message** function of the Scheduler, and sets the Process State to INERROR.

## Activity Event

There four specific functions related to activity events. The processing involved with each is described below.

### Register Activity Event

The Register Activity Event function performs the following steps:

1. The Workflow Engine inserts an Activity Event record in the Activity Event table.
2. The Workflow Engine initializes the Activity Event Waiting Flag to false

### Post Activity Event

The Post Activity Event function performs the following steps:

1. The Workflow Engine updates the Activity Event Time and Data as specified.
2. If the Activity Event Waiting Flag is true, the Workflow Engine triggers the **Activity Instance Finished** function, and deletes the Activity Event record

### Wait Activity Event

The Wait Activity Event function performs the following steps:

1. If the Activity Event has already been posted, the Workflow Engine triggers the **Activity Instance Finished** function and deletes the Activity Event record.
2. If the Activity Event has not been posted, the Workflow Engine sets the Activity Event Waiting Flag to TRUE.

### Clear Activity Event

The Clear Activity Event function performs the following steps:

1. The Workflow Engine deletes the Activity Event record

## Workflow Function Activities

Several of the Workflow Engine functions are also available activities in workflow management processes. This section describes these functions, and provides details regarding input parameters for each. The input parameters listed below are the context values used for each activity.

### Suspend

The Suspend activity suspends an existing running process. See **Suspend Process** on page 13-8 for more information.

#### Input Parameters

The Suspend activity uses the following input parameters:

- **UIDPROCSUSPEND:** The UID of the Process Instance to be suspended. Optionally, you can provide “THIS” to suspend the current process (Required).

### Resume

The Resume activity resumes an existing suspended process. See **Resume Process** on page 13-8 for more information.

#### Input Parameters

The Resume activity uses the following input parameters:

- **UIDPROCRESUME:** The UID of the Process Instance to be resumed (Required).

### Terminate

The Terminate activity terminates an existing running, suspended or in error process. See **Terminate Process** on page 13-8 for more information.

#### Input Parameters

The Terminate activity uses the following input parameters:

- **UIDPROCSTOP:** The UID of the Process to be terminated. Optionally, you can provide “THIS” to terminate the current process (Required).
- **NOTE:** A note that describes the reason for terminating the process (Required).

### Wait

The Wait activity causes a process to wait for a specified time before continuing.

#### Input Parameters

The Wait activity uses the following input parameters:

- **WAITTYPE:** Indicates a Factor in the Oracle Utilities Data Repository that stores the number of days the activity is to wait before proceeding to the next activity.
- **USEBUSINESSDAY:** Indicates if the business calendar is to be used when calculating the date on which the Wait activity is to complete. For example, if the activity is to wait four days but the fourth day falls on a Saturday, the activity uses the next valid business day. Valid values are “TRUE” and “FALSE”. Defaults to “TRUE”.
- **RELATIVE:** Indicates if the activity uses a relative date or absolute date when calculating the date on which the Wait activity is to complete. Valid values are “TRUE”, “FALSE”, or “PREVSTART”. TRUE means use a relative date. FALSE means use an absolute date. PREVSTART means the Wait time is calculated as a relative date based on the previous activity's start date. If this value is not supplied it will be based on the current activity's (the Wait activity) start date. If RELATIVE is “TRUE” or “PREVSTART”, either WAITTYPE or EXPTIME must be present.

- **EXPTIME:** An integer representing the number of days the activity is to wait. This is used instead of the WAITTYPE value.

**Note:** Only either EXPTIME and WAITTYPE is required. If both are provided, the EXPTIME value takes precedence.

## End

The End activity ends a process.

### Input Parameters

The End activity uses the following input parameters:

- **UIDPROCSTOP:** The UID of the Process to be ended (Required).
- **NOTE:** An optional Reason for ending the process.



# Chapter 14

---

## The Workflow Scheduler

This chapter describes the Scheduler used by the workflow management functionality of Oracle Utilities Billing Component, including:

- **Workflow Scheduler Functions and Processing**
- **Workflow Scheduler Components**

## Workflow Scheduler Functions and Processing

The workflow management functionality of Oracle Utilities Billing Component provides a pair of functions for working with scheduled messages. Each requires the following data:

- An XML file that contains data source connection data (user id, password, datasource, and qualifier), and
- An XML file that specifies the message data itself.

### Schedule Message

This function is used to schedule a single message.

A scheduled message contains the following data:

- A valid message type code from the Message Type table.
- A scheduled time. This contains an absolute or relative time for the message. Relative times are indicated by the RELATIVE attribute while absolute time use a regular time format. The default value of the RELATIVE attribute is FALSE. The relative time value must follow the following format: YYYY-MM-DD HH:MM:SS where YYYY is the number of years, MM is the number of months, and DD is the number of days in YYYY-MM-DD; HH is the number of hours, MM is the number of minutes, and SS is the number of seconds in HH:MM:SS. The relative time will be converted to absolute time from now (the current time) by the scheduler. The USEBUSINESSDAYS attribute specifies whether to use business day dates configured in the database or not. The default value of the USEBUSINESSDAYS attribute is FALSE. When this attribute is true, the Scheduler gets the nth business day that is specified in YYYY-MM-DD HH:MM:SS from the BUSINESSDAY and BUSINESSCALENDAR tables.
- An XML string containing the message data.
- Optional data including the Source, Operating Company, Jurisdiction, Account ID, Amount, and Note.

### Cancel Message

This function is used to cancel a single scheduled message from the database.

# Workflow Scheduler Components

The Workflow Scheduler is made up of a number of components, including:

- **Workflow Scheduler API**
- **Workflow Scheduler Executable**

## Workflow Scheduler API

The Workflow Engine API provides access to the Scheduler functions (see **Workflow Scheduler Functions and Processing** on page 14-2) from external systems such as a Customer Relationship Management (CRM) application or other system.

## Workflow Scheduler Executable

The Workflow Scheduler Executable (LSSCHDLR.EXE) monitors the Scheduled Message table in the Oracle Utilities Data Repository. The frequency of monitoring the database can be configured as needed (the default frequency is 15 minutes). As it monitors the Scheduled Message table, for each scheduled message with a scheduled time less than or equal to ( $\leq$ ) the current time, the Scheduler executable posts the message to the appropriate message queue and then deletes it from the database.

The Workflow Scheduler Executable is implemented as a Windows Service. When installed, the executable can be configured to run automatically whenever the server is started.

The Workflow Scheduler Executable uses the following parameters, which must be set during installation or via Windows Service properties in order for the service to run automatically.

Parameter	Description
-d	<i>datasourcename</i> is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\<machine>\oracle\network\admin directory)
-u	<i>userid</i> The userid of the user running the Workflow Engine Executable (most often a System Administrator, but can be any user with access to the datasource).
-p	<i>password</i> The datasource password of the user running the Workflow Engine Executable (most often a System Administrator, but can be any user with access to the datasource).
-q	<i>qualifier</i> is an optional database qualifier.
-o	<i>data provider</i> The database provider, based on the database type. <ul style="list-style-type: none"> <li>• Oracle: ODP</li> </ul>
-m	<i>monitortime</i> is monitoring frequency, in number of milliseconds. The default value is 15 minutes (900,000).





# Chapter 15

---

## The Rules Language Engine

This chapter describes the Rules Language Engine used by the workflow management functionality of Oracle Utilities Billing Component, including:

- **Rate Form Activities and the Rules Language Engine**
- **Rules Language Engine Components**
- **Creating Rate Forms for use with the Rules Language Engine**

## Rate Form Activities and the Rules Language Engine

This section provides a high-level overview of how the Rules Language Engine processes rate forms.

1. When a Rate Form activity is started (via **Start Activity** on page 13-7), the Workflow Engine posts a message in the Rules Language Engine message queue (LSRLENG).

This message contains the following data:

- The Rate Form code of the Rate Form to be processed, from the Rate Form table. If the rate form is associated to a specific Operating Company and Jurisdiction, the appropriate Operating Company and Jurisdiction codes precede the Rate Form code, separated by colons.
  - Optional message data in XML format. This is the context of the rate form activity that posted the message.
  - An optional account ID and/or note.
2. The Rules Language Engine picks up the message in the LSRLENG message queue, and processes the rate form.

If the message contains an Account ID, the Rules Language can use that AccountID in processing.

If the message contains XML data, that data is passed as input to the Rules Language Engine as a string. This input can be loaded in the Rules Language by using the following statement:

```
CONTEXT_DOC = DOMDOCLOADXML (RATE_SCHEDULE_INPUT_XML);
```

Once the XML data is loaded, the Rules Language can be used to get Element Text Node Values and Attributes from the XML data elements, which can in turn be used in Rules Language processing. The Rules Language can also create output XML data from the results of processing. See **Creating Rate Forms for use with the Rules Language Engine** on page 15-5 for more information about how to design rate forms suitable for processing by the Rules Language Execution Engine. Refer to **Appendix F: XML Rules Language Statements and Functions** for more information about the **DOMDOCLOADXML** function and about manipulating XML data using the Rules Language.

3. After running the rate form, the results (along with output XML if applicable) will be posted back to the Workflow Engine message queue.

# Rules Language Engine Components

The Rules Language Engine is made up of a pair of components, including:

- **Rules Language Engine Message Queue**
- **Rules Language Execution Engine Executable**

## Rules Language Engine Message Queue

The Rules Language Engine Message Queue is an MSMQ (Microsoft Message Queue) message queue used by the Rules Language Engine. The Rules Language Engine Executable monitors the Rules Language Engine Message Queue for incoming messages and as messages are posted to this queue, it handles each one at a time.

**Note:** The Rules Language Engine Executable must be installed on the same machine as the Rules Language Engine Message Queue.

In order for the Rules Language Engine Message Queue to function properly, a number of records must be created in the Message Queue, Message Type, and Message Type Queue tables in the Oracle Utilities Data Repository. The specifics of these records are provided below.

### Message Queue Table

A record in the Message Queue table specifies the code, type, and name of the Rules Language Engine Message Queue. These should be as follows:

- **Code:** LSRLENG
- **Type:** MSMQ (for Microsoft Message Queue)
- **Name:** <SERVERNAME>\LSRLENG

Where:

<SERVERNAME> is the name of the server on which the actual message queue resides. For instance, if the server name is FME3, the message queue name would be FME3\LSWFENG. Note that “the server on which the message queue resides” refers to the actual message queue which is not necessarily the same machine where the Data Repository resides.

### Message Type Table

A record in the Message Type table defines a specific message type used by the Rules Language Execution Engine. This record should be as follows:

- **Code:** LSRLENG\_RUN
- **Description:** *Optional*
- **File Path:** *Optional*

### Message Type Queue Table

A record in the Message Type Queue table associates the specific message type used by the Rules Language Engine with the Rules Language Engine Message Queue. This record should be as follows:

- **Message Type:** LSRLENG\_RUN
- **Message Queue:** LSRLENG

## Rules Language Execution Engine Executable

The Rules Language Engine Executable (LSRLENG.EXE) monitors the Rules Language Engine message Queue for incoming messages and handles each one at a time.

The Rules Language Engine Executable is implemented as a Windows Service. The Rules Language Engine Executable must be installed on the same machine as the Rules Language Engine Message Queue. When installed, the executable can be configured to run automatically whenever the server is started.

The Rules Language Engine Executable uses the following parameters, which must be set during installation or via Windows Service properties in order for the service to run automatically.

Parameter	Description
-d	<i>datasourcename</i> is the Oracle TNS Name for the data source, from the TNS_NAMES.ora file (typically located in the \\<machine>\oracle\network\admin directory)
-u	<i>userid</i> The userid of the user running the Rules Language Engine Executable (most often a System Administrator, but can be any user with access to the datasource).
-p	<i>password</i> The datasource password of the user running the Rules Language Engine Executable (most often a System Administrator, but can be any user with access to the datasource).
-q	<i>qualifier</i> is an optional database qualifier.
-o	<i>data provider</i> The database provider, based on the database type. <ul style="list-style-type: none"> <li>Oracle: ODP</li> </ul>
-f	<i>configfilename</i> is the name of the configuration file that defines the working environment of the Rule Language Engine and other Oracle Utilities software (e.g., directs the software where to find and place the application data files and so on). If you do not supply a value for <i>configfilename</i> , the system uses the default (LODESTAR.CFG). If the configuration file is located in a different directory than LSRLENG.EXE, you must specify a path to the file. If specifying a path in the Services Control Panel, use double back-slashes in the path. For example, "d: \\LODESTAR\\CFG\\LODESTAR.CFG".  For information about the contents of this configuration file, please refer to the <i>Oracle Utilities Energy Information Platform Installation and Configuration Guide</i> .
-m	<i>multiple</i> Specifies if the service can support multiple providers (ODP for Oracle databases). Valid values are "True" and "False".

## Creating Rate Forms for use with the Rules Language Engine

This section describes how to create rate forms that can be processed by the Rules Language Engine.

Because the Rules Language Engine can only obtain input data from messages posted by the Workflow Engine, rate forms that require input data not defined in the activity context of the rate form activity that initiated the message cannot be processed. For instance, the Rules Language Engine could not process a rate form that requires a specific customer ID unless that customer ID is defined in the activity context.

There are two types of rate forms suitable for use with the Rules Language Engine:

- **No Required Input Data**
- **Input Data from Context**

### No Required Input Data

Rate forms that don't require input data are those that obtain the data they require for processing from the Oracle Utilities Data Repository, usually through use of table.column list or other database identifiers (see **Chapter Four: Identifiers, Constants, and Expressions** in the *Rules Language User's Guide*).

Rate forms of this type include those that use the Workflow Management statements (see **Appendix E: Workflow Management Rules Language Statements**), but might also include settlement schedules used by the Oracle Utilities Load Profiling and Settlement and transaction processing schedules used by the Oracle Utilities Transaction Management.

### Input Data from Context

The more common type of rate forms processed by the Rules Language Engine are those that can obtain required input data from the activity context. Rate forms of this type must be designed to properly extract the required input data from the context data.

Context data is passed as input to the Rules Language Engine as a string. Before it can be used by the rate form, this input must be loaded into an XML structure that the Rules Language can then extract the required data from. When the context string is passed to the Rules Language Execution Engine, it is assigned as the value of RATE\_SCHEDULE\_INPUT\_XML, a pre-defined identifier. The context string can be loaded in the Rules Language by using the following statement:

```
CONTEXT_DOC = DOMDOCLOADXML (RATE_SCHEDULE_INPUT_XML);
```

This statement uses the **DOMDOCLOADXML Function** on page F-15 to set the value of the CONTEXT\_DOC identifier to an XML structure derived from the context string.

Once the XML data is loaded, Rules Language statements and functions are used to extract Element Text Node Values and Attributes from the XML structure and assign them to identifiers. These identifiers can in turn be used in Rules Language processing, the results of which can be saved back to the XML structure for use as output data, loaded into RATE\_SCHEDULE\_INPUT\_XML, a pre-defined identifier.

## Example

The following simple example demonstrates how data can be extracted from saved back to an XML structure. Assuming the activity context was as follows:

```
<CONTEXT>
  <ACCOUNTID>123</ACCOUNTID>
  <PASTDUEAMT>90.00</PASTDUEAMT>
  <OTHER />
</CONTEXT>
```

The following statements could be used to load the XML structure and extract the Account ID.

```
\* Define the Context Structure *\
IDENTIFIER = CONTEXT
XML_ELEMENT ACCOUNTID NODENAME "ACCOUNTID" PARENT CONTEXT
XML_ELEMENT PAST_DUE NODENAME "PASTDUEAMT" PARENT CONTEXT
XML_ELEMENT OTHER NODENAME "OTHER" PARENT CONTEXT
\* Load the XML document *\
CONTEXT_DOC = DOMDOCLOADXML (RATE_SCHEDULE_INPUT_XML);
\* Obtain the Root Element *\
DOC_ROOT = DOMDOCGETROOT (CONTEXT_DOC)
\* Get the Account ID *\
ACCT_ID = ACCOUNTID.NODEVALUE;
...
```

The following statement could be used to set the value of the Arrangement attribute of the OTHER element.

```
\* Set the Arrangement Attribute of the OTHER element *\
OTHER.ARRANGEMENT = "TRUE";
...
```

Once all the data processing has been performed and any required data saved back to the XML structure, the following statement can be used to load the XML structure into the RATE\_SCHEDULE\_INPUT\_XML identifier.

```
CONTEXT_DOC = RATE_SCHEDULE_OUTPUT_XML;
```

When the Rules Language Engine finishes processing the rate form and posts a message in the Workflow Engine Message Queue, the context would look like this:

```
<CONTEXT>
  <ACCOUNTID>123</ACCOUNTID>
  <PASTDUEAMT>90.00</PASTDUEAMT>
  <OTHER Arrangement='TRUE' />
</CONTEXT>
```

Refer to **Appendix E: Workflow Management Rules Language Statements** for more information about the **DOMDOCLOADXML** function and about manipulating XML data using the Rules Language.

# Part Three

---

## Receivables Management Interfaces

Part Three describes the COM interfaces available with the receivables management functionality of Oracle Utilities Billing Component, and contains the following chapters:

- **Chapter 16: Oracle Utilities Receivables Component Financial Engine Interface**
- **Chapter 17: Oracle Utilities Receivables Component Billing Interface**
- **Chapter 18: Oracle Utilities Receivables Component Remittance Interface**
- **Chapter 19: Oracle Utilities Receivables Component Maintenance Interface**
- **Chapter 20: Oracle Utilities Receivables Component Collections Interface**
- **Chapter 21: Messaging Interface**

---



# Chapter 16

---

## Oracle Utilities Receivables Component Financial Engine Interface

This chapter describes the methods/functions available to external systems through the Financial Management Interface (IAREngine). These methods allow users to perform a number of financial management functions available in the Oracle Utilities Receivables Component from external systems. These functions include the following:

- Sub-Ledger Roll-up
- General Ledger Update
- Balance Journal
- Balance Accounts.

See **Chapter 5: The Financial Engine** for more information about these functions.

## Methods, Interfaces, and Syntax

The method names, interface objects, and syntax for the available methods of the Financial Management interface are as follows:

### Sub-Ledger Roll-Up

**Description:** Used to roll up records in the Journal Transaction Table to their corresponding sub-ledger account records in the Sub-Ledger Account Table.

**Method Name:** UpdateSubledger

**Interface:** IAREngine

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.AREngine

**Syntax:**

```
HRESULT UpdateSubledger ([in] BSTR xmlDataSource);
```

### General Ledger Update

**Description:** Used to outputs records from the Sub-ledger Account table to update an external general ledger system.

**Method Name:** UpdateGenLedger

**Interface:** IAREngine

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.AREngine

**Syntax:**

```
HRESULT UpdateGenLedger ([in] BSTR xmlDataSource;  
                        ([in] BSTR xmlGenLedgerParams;  
                        ([out] BSTR xmlGenLedgerFile);
```

### Balance Journal

**Description:** Used to balance records in the Journal Transaction Table against records in the Transaction and Credit Application tables.

**Method Name:** BalanceJournal

**Interface:** IAREngine

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.AREngine

**Syntax:**

```
HRESULT BalanceJournal ([in] BSTR xmlDataSource,  
                       [in] BSTR xmlJournalParams,  
                       [in, optional] BSTR xmlJournalDataSource);
```

### Balance Accounts

**Description:** Used to balance records in the Transaction Table for a specified subset of accounts.

**Method Name:** BalanceAccounts

**Interface:** IAREngine

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.AREngine

**Syntax:**

```
HRESULT UpdateSubledger ([in] BSTR xmlDataSource,  
                          [in] BSTR xmlQuery,
```

## Interface Arguments

The methods available in the Financial Management interface use the following arguments:

### **xmlDataSource Argument**

The xmlDataSource argument is an xml string that contains database connection and other related information. A DTD, xml example, and data element descriptions for this argument can be found on page 15-7 in the *Oracle Utilities Energy Information Platform Configuration Guide*.

### **xmlAccount Argument**

The xmlAccount argument is an xml string that contains the information necessary to identify an account and the date for which to get the information. A DTD, xml example, and data element descriptions for this argument can be found on page 16-4.

### **xmlTransaction Argument**

The xmlTransaction argument is an xml string that contains transaction data. A DTD, xml example, and data element descriptions for this argument can be found on page 16-6.

### **xmlGenLedgerParams Argument**

The xmlGenLedgerParams argument is an xml string that specifies the parameters needed to create the General Ledger File. It is used with the Update General Ledger method. A DTD, xml example, and data element descriptions for this argument can be found on page 16-12.

### **xmlGenLedgerFile Argument**

The xmlGenLedgerFile argument is an output xml string containing the general ledger information accumulated from the sub-ledger accounts. It is created by the Update General Ledger method. A DTD, xml example, and data element descriptions for this argument can be found on page 16-14.

### **xmlJournalParams Argument**

The xmlJournalParams argument is an xml string that specifies the parameters over which to balance journal transactions. It is used with the Balance Journal method. A DTD, xml example, and data element descriptions for this argument can be found on page 16-13.

### **xmlJournalDataSource Argument**

The xmlJournalDataSource argument is an xml string that contains database connection and other related information. This optional argument is identical to the xmlDataSource argument (described on 15-7 in the *Oracle Utilities Energy Information Platform Configuration Guide*), but is used when the Journal Transaction Table is located in a different database than the financial transaction and other Oracle Utilities Data Repository tables.

### **xmlQuery Argument**

The xmlQuery argument is an xml string that specifies a SQL query used by the account balancing functionality of the Oracle Utilities Receivables Component. It is used with the Balance Accounts method. A DTD, xml example, and data element descriptions for this argument can be found on page 16-13.

## Input Values

The Data Type Definition (DTD), an xml example, and data element descriptions used as input values for the Financial Management interface (IAREngine) are provided below.

### xmlAccount

#### DTD - xmlAccount

```
<!DOCTYPE ACCOUNT
[
<!ELEMENT ACCOUNT (CUSTOMER?, STARTTIME?, STOPTIME?, OPCOCODE?,
JURISCODE?, REGIONCODE?, SIC?, STATUSCODE?, REVENUECODE?,
LASTBALANCETIME?, LASTBALANCE?, NEWACTIVITY?, UIDTXFRTOACCT?,
RECEIVABLESTATUS?, WRITEOFFREASONCODE?, BALANCEDATE?, CURRENTBALANCE?,
PASTDUEBALANCE?, LASTTXACTIONNO?)>
<!ATTLIST ACCOUNT
    UIDCDATA          #IMPLIED
    ACCOUNTIDCDATA   #IMPLIED
    UIDSUBACCOUNTCDATA#IMPLIED
    SUBACCOUNTIDCDATA#IMPLIED
    NAMECDATA        #IMPLIED>
<!ELEMENT CUSTOMER EMPTY>
<!ATTLIST CUSTOMER
    UIDCDATA          #IMPLIED>
<!ELEMENT STARTTIME (#PCDATA)>
<!ELEMENT STOPTIME (#PCDATA)>
<!ELEMENT OPCOCODE (#PCDATA)>
<!ELEMENT JURISCODE (#PCDATA)>
<!ELEMENT REGIONCODE (#PCDATA)>
<!ELEMENT SIC (#PCDATA)>
<!ELEMENT STATUSCODE (#PCDATA)>
<!ELEMENT REVENUECODE (#PCDATA)>
<!ELEMENT LASTBALANCETIME (#PCDATA)>
<!ELEMENT LASTBALANCE (#PCDATA)>
<!ELEMENT NEWACTIVITY (#PCDATA)>
<!ELEMENT UIDTXFRTOACCT (#PCDATA)>
<!ELEMENT RECEIVABLESTATUS (#PCDATA)>
<!ELEMENT WRITEOFFREASONCODE (#PCDATA)>
<!ELEMENT BALANCEDATE (#PCDATA)>
<!ELEMENT CURRENTBALANCE (#PCDATA)>
<!ELEMENT PASTDUEBALANCE (#PCDATA)>
<!ELEMENT LASTTXACTIONNO (#PCDATA)>
]>
```

#### XML Example - xmlAccount

```
<ACCOUNT UID="221" ACCOUNTID="FE-81486-CANCEL-81462">
  <CUSTOMER UID="428"/>
  <STARTTIME>1999-01-01-T00:00:00</STARTTIME>
  <OPCODE>SESCO</OPCODE>
  <JURISCODE>TX</JURISCODE>
  <REGIONCODE>CENTRAL</REGIONCODE>
  <REVENUECODE>30</REVENUECODE>
</ACCOUNT>
```

## Element Descriptions - xmlAccount

The use of each individual attribute and element in the xmlAccount argument is described below.

Account Attributes:

**UID:** Unique identifier for account.

**ACCOUNTID:** Alternate unique identifier for account.

**UIDSUBACCOUNT:** Optional Unique ID for the sub-account for which this account is the master.

**SUBACCOUNTID:** Optional ID for the sub-account for which this account is the master.

**NAME:** Optional name of account.

Elements:

**CUSTOMER:** Parent customer of account.

Attributes:

**UID:** Unique identifier for customer.

**STARTTIME:** Effective start time for account.

**STOPTIME:** Optional stop time for closed account.

**OPCODE:** Optional operating company code for account.

**JURISCODE:** Optional jurisdiction code for account.

**REGIONCODE:** Optional region code for account.

**SIC:** Optional SIC code for account.

**STATUSCODE:** Optional status code for account.

**REVENUECODE:** Optional revenue code for account.

**LASTBALANCETIME:** Time that account was last balanced.

**LASTBALANCE:** Balance of account at the last balance time.

Attributes:

**CURRENCY:** Currency code for the balance.

**NEWACTIVITY:** Activity for account since the last balance time.

Attributes:

**CURRENCY:** Currency code for the new activity.

**UIDTXFRTOACCT:** Optional UID of account to which this account has been transferred.

**RECEIVABLESTATUS:** Receivable status for account. May be one of "CURRENT", "PASTDUE", "COLLECTIONS", "UNCOLLECTIBLE", or "UNREFUNDABLE".

**WRITEOFFREASONCODE:** Optional write-off reason for account if Receivable Status is "UNCOLLECTIBLE".

**BALANCEDATE:** Date for current and past due balance calculations.

**CURRENTBALANCE:** Current balance as of balance date for account.

Attributes:

**CURRENCY:** Currency code for the current balance.

**PASTDUEBALANCE:** Past due balance as of balance date for account.

Attributes:

**CURRENCY:** Currency code for the past due balance.

**LASTTXACTIONNO:** Last transaction number for account.

## xmlTransaction

### DTD - xmlTransaction

```
<!DOCTYPE TRANSACTION
[
<!ELEMENT TRANSACTION (ACCOUNT, TRANSACTIONTYPE?, TRANSACTIONID?,
TRANSACTIONTIME?, REVENUEMONTH?, USERID?, NOTE?, DEBITACCTID?,
CREDITACCTID?, COSTCENTERID?, CANCELTIME?, CANCELREVENUEMONTH?,
CANCELUSERID?, CANCELREASONCODE?, CANCELNOTE?, CANCELDEBITACCTID?,
CANCELCREDITACCTID?, CANCELCOSTCENTERID?, CHARGEORCREDIT?, AMOUNT,
BALANCE?, BILLEDORPAIDDATE?, DUEDATE?, RECEIVABLETYPE?, CHARGETYPE?,
RATECODE?, OPCOCODE?, JURISCODE?, STATEMENTDATE?, INVOICEID?,
INVOICEDATE?, BILLCYCLEDATE?, CANCELSTATEMENTDATE?, CANCELINVOICEID?,
CANCELINVOICEDATE?, BILLCYCLEDATE?, BILLHISTORY?, UIDSUBACCOUNT?,
TRANSFERREDTOTRANSACTION?, TRANSFERREDFROMTRANSACTION?, PAYMENT?,
SERVICEPLAN?, BUDGETPLAN?, INSTALLMENTPLAN?, DEPOSIT?, DEPINTRATE?,
TAXAMOUNT?, TAXRATE?, TAXEXEMPT?, BILLSTARTTIME?, BILLSTOPTIME?,
SUSPENDAUTOPAYMENT?, BATCHTRANSACTION?, BATCHCANCEL?,
RELATEDTRANSACTIONS?, USERATTRS?>>
<!ATTLIST TRANSACTION
    UIDCDATA #IMPLIED
    TRANSACTIONNOCDATA#IMPLIED>
    APPLICATIONMETHOD (DEFERRED|
        IMMEDIATE|
        SPECIFIED|
        INVOICEID|
        RECEIVABLETYPE) "DEFERRED">
    DEFERBALANCE (TRUE|FALSE) "FALSE">
<!ELEMENT ACCOUNT (see above)>
<!ELEMENT TRANSACTIONTYPE (#PCDATA)>
<!ELEMENT TRANSACTIONID (#PCDATA)>
<!ELEMENT TRANSACTIONTIME (#PCDATA)>
<!ELEMENT REVENUEMONTH (#PCDATA)>
<!ELEMENT USERID (#PCDATA)>
<!ELEMENT NOTE (#PCDATA)>
<!ELEMENT DEBITACCTID (#PCDATA)>
<!ELEMENT CREDITACCTID (#PCDATA)>
<!ELEMENT COSTCENTERID (#PCDATA)>
<!ELEMENT CANCELTIME (#PCDATA)>
<!ELEMENT CANCELREVENUEMONTH (#PCDATA)>
<!ELEMENT CANCELUSERID (#PCDATA)>
<!ELEMENT CANCELREASONCODE (#PCDATA)>
<!ELEMENT CANCELNOTE (#PCDATA)>
<!ELEMENT CANCELDEBITACCTID (#PCDATA)>
<!ELEMENT CANCELCREDITACCTID (#PCDATA)>
<!ELEMENT CANCELCOSTCENTERID (#PCDATA)>
<!ELEMENT CHARGEORCREDIT (#PCDATA)>
<!ELEMENT AMOUNT (#PCDATA)>
<!ELEMENT BALANCE (#PCDATA)>
<!ELEMENT BILLEDORPAIDDATE (#PCDATA)>
<!ELEMENT DUEDATE (#PCDATA)>
<!ELEMENT RECEIVABLETYPE (DESCRIPTION)>
<!ATTLIST RECEIVABLETYPE
    UIDCDATA #IMPLIED
    NAMECDATA #IMPLIED>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT CHARGETYPE EMPTY>
```

```

<!ATTLIST CHARGETYPE
    UIDCDATA          #IMPLIED
    IDENTIFIERCDATA  #IMPLIED
    NAMECDATA        #IMPLIED>
<!ELEMENT RATECODE (RATEFORM)>
<!ATTLIST RATECODE
    CODECDATA          #IMPLIED>
<!ELEMENT RATEFORM EMPTY>
<!ATTLIST RATEFORM
    UIDCDATA          #IMPLIED
<!ELEMENT OPCOCODE (#PCDATA)>
<!ELEMENT JURISCODE (#PCDATA)>
<!ELEMENT STATEMENTDATE (#PCDATA)>
<!ELEMENT INVOICEID (#PCDATA)>
<!ELEMENT INVOICEDATE (#PCDATA)>
<!ELEMENT CANCELSTATEMENTDATE (#PCDATA)>
<!ELEMENT CANCELINVOICEID (#PCDATA)>
<!ELEMENT CANCELINVOICEDATE (#PCDATA)>
<!ELEMENT BILLCYCLEDATE (#PCDATA)>
<!ELEMENT BILLHISTORY EMPTY>
<!ATTLIST BILLHISTORY
    UIDCDATA          #IMPLIED
<!ELEMENT UIDSUBACCOUNT (#PCDATA)>
<!ELEMENT TRANSFERREDTOTRANSACTION EMPTY>
<!ATTLIST TRANSFERREDTOTRANSACTION
    UIDCDATA          #IMPLIED
<!ELEMENT TRANSFERREDFROMTRANSACTION EMPTY>
<!ATTLIST TRANSFERREDFROMTRANSACTION
    UIDCDATA          #IMPLIED
<!ELEMENT PAYMENT EMPTY>
<!ATTLIST PAYMENT
    UIDCDATA          #IMPLIED
<!ELEMENT SERVICEPLAN EMPTY>
<!ATTLIST SERVICEPLAN
    UIDCDATA          #IMPLIED
<!ELEMENT BUDGETPLAN EMPTY>
<!ATTLIST BUDGETPLAN
    UIDCDATA          #IMPLIED
<!ELEMENT INSTALLMENTPLAN (see Chapter Ten: Oracle Utilities
Receivables Component Billing Interface)>
<!ELEMENT DEPOSIT EMPTY>
<!ATTLIST DEPOSIT
    UIDCDATA          #IMPLIED>
<!ELEMENT DEPINTRATE (#PCDATA)>
<!ELEMENT TAXAMOUNT (#PCDATA)>
<!ELEMENT TAXRATE (#PCDATA)>
<!ELEMENT TAXEXEMPT (#PCDATA)>
<!ELEMENT BILLSTARTTIME (#PCDATA)>
<!ELEMENT BILLSTOPTIME (#PCDATA)>
<!ELEMENT SUSPENDAUTOPAYMENT (#PCDATA)>
<!ELEMENT BATCHTRANSACTION EMPTY>
<!ATTLIST BATCHTRANSACTION
    UIDCDATA          #IMPLIED
<!ELEMENT BATCHCANCEL EMPTY>
<!ATTLIST BATCHCANCEL
    UIDCDATA          #IMPLIED
<!ELEMENT RELATEDTRANSACTIONS (TRANSACTION+)>
<!ELEMENT USERATTRS (USERATTR+)>
<!ELEMENT USERATTR EMPTY>
<!ATTLIST USERATTR
    NAMECDATA        #REQUIRED
    TYPE (CHAR|

```

```

                                VARCHAR |
                                TINYINT |
                                SMALLINT |
                                INT |
                                BIGINT |
                                REAL |
                                FLOAT |
                                DECIMAL |
                                DATE |
                                TIMESTAMP) #REQUIRED
SIZECDATA                       #IMPLIED
PRECISIONCDATA                  #IMPLIED
SCALECDATA                      #IMPLIED
VALUECDATA                      #REQUIRED>
]>

```

### XML Example - xmlTransaction

```

<TRANSACTION>
  <ACCOUNT UID="221"/>
  <TRANSACTIONID>300</TRANSACTIONID>
  <TRANSACTIONTIME>1999-01-05-T00:00:00</TRANSACTIONTIME>
  <REVENUEMONTH>01/01/1999</REVENUEMONTH>
  <CHARGEORCREDIT>CH</CHARGEORCREDIT>
  <AMOUNT CURRENCY="USD">30.00</AMOUNT>
  <BALANCE CURRENCY="USD">30.00</BALANCE>
  <BILLEDORPAIDDATE>1999-01-10</BILLEDORPAIDDATE>
  <DUEDATE>1999-01-30</DUEDATE>
  <RECEIVABLETYPE UID="2">
    <EMPTY/>
  </RECEIVABLETYPE>
  <CHARGETYPE UID="177">
    <EMPTY/>
  </CHARGETYPE>
  <OPCODE>UCG</OPCODE>
  <JURISCODE>GA</JURISCODE>
  <SRVICEPLAN UID="167">
    <EMPTY/>
  </SRVICEPLAN>
</TRANSACTION>

```

### Element Descriptions - xmlTransaction

The use of each individual attribute and element in the xmlTransaction argument is described below.

Transaction Attributes:

**UID:** Unique identifier for transaction.

**TRANSACTIONNO:** Unique number for transaction within account.

**APPLICATIONMETHOD:** Indicates credit application method

**DEFERBALANCE:** Indicates whether or not to defer the balance associated with the transaction.

Elements:

**ACCOUNT:** Parent account of transaction.

**TRANSACTIONTYPE:** Transaction Type for transaction. May be inferred from Transaction ID.

**TRANSACTIONID:** Transaction ID for transaction.

**TRANSACTIONTIME:** Time of transaction.



**REVENUEMONTH:** Month for which any journal transactions will apply.

**USERID:** User ID of authenticated user posting transaction.

**NOTE:** Optional note associated with posting transaction.

**DEBITACCTID:** Optional debit journal account ID associated with posting transaction.

**CREDITACCTID:** Optional credit journal account ID associated with posting transaction.

**COSTCENTERID:** Optional cost center ID associated with posting transaction.

**CANCELTIME:** Time that transaction was cancelled.

**CANCELREVENUEMONTH:** Month for which any journal transactions will apply.

**CANCELUSERID:** User ID of authenticated user canceling transaction.

**CANCELREASONCODE:** Optional reason for canceling the transaction.

**CANCELNOTE:** Optional note associated with canceling transaction.

**CANCELDEBITACCTID:** Optional debit journal account ID associated with canceling transaction.

**CANCELCREDITACCTID:** Optional credit journal account ID associated with canceling transaction.

**CANCELCOSTCENTERID:** Optional cost center ID associated with canceling transaction.

**CHARGEORCREDIT:** Indicates whether transaction amount is a charge or a credit.

**AMOUNT:** Amount of transaction.

Attributes:

**CURRENCY:** Currency code for the amount.

**BALANCE:** Remaining (unapplied) balance of transaction.

Attributes:

**CURRENCY:** Currency code for the balance.

**BILLEDORPAIDDATE:** Billed date for charges, paid date for credits.

**DUEDATE:** Due date for charges.

**RECEIVABLETYPE:** Receivable type for transaction.

Attributes:

**UID:** Unique identifier for receivable type.

**NAME:** Unique name of receivable type.

Elements:

**DESCRIPTION:** Optional description for receivable type.

**CHARGETYPE:** Charge type for transaction.

Attributes:

**UID:** Unique identifier for charge type.

**IDENTIFIER:** Alternate unique identifier for charge type.

**NAME:** Unique name of charge type.

**RATECODE:** Associated rate code for charge.

Attributes:

**CODE:** Unique identifier for rate code within parent rate form.

Elements:

**RATEFORM:** Parent rate form of rate code.

Attributes:

**UID:** Unique identifier for rate form.

**OPCOCODE:** Associated operating company code for transaction.

**JURISCODE:** Associated jurisdiction code for transaction.

**STATEMENTDATE:** Associated statement date for transaction.

**INVOICEID:** Associated invoice ID for transaction.

**INVOICEDATE:** Associated invoice date for transaction.

**CANCELSTATEMENTDATE:** Associated statement date for cancelled transaction.

**CANCELINVOICEID:** Associated invoice ID for cancelled transaction.

**CANCELINVOICEDATE:** Associated invoice date for cancelled transaction.

**BILLCYCLEDATE:** Associated cycle date for transaction. Optional for posting, canceling, or transferring.

**BILLHISTORY:** Associated bill history record for transaction.

Attributes:

**UID:** Unique identifier for bill history.

**UIDSUBACCOUNT:** Optional unique identifier of originating subaccount for summary billing.

**TRANSFERREDTOTRANSACTION:** Associated transferred “to” transaction for a transferred “from” transaction.

Attributes:

**UID:** Unique identifier for transferred “to” transaction.

**TRANSFERREDFROMTRANSACTION:** Associated transferred “from” transaction for a transferred “to” transaction.

Attributes:

**UID:** Unique identifier for transferred “from” transaction.

**PAYMENT:** Associated payment record for PYMNT transaction.

Attributes:

**UID:** Unique identifier for payment.

**SERVICEPLAN:** Associated service plan for transaction.

Attributes:

**UID:** Unique identifier for service plan.

**BUDGETPLAN:** Associated budget plan for transaction.

Attributes:

**UID:** Unique identifier for budget plan.

**INSTALLMENTPLAN:** Associated installment plan for deferred charge or INST transaction (see definition in **Chapter Ten: Oracle Utilities Receivables Component Billing Interface**).

**DEPOSIT:** Associated deposit for DEP, DEPINT, DEPAPP, or INST transaction.

Attributes:

**UID:** Unique identifier for deposit.

**DEPINTRATE:** Interest rate on deposit for DEP transaction.

**TAXAMOUNT:** Tax amount for related taxed transaction.

Attributes:

**CURRENCY:** Currency code for the tax amount.

**TAXRATE:** Tax rate for either TAX transaction or related taxed transaction.

**TAXEXEMPT:** Tax exempt flag for related taxed transaction. Should be either "TRUE" or "FALSE". Defaults to false.

**BILLSTARTTIME:** Bill start time for BILL transaction.

**BILLSTOPTIME:** Bill stop time for BILL transaction.

**SUSPENDAUTOPAYMENT:** If "TRUE", indicates that automatic payments for the bill transaction should be suspended. Should be either "TRUE" or "FALSE". Defaults to false.

**BATCHTRANSACTION:** Batch transaction from where this transaction came.

Attributes:

**UID:** Unique identifier for batch transaction. Required for posting and cancelling.

**BATCHCANCEL:** Batch transaction that cancelled this transaction.

Attributes:

**UID:** Unique identifier for batch transaction. Required for posting and cancelling.

**RELATEDTRANSACTIONS:** Related transactions for this transaction. For transfer transactions, this is the list of transferred transactions. For adjustments or refunds, this is the optional list of transactions that are being adjusted or refunded so that directed credit applications can be properly applied.

Elements:

**TRANSACTION:** Individual related transaction.

**USERATTRS:** Collection of user-defined attributes for the transaction.

Elements:

**USERATTR:** Individual user-defined attribute.

Attributes:

**NAME:** Name for the attribute. Should be the actual column name in the database.

**TYPE:** Type of the attribute.

**SIZE:** Optional size of the attribute. Required for CHAR and VARCHAR types.

**PRECISION:** Optional precision of the attribute. Required for DECIMAL types.

**SCALE:** Optional scale of the attribute. Required for DECIMAL types.

**VALUE:** Value of the attribute.

## xmlGenLedgerParams

### DTD - xmlGenLedgerParams

```

<!DOCTYPE GENLEDGERPARAMS
[
<!ELEMENT GENLEDGERPARAMS (GENLEDGERMAP)>
<!ATTLIST GENLEDGERPARAMS
    REVENUEMONTHCDATA#IMPLIED
    CLOSE{TRUE | FALSE}"TRUE">
<!ELEMENT GENLEDGERMAP (GENLEDGERACCOUNT, SUBLEDGERACCOUNT*)>
<!ELEMENT GENLEDGERACCOUNT>
<!ATTLIST GENLEDGERACCOUNT
    ACCOUNTIDCDATA #REQUIRED
    COSTCENTERIDCDATA#IMPLIED>
<!ELEMENT SUBLEDGERACCOUNT>
<!ATTLIST SUBLEDGERACCOUNT
    ACCOUNTIDCDATA #REQUIRED
    COSTCENTERIDCDATA#IMPLIED
    USAGE{TRUE | FALSE}"FALSE">
]>

```

### XML Example - xmlGenLedgerParams

```

<GENLEDGERPARAMS CLOSE="FALSE" REVENUEMONTH="08/2001">
  <GENLEDGERMAP>
    <GENLEDGERACCOUNT ACCOUNTID="acctG1" COSTCENTERID="center1"/>
    <SUBLEDGERACCOUNT ACCOUNTID="acctS1" COSTCENTERID="center1"/>
    <SUBLEDGERACCOUNT ACCOUNTID="acctS2" COSTCENTERID="center1"/>
  </GENLEDGERMAP>
  <GENLEDGERMAP>
    <GENLEDGERACCOUNT ACCOUNTID="acctG2" COSTCENTERID="center2"/>
    <SUBLEDGERACCOUNT ACCOUNTID="acctS3" COSTCENTERID="center1" USAGE="FALSE"/>
  </GENLEDGERMAP>
  <GENLEDGERMAP>
    <GENLEDGERACCOUNT ACCOUNTID="acctG3" COSTCENTERID="center3"/>
    <SUBLEDGERACCOUNT ACCOUNTID="acctS4" COSTCENTERID="center3" USAGE="TRUE"/>
  </GENLEDGERMAP>
</GENLEDGERPARAMS>

```

### Element Descriptions - xmlGenLedgerParams

The use of each individual attribute and element in the xmlGenLedgerParams argument is described below.

Attributes

**CLOSE:** Flag that specifies whether to close the sub-ledger accounts after processing.

**REVENUEMONTH:** Revenue month used to select the sub-ledger account records processed. If not supplied, the prior calendar month/year is used.

Elements:

**GENLEDGERMAP:** If specified, maps one or more sub-ledger accounts to a general ledger account, or maps a sub-ledger account that contains usage information to a general ledger account. If not specified, all sub-ledger accounts are mapped directly to general ledger accounts with the same Account ID and Cost Center ID.

Elements:

**GENLEDGERACCOUNT:** General ledger account

Attributes:

**ACCOUNTID:** Account ID of the general ledger account.

**COSTCENTERID:** Cost center ID associated with the general ledger account.

**SUBLEDGERACCOUNT:**

Attributes:

**ACCOUNTID:** Account ID of the sub-ledger account.

**COSTCENTERID:** Cost center ID associated with the sub-ledger account.

**USAGE:** Specifies whether the sub-ledger account represents quantity information.

## xmlJournalParams

### DTD - xmlJournalParams

```
<!DOCTYPE JOURNALPARAMS
[
<!ELEMENT JOURNALPARAMS>
<!ATTLIST JOURNALPARAMS
    STARTTIMECDATA #REQUIRED
    STOPTIMECDATA #REQUIRED
    REVENUEMONTHCDATA#IMPLIED>
]>
```

### XML Example - xmlJournalParams

```
<JOURNALPARAMS>
  <STARTTIME>BILL_START</STARTTIME>
  <STOPTIME>BILL_STOP</STOPTIME>
  <REVENUEMONTH>11/2000</REVENUEMONTH>
</JOURNALPARAMS>
```

### Element Descriptions - xmlJournalParams

Each of the data elements used by the xmlJournalParams argument is described below.

JournalParams Attributes:

**STARTTIME:** The start time of the period for which the journal is to be balanced.

**STOPTIME:** The stop time of the period for which the journal is to be balanced.

**REVENUEMONTH:** The revenue month for which the journal is to be balanced.

## xmlQuery

### DTD - xmlQuery

```
<!DOCTYPE QUERY
[
<!ELEMENT QUERY (#PCDATA) >
]>
```

### XML - xmlQuery

```
<QUERY>
  SELECT UIDACCOUNT from %%QUAL%%.ACCOUNT where JURISCODE="MA"
</QUERY>
```

### Element Descriptions - xmlQuery

Each of the data elements used by the xmlQuery argument is described below.

**QUERY:** The SQL query used to identify the account to be balanced. The query must have a select list containing only the "UIDACCOUNT" column name.

## Return Values

### xmlGenLedgerFile

#### DTD - xmlGenLedgerFile

```

<!DOCTYPE GENLEDGERFILE
[
<!ELEMENT GENLEDGERFILE (GENLEDGERACCOUNT*)>
<!ATTLIST GENLEDGERFILE
    REVENUEMONTHCDATA#REQUIRED
    CLOSE{TRUE | FALSE}"TRUE">
<!ELEMENT GENLEDGERACCOUNT (
    BALANCE, OLDBALANCETOGL, CURBALANCETOGL, RUNNINGBALANCE, QUANTITY,
    OLDQUANTITYTOGL, CURQUANTITYTOGL, RUNNINGQUANTITY)
<!ATTLIST GENLEDGERACCOUNT
    ACCOUNTIDCDATA #REQUIRED
    COSTCENTERIDCDATA#IMPLIED>
<!ELEMENT BALANCE (#PCDATA)>
<!ELEMENT OLDBALANCETOGL (#PCDATA)>
<!ELEMENT CURBALANCETOGL (#PCDATA)>
<!ELEMENT RUNNINGBALANCE (#PCDATA)>
<!ELEMENT QUANTITY (#PCDATA)>
<!ELEMENT OLDQUANTITYTOGL (#PCDATA)>
<!ELEMENT CURQUANTITYTOGL (#PCDATA)>
<!ELEMENT RUNNINGQUANTITY (#PCDATA)>
]>

```

#### XML Example - xmlGenLedgerFile

```

<GENLEDGERFILE REVENUEMONTH="08/2001">
  <GENLEDGERACCOUNT ACCOUNTID="acctG1" COSTCENTERID="cencter1">
    <BALANCE CURRENCY="USD"></BALANCE>
    <OLDBALANCETOGL CURRENCY="USD"></OLDBALANCETOGL>
    <CURBALANCETOGL CURRENCY="USD"></CURBALANCETOGL>
    <RUNNINGBALANCE CURRENCY="USD"></RUNNINGBALANCE>
    <QUANTITY>5</QUANTITY>
    <OLDQUANTITYTOGL></OLDQUANTITYTOGL>
    <CURQUANTITYTOGL></CURQUANTITYTOGL>
    <RUNNINGQUANTITY></RUNNINGQUANTITY>
  </GENLEDGERACCOUNT>
</GENLEDGERFILE>

```

#### Element Descriptions - xmlGenLedgerFile

The use of each individual attribute and element in the xmlGenLedgerFile argument is described below.

Attributes

**CLOSE:** Flag that indicates

**REVENUEMONTH:** Revenue month for the general ledger update.

Elements:

**GENLEDGERACCOUNT:** General ledger account

Attributes:

**ACCOUNTID:** Account ID of the general ledger account.

**COSTCENTERID:** Cost center ID associated with the general ledger account.

Elements:

---

**BALANCE:** Total of BALANCEs of sub-ledger accounts associated with the general ledger account.

Attributes:

**CURRENCY:** Currency code for the balance.

**OLDBALANCETOGL:** Total BALANCETOGLs of sub-ledger accounts associated with general ledger account.

Attributes:

**CURRENCY:** Currency code for the balance.

**CURBALANCETOGL:** Difference between BALANCE and OLDBALANCETOGL. Represents the amount to be moved to the general ledger account.

Attributes:

**CURRENCY:** Currency code for the balance.

**RUNNINGBALANCE:** Total of RUNNING BALANCEs of sub-ledger accounts associated with the general ledger account.

Attributes:

**CURRENCY:** Currency code for the balance.

**QUANTITY:** Total of QUANTITYs of sub-ledger accounts associated with the general ledger account. Used with sub-ledger accounts that represent quantities rather than amounts.

**OLDQUANTITYTOGL:** Total QUANTITYTOGLs of sub-ledger accounts associated with general ledger account. Used with sub-ledger accounts that represent quantities rather than amounts.

**CURQUANTITYTOGL:** Difference between QUANTITY and OLDQUANTITYTOGL. Represents the amount to be moved to the general ledger account. Used with sub-ledger accounts that represent quantities rather than amounts.

**RUNNINGQUANTITY:** Total of RUNNING QUANTITYs of sub-ledger accounts associated with the general ledger account. Used with sub-ledger accounts that represent quantities rather than amounts.





# Chapter 17

---

## Oracle Utilities Receivables Component Billing Interface

This chapter describes the methods/functions available to external systems through the Oracle Utilities Receivables Component Billing interface (IBilling). These methods allow users to perform a number of billing functions available in the Billing module of the Oracle Utilities Receivables Component to Oracle Utilities Billing Component from external systems. These functions include the following:

- Post/Cancel Charge or Credit
- Post/Cancel Tax
- Create/Cancel Installment Plan
- Post/Cancel Installment
- Post/Cancel Deposit
- Post/Cancel Deposit Interest
- Apply/Unapply Deposit
- Post Statement
- Post Bill
- Cancel Transactions
- Get Bill Info.

See **Chapter 6: Billing** for more information about these functions.

## Methods, Interfaces, and Syntax

The methods, interface objects, and syntax for the functions performed through the Oracle Utilities Receivables Component Billing interface are as follows:

### Post Charge or Credit

**Description:** Used to post either a charge or credit transaction against the specified account (the default is charge). The transaction may be either deferred or not deferred (default is not deferred). An optional service plan or budget plan may be associated with the transaction. If a budget plan is provided then the plan's variance will be updated accordingly.

**Method Name:** PostChargeOrCredit

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT PostChargeOrCredit ([in] BSTR xmlDataSource,  
                             [in] BSTR xmlAcctTrans);
```

### Cancel Charge or Credit

**Description:** Used to cancel a previously posted charge or credit transaction.

**Method Name:** CancelChargeOrCredit

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT CancelChargeOrCredit ([in] BSTR xmlDataSource,  
                               [in] BSTR xmlAcctTrans);
```

### Post Tax

**Description:** Used to post a tax charge or credit transaction against the specified account (the default is charge). The transaction may be either deferred or not deferred (default is not deferred). An optional service plan or budget plan may be associated with the transaction. If a budget plan is provided then the plan's variance will be updated accordingly.

**Method Name:** PostTax

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT PostTax ([in] BSTR xmlDataSource,  
                 [in] BSTR xmlAcctTrans);
```

### Cancel Tax

**Description:** Used to cancel a previously posted tax charge or credit transaction.

**Method Name:** CancelTax

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT CancelTax ([in] BSTR xmlDataSource,
                  [in] BSTR xmlAcctTrans);
```

### Create Installment Plan

**Description:** Used to create an installment plan related to a previously posted deferred charge transaction. This process can alternatively occur as a single transaction if the installment plan is related to the deferred charge when the deferred charge is initially posted.

**Method Name:** CreateInstallmentPlan

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT CreateInstallmentPlan ([in] BSTR xmlDataSource,
                              [in] BSTR xmlInstallmentPlan);
```

### Cancel Installment Plan

**Description:** Used to cancel a previously created installment plan

**Method Name:** CancelInstallmentPlan

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT CancelInstallmentPlan ([in] BSTR xmlDataSource,
                              [in] BSTR xmlInstallmentPlan);
```

### Post Installment

**Description:** Used to post a non-deferred charge transaction related to a previously created installment plan against the specified account.

**Method Name:** PostInstallment

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT PostInstallment ([in] BSTR xmlDataSource,
                        [in] BSTR xmlAcctTrans);
```

### Cancel Installment

**Description:** Used to cancel a previously posted installment transaction.

**Method Name:** CancelInstallment

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT CancelInstallment ([in] BSTR xmlDataSource,  
                           [in] BSTR xmlAcctTrans);
```

## Post Deposit

**Description:** Used to post either a non-deferred or deferred charge transaction against the specified account (the default is non-deferred). A deferred deposit would typically have an installment plan created for it.

**Method Name:** PostDeposit

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT PostDeposit ([in] BSTR xmlDataSource,  
                   [in] BSTR xmlAcctTrans);
```

## Cancel Deposit

**Description:** Used to cancel a previously posted deposit transaction.

**Method Name:** CancelDeposit

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT CancelDeposit ([in] BSTR xmlDataSource,  
                      [in] BSTR xmlAcctTrans);
```

## Post Deposit Interest

**Description:** Used to post a deferred credit transaction against the specified account representing an amount of interest accrual for the associated deposit.

**Method Name:** PostDepositInterest

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT PostDepositInterest ([in] BSTR xmlDataSource,  
                            [in] BSTR xmlAcctTrans);
```

## Cancel Deposit Interest

**Description:** Used to cancel a previously posted deposit interest transaction.

**Method Name:** CancelDepositInterest

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT CancelDepositInterest ([in] BSTR xmlDataSource,
                               [in] BSTR xmlAcctTrans);
```

### Apply Deposit

**Description:** Used to post a non-deferred credit transaction against the specified account representing an amount of the associated deposit balance that is applied to the account.

**Method Name:** ApplyDeposit

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT ApplyDeposit ([in] BSTR xmlDataSource,
                      [in] BSTR xmlAcctTrans);
```

### Unapply Deposit

**Description:** Used to cancel a previously posted deposit application transaction.

**Method Name:** UnapplyDeposit

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT UnapplyDeposit ([in] BSTR xmlDataSource,
                        [in] BSTR xmlAcctTrans);
```

### Post Statement

**Description:** Used to post a single deferred statement transaction for an individual account. This transaction typically indicates the current balance amount for the account.

**Method Name:** PostStatement

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT PostStatement ( [in] BSTR xmlDataSource,
                        [in] BSTR xmlAcctTrans);
```

### Post Bill

**Description:** Used to post a single bill (invoice) transaction for an individual account.

**Method Name:** PostBill

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT PostBill ([in] BSTR xmlDataSource,
                  [in] BSTR xmlAcctTrans);
                [out, retval] BSTR* xmlTransOut);
```

## Get Bill Info

**Description:** Used to obtain billing information about an account, including the account's current and past due balances.

**Method Name:** GetBillInfo

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT GetBillInfo ( [in] BSTR xmlDataSource,  
                     [in] BSTR xmlAccountIn,  
                     [in] BSTR xmlAccountOut);
```

## Cancel Transactions

**Description:** Used to cancel transactions created by rate schedules for an individual account.

**Method Name:** CancelTransactions

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT CancelTransactions ([in] BSTR xmlDataSource,  
                           [in] BSTR xmlAcctTrans);
```

## Interface Arguments

The methods available in the Oracle Utilities Receivables Component Billing interface use the following arguments:

### **xmlDataSource Argument**

The xmlDataSource argument is an xml string that contains database connection and other related information. A DTD, xml example, and data element descriptions for this argument can be found on page 15-7 in the *Oracle Utilities Energy Information Platform Configuration Guide*.

### **xmlAcctTrans Argument**

The xmlAcctTrans argument is an xmlTransaction xml string containing the appropriate information to perform the specific function. A DTD and data element descriptions for this argument can be found on page 16-6. XML examples for this argument can be found on page 17-8.

### **xmlTransOut Argument**

The xmlTransOut argument is an xmlTransaction xml string that contains attributes passed to the Post Bill method.

### **xmlInstallmentPlan Argument**

The xmlInstallmentPlan argument is an xml string containing the appropriate information to perform the specific function. A DTD, XML examples and data element descriptions for this argument can be found on page 17-10.

### **xmlAccountIn Argument**

The xmlAccountIn argument is an xml string that contains the information necessary to identify an account and the date for which to get the information. The only information that is required in this structure is the unique ID of the account (ACCOUNT UID or ACCOUNT ID, but not both) and the Balance Date (the Balance Date will default to the current date if not provided). A DTD, xml example, and data element descriptions for this argument can be found on page 17-12.

### **xmlAccountOut Argument**

The xmlAccountOut argument is an xml string that contains an account's billing information. A DTD, xml example, and data element descriptions for this argument can be found on page 17-14.

# Input Values

XML examples and data element descriptions used as input values for the Oracle Utilities Receivables Component Billing interface (IBilling) are provided below.

## xmlAcctTrans

### XML Examples - xmlAcctTrans

#### Post Charge or Credit

```
<TRANSACTION>
  <ACCOUNT UID="338">
    </ACCOUNT>
    <TRANSACTIONID>300</TRANSACTIONID>
    <CHARGEORCREDIT>CH</CHARGEORCREDIT>
    <AMOUNT CURRENCY="USD">81</AMOUNT>
    <BILLEDORPAIDDATE>2000-10-25</BILLEDORPAIDDATE>
    <DUEDATE>2000-11-16</DUEDATE>
    <RECEIVABLETYPE UID="2" NAME="ESCO ELECTRIC" />
    <CHARGETYPE UID="177" NAME="ESCO ELECTRIC ENERGY CHARGE" />
    <OPCODE>AGL</OPCODE>
    <JURISCODE>GA</JURISCODE>
    <SERVICEPLAN UID = "286">
    </SERVICEPLAN>
  </TRANSACTION>
```

#### Cancel Charge or Credit

```
<TRANSACTION UID="1987" />
```

**Note:** The xmlAcctTrans arguments used for Cancel methods contains the minimal information required to identify the transaction. They can also include additional information as desired, such as Cancel Reason, Cancel Note, or other information.

#### Post Tax

```
<TRANSACTION>
  <ACCOUNT UID="406" />
  <RELATEDTRANSACTIONS>
    <TRANSACTION UID="2629">
      <AMOUNT CURRENCY="USD">435.00</AMOUNT>
      <TAXRATE>0.03</TAXRATE>
    </TRANSACTION>
  </RELATEDTRANSACTIONS>
</TRANSACTION>
```

#### Cancel Tax

```
<TRANSACTION UID="1987" />
```

#### Post Installment

```
<TRANSACTION>
  <ACCOUNT UID="387"/>
  <INSTALLMENTPLAN UID="315"/>
</TRANSACTION>
```

#### Cancel Installment

```
<TRANSACTION UID="1987" />
```

#### Post Deposit

```
<TRANSACTION>
  <ACCOUNT UID="387"/>
  <AMOUNT CURRENCY="USD">100.00</AMOUNT>
</TRANSACTION>
```



**Cancel Deposit**

```
<TRANSACTION UID="1987" />
```

**Post Deposit Interest**

```
<TRANSACTION>
  <ACCOUNT UID="387"/>
  <AMOUNT CURRENCY="USD">200.00</AMOUNT>
  <DEPOSIT UID="315"/>
</TRANSACTION>
```

**Cancel Deposit Interest**

```
<TRANSACTION UID="1987" />
```

**Apply Deposit**

```
<TRANSACTION>
  <ACCOUNT UID="387"/>
  <AMOUNT CURRENCY="USD">200.00</AMOUNT>
  <DEPOSIT UID="315"/>
</TRANSACTION>
```

**Unapply Deposit**

```
<TRANSACTION UID="1987" />
```

**Post Statement**

```
<TRANSACTION>
  <ACCOUNT UID="387"/>
  <AMOUNT CURRENCY="USD">100.00</AMOUNT>
</TRANSACTION>
```

**Post Bill**

```
<TRANSACTION>
  <ACCOUNT UID="387"/>
  <AMOUNT CURRENCY="USD">100.00</AMOUNT>
</TRANSACTION>
```

**Cancel Transactions**

```
<TRANSACTION>
  <ACCOUNT UID="292">
  </ACCOUNT>
  <BILLCYCLEDATE>2000-08-31</BILLCYCLEDATE>
</TRANSACTION>
```

## xmlInstallmentPlan

### DTD - xmlInstallmentPlan

```

<!DOCTYPE INSTALLMENTPLAN
[
<!ELEMENT INSTALLMENTPLAN (STOPTIME?, TRANSACTIONID?,
UIDRECEIVABLETYPE?, UIDCHARGETYPE?, OPCOCODE?, JURISCODE?, DEPOSIT?,
NUMINSTALLMENTS?, TOTALAMOUNT?, FIRSTAMOUNT?, INSTAMOUNT?,
REINSTALLMENTS?, REMAMOUNT?, LASTINSTMONTH?)>
<!ATTLIST INSTALLMENTPLAN
    UIDCDATA          #IMPLIED
    UIDTRANSACTIONCDATA#IMPLIED
    TRANSACTIONNOCDATA#IMPLIED
    UIDACCOUNTCDATA  #IMPLIED>
<!ELEMENT STOPTIME (#PCDATA)>
<!ELEMENT TRANSACTIONID (#PCDATA)>
<!ELEMENT UIDRECEIVABLETYPE (#PCDATA)>
<!ELEMENT UIDCHARGETYPE (#PCDATA)>
<!ELEMENT OPCOCODE (#PCDATA)>
<!ELEMENT JURISCODE (#PCDATA)>
<!ELEMENT DEPOSIT EMPTY>
<!ATTLIST DEPOSIT
    UIDCDATA          #IMPLIED>
<!ELEMENT NUMINSTALLMENTS (#PCDATA)>
<!ELEMENT TOTALAMOUNT (#PCDATA)>
<!ELEMENT FIRSTAMOUNT (#PCDATA)>
<!ELEMENT INSTAMOUNT (#PCDATA)>
<!ELEMENT REINSTALLMENTS (#PCDATA)>
<!ELEMENT REMAMOUNT (#PCDATA)>
<!ELEMENT LASTINSTMONTH (#PCDATA)>]>

```

### XML Examples - xmlInstallmentPlan

#### Create Installment Plan

```
<INSTALLMENTPLAN UID="1987" />
```

#### Cancel Installment Plan

```
<INSTALLMENTPLAN UID="1987" />
```

### Element Descriptions - xmlInstallmentPlan

The use of each individual attribute and element in the xmlInstallmentPlan argument is described below.

Installment Plan Attributes:

**UID:** Unique identifier for installment plan.

**UIDTRANSACTION:** Unique identifier for associated deferred charge transaction.

**TRANSACTIONNO:** Transaction number for associated deferred charge transaction.

**UIDACCOUNT:** Unique identifier for associated account.

Elements:

**STOPTIME:** Stop time for installment plan. If not null then installment plan is no longer in effect.

**TRANSACTIONID:** Default Transaction ID for installments associated with this installment plan.

---

**UIDRECEIVABLETYPE:** Default Receivable Type Uid for installments associated with this installment plan.

**UIDCHARGETYPE:** Default Charge Type Uid for installments associated with this installment plan.

**OPCODE:** Default Operating Company Code for installments associated with this installment plan.

**JURISCODE:** Default Jurisdiction Code for installments associated with this installment plan.

**DEPOSIT:** Optional deposit associated with installment plan.

Attributes:

**UID:** Unique identifier for deposit.

**NUMINSTALLMENTS:** Number of installments for installment plan.

**TOTALAMOUNT:** Total amount of all installments for installment plan.

Attributes:

**CURRENCY:** Currency code for the total amount.

**FIRSTAMOUNT:** Amount of first installment, if different.

Attributes:

**CURRENCY:** Currency code for the first amount.

**INSTAMOUNT:** Installment amount.

Attributes:

**CURRENCY:** Currency code for the installment amount.

**REINSTALLMENTS:** Number of remaining unbilled installments.

**REMAMOUNT:** Remaining unbilled amount.

Attributes:

**CURRENCY:** Currency code for the remaining amount.

**LASTINSTMONTH:** Revenue month of last installment.

## xmlAccountIn

### DTD - xmlAccountIn

```

<!DOCTYPE ACCOUNT
[
<!ELEMENT ACCOUNT (CUSTOMER?, STARTTIME?, STOPTIME?, OPCOCODE?,
JURISCODE?, REGIONCODE?, SIC?, STATUSCODE?, REVENUECODE?,
LASTBALANCETIME?, LASTBALANCE?, NEWACTIVITY?, UIDTXFRTOACCT?,
RECEIVABLESTATUS?, WRITEOFFREASONCODE?, BALANCEDATE?, CURRENTBALANCE?,
PASTDUEBALANCE?, LASTTXACTIONNO?)>
<!ATTLIST ACCOUNT
    UID CDATA                #IMPLIED
    ACCOUNTID CDATA         #IMPLIED
    UIDSUBACCOUNT CDATA     #IMPLIED
    SUBACCOUNTID CDATA     #IMPLIED
    NAME CDATA              #IMPLIED>
<!ELEMENT CUSTOMER EMPTY>
<!ATTLIST CUSTOMER
    UID CDATA                #IMPLIED>
<!ELEMENT STARTTIME (#PCDATA)>
<!ELEMENT STOPTIME (#PCDATA)>
<!ELEMENT OPCOCODE (#PCDATA)>
<!ELEMENT JURISCODE (#PCDATA)>
<!ELEMENT REGIONCODE (#PCDATA)>
<!ELEMENT SIC (#PCDATA)>
<!ELEMENT STATUSCODE (#PCDATA)>
<!ELEMENT REVENUECODE (#PCDATA)>
<!ELEMENT LASTBALANCETIME (#PCDATA)>
<!ELEMENT LASTBALANCE (#PCDATA)>
<!ELEMENT NEWACTIVITY (#PCDATA)>
<!ELEMENT UIDTXFRTOACCT (#PCDATA)>
<!ELEMENT RECEIVABLESTATUS (#PCDATA)>
<!ELEMENT WRITEOFFREASONCODE (#PCDATA)>
<!ELEMENT BALANCEDATE (#PCDATA)>
<!ELEMENT CURRENTBALANCE (#PCDATA)>
<!ELEMENT PASTDUEBALANCE (#PCDATA)>
<!ELEMENT LASTTXACTIONNO (#PCDATA)>
]>

```

### XML Example - xmlAccountIn

```

<ACCOUNT UID="139">
  <BALANCEDATE>2000-11-16</BALANCEDATE>
</ACCOUNT>

```

### Element Descriptions - xmlAccountIn

The use of each individual attribute and element in the xmlAccountIn argument is described below.

Account Attributes:

**UID:** Unique identifier for account.

**ACCOUNTID:** Alternate unique identifier for account.

**UIDSUBACCOUNT:** Optional unique identifier for sub-account for which this account is the master.

**SUBACCOUNTID:** Optional ID for sub-account for which this account is the master.

**NAME:** Optional name of account.

---

Elements:

**CUSTOMER:** Parent customer of account.

Attributes:

**UID:** Unique identifier for customer.

**STARTTIME:** Effective start time for account.

**STOPTIME:** Optional stop time for closed account.

**OPCOCODE:** Optional operating company code for account.

**JURISCODE:** Optional jurisdiction code for account.

**REGIONCODE:** Optional region code for account.

**SIC:** Optional SIC code for account.

**STATUSCODE:** Optional status code for account.

**REVENUECODE:** Optional revenue code for account.

**LASTBALANCETIME:** Time that account was last balanced.

**LASTBALANCE:** Balance of account at the last balance time.

Attributes:

**CURRENCY:** Currency code for the balance.

**NEWACTIVITY:** Activity for account since the last balance time.

Attributes:

**CURRENCY:** Currency code for the new activity.

**UIDTXFRTOACCT:** Optional UID of account to which this account has been transferred.

**RECEIVABLESTATUS:** Receivable status for account. May be one of "CURRENT", "PASTDUE", "COLLECTIONS", or "UNCOLLECTIBLE".

**WRITEOFFREASONCODE:** Optional write-off reason for account if Receivable Status is "UNCOLLECTIBLE".

**BALANCEDATE:** Date for current and past due balance calculations.

**CURRENTBALANCE:** Current balance as of balance date for account.

Attributes:

**CURRENCY:** Currency code for the current balance.

**PASTDUEBALANCE:** Past due balance as of balance date for account.

Attributes:

**CURRENCY:** Currency code for the past due balance.

**LASTTXACTIONNO:** Last transaction number for account.

## Return Values

The data returned from the Oracle Utilities Receivables Component Billing interface is described in the following DTD, xml example, and data element descriptions.

### xmlAccountOut

#### DTD - xmlAccountOut

```

<!DOCTYPE ACCOUNT
[
<!ELEMENT ACCOUNT (CUSTOMER?, STARTTIME?, STOPTIME?, OPCOCODE?
JURISCODE?, REGIONCODE?, SIC?, STATUSCODE?, REVENUECODE?,
LASTBALANCETIME?, LASTBALANCE?, NEWACTIVITY?, UIDTXFRTOACCT?,
RECEIVABLESTATUS?, WRITEOFFREASONCODE?, BALANCEDATE?, CURRENTBALANCE?,
PASTDUEBALANCE?, LASTTXACTIONNO?)>
<!ATTLIST ACCOUNT
    UIDCDATA                #IMPLIED
    ACCOUNTIDCDATA          #IMPLIED
    UIDSUBACCOUNTCDATA      #IMPLIED
    SUBACCOUNTIDCDATA       #IMPLIED
    NAMECDATA                #IMPLIED>
<!ELEMENT CUSTOMER EMPTY>
<!ATTLIST CUSTOMER
    UIDCDATA                #IMPLIED>
<!ELEMENT STARTTIME (#PCDATA)>
<!ELEMENT STOPTIME (#PCDATA)>
<!ELEMENT OPCOCODE (#PCDATA)>
<!ELEMENT JURISCODE (#PCDATA)>
<!ELEMENT REGIONCODE (#PCDATA)>
<!ELEMENT SIC (#PCDATA)>
<!ELEMENT STATUSCODE (#PCDATA)>
<!ELEMENT REVENUECODE (#PCDATA)>
<!ELEMENT LASTBALANCETIME (#PCDATA)>
<!ELEMENT LASTBALANCE (#PCDATA)>
<!ELEMENT NEWACTIVITY (#PCDATA)>
<!ELEMENT UIDTXFRTOACCT (#PCDATA)>
<!ELEMENT RECEIVABLESTATUS (#PCDATA)>
<!ELEMENT WRITEOFFREASONCODE (#PCDATA)>
<!ELEMENT BALANCEDATE (#PCDATA)>
<!ELEMENT CURRENTBALANCE (#PCDATA)>
<!ELEMENT PASTDUEBALANCE (#PCDATA)>
<!ELEMENT LASTTXACTIONNO (#PCDATA)>
]>

```

## XML Example - xmlAccountOut

```

<ACCOUNT UID="127">
  <ACCOUNTID>BE-DNT</ACCOUNTID>
  <CUSTOMERID>BE-DNT</CUSTOMERID>
  <STARTTIME>2000-01-01</STARTTIME>
  <LASTBALANCETIME/>
  <LASTBALANCE/>
  <NEWACTIVITY/>
  <RECIEVABLESTATUS>CURRENT</RECIEVABLESTATUS>
  <BALANCEDATE>2000-07-07</BALANCEDATE>
  <CURRENTBALANCE CURRENCY="USD">$1000</CURRENTBALANCE>
  <PASTDUEBALANCE/>
  <LASTTXACTIONNO>9</LASTTXACTIONNO>
</ACCOUNT>

```

## Element Descriptions - xmlAccountOut

The use of each individual attribute and element in the xmlAccountOut argument is described below.

Account Attributes:

**UID:** Unique identifier for account.

**ACCOUNTID:** Alternate unique identifier for account.

**UIDSUBACCOUNT:** Optional unique identifier for sub-account for which this account is the master.

**SUBACCOUNTID:** Optional ID for sub-account for which this account is the master.

**NAME:** Optional name of account.

Elements:

**CUSTOMER:** Parent customer of account.

Attributes:

**UID:** Unique identifier for customer.

**STARTTIME:** Effective start time for account.

**STOPTIME:** Optional stop time for closed account.

**OPCOCODE:** Optional operating company code for account.

**JURISCODE:** Optional jurisdiction code for account.

**REGIONCODE:** Optional region code for account.

**SIC:** Optional SIC code for account.

**STATUSCODE:** Optional status code for account.

**REVENUECODE:** Optional revenue code for account.

**LASTBALANCETIME:** Time that account was last balanced.

**LASTBALANCE:** Balance of account at the last balance time.

Attributes:

**CURRENCY:** Currency code for the balance.

**NEWACTIVITY:** Activity for account since the last balance time.

Attributes:

**CURRENCY:** Currency code for the new activity.

**UIDTXFRTOACCT:** Optional UID of account to which this account has been transferred.

**RECEIVABLESTATUS:** Receivable status for account. May be one of “CURRENT”, “PASTDUE”, “COLLECTIONS”, or “UNCOLLECTIBLE”.

**WRITEOFFREASONCODE:** Optional write-off reason for account if Receivable Status is “UNCOLLECTIBLE”.

**BALANCEDATE:** Date for current and past due balance calculations.

**CURRENTBALANCE:** Current balance as of balance date for account.

Attributes:

**CURRENCY:** Currency code for the current balance.

**PASTDUEBALANCE:** Past due balance as of balance date for account.

Attributes:

**CURRENCY:** Currency code for the past due balance.

**LASTTXACTIONNO:** Last transaction number for account.



## Deprecated Methods

The following methods have been replaced by the Post Charge Or Credit and Cancel Charge Or Credit methods (as appropriate). Although these methods are still supported, using the Post Charge or Credit method with specified data as appropriate will produce the same results.

### Post Service Charge

**Description:** Used to post charges (or credits) associated with a service plan for an individual account.

**Method Name:** PostServiceCharge

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT PostServiceCharge ([in] BSTR xmlDataSource,
                           [in] BSTR xmlAcctTrans);
```

### Post Deferred Service Charge

**Description:** Used to post deferred charges (or credits) associated with a service plan for an individual account.

**Method Name:** PostDeferredServiceCharge

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT PostDeferredServiceCharge ([in] BSTR xmlDataSource,
                                   [in] BSTR xmlAcctTrans);
```

### Post Budget Service Charge

**Description:** Used to post actual service charges (or credits) against an account that is on budget billing.

**Method Name:** PostBudgetServiceCharge

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT PostBudgetServiceCharge ([in] BSTR xmlDataSource,
                                  [in] BSTR xmlAcctTrans);
```

### Post Budget Bill Charge

**Description:** Used to post the billed budget amount associated with a budget plan for an individual account.

**Method Name:** PostBudgetBillCharge

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT PostBudgetBillCharge ([in] BSTR xmlDataSource,  
                              [in] BSTR xmlAcctTrans);
```

### Post Budget Bill Trueup

**Description:** Used to post a single true-up transaction charge or credit associated with a Budget Plan for an individual account.

**Method Name:** PostBudgetBillTrueup

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT PostBudgetBillTrueup ([in] BSTR xmlDataSource,  
                              [in] BSTR xmlAcctTrans);
```

### Post Installment Charge

**Description:** Used to post a single installment charge transaction for an individual account.

**Method Name:** PostInstallmentCharge

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT PostInstallmentCharge ([in] BSTR xmlDataSource,  
                               [in] BSTR xmlAcctTrans);
```

### Cancel Service Charge

**Description:** Used to cancel charges (or credits) associated with a service plan for an individual account.

**Method Name:** CancelServiceCharge

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT CancelServiceCharge ([in] BSTR xmlDataSource,  
                             [in] BSTR xmlAcctTrans);
```

### Cancel Deferred Service Charge

**Description:** Used to cancel deferred charges (or credits) associated with a service plan for an individual account.

**Method Name:** CancelDeferredServiceCharge

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT CancelDeferredServiceCharge ([in] BSTR xmlDataSource,
                                      [in] BSTR xmlAcctTrans);
```

**Cancel Budget Service Charge**

**Description:** Used to cancel actual service charges (or credits) against an account that is on budget billing.

**Method Name:** CancelBudgetServiceCharge

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT CancelBudgetServiceCharge ([in] BSTR xmlDataSource,
                                    [in] BSTR xmlAcctTrans);
```

**Cancel Budget Bill Charge**

**Description:** Used to cancel the billed budget amount associated with a budget plan for an individual account.

**Method Name:** CancelBudgetBillCharge

**Interface:** IBilling

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Billing

**Syntax:**

```
HRESULT CancelBudgetBillCharge ( [in] BSTR xmlDataSource,
                                 [in] BSTR xmlAcctTrans);
```

**Cancel Budget Bill Trueup**

**Description:** Used to cancel a single true-up transaction charge or credit associated with a Budget Plan for an individual account.

**Method Name:** CancelBudgetBillTrueup

**Interface:** IBilling

**Syntax:**

```
HRESULT CancelBudgetBillTrueup ( [in] BSTR xmlDataSource,
                                  [in] BSTR xmlAcctTrans);
```

**XML Examples**

**Post Service Charge**

```
<TRANSACTION>
  <ACCOUNT UID="338">
  </ACCOUNT>
  <TRANSACTIONID>300</TRANSACTIONID>
  <CHARGEORCREDIT>CH</CHARGEORCREDIT>
  <AMOUNT CURRENCY="USD">81</AMOUNT>
  <BILLEDORPAIDDATE>2000-10-25</BILLEDORPAIDDATE>
  <DUEDATE>2000-11-16</DUEDATE>
  <RECEIVABLETYPE UID="2" NAME="ESCO ELECTRIC" />
  <CHARGETYPE UID="177" NAME="ESCO ELECTRIC ENERGY CHARGE" />
  <OPCOCODE>AGL</OPCOCODE>
  <JURISCODE>GA</JURISCODE>
```

```

    <SERVICEPLAN UID = "286">
  </SERVICEPLAN>
</TRANSACTION>

```

### Post Deferred Service Charge

```

<TRANSACTION>
  <ACCOUNT UID="387"/>
  <TRANSACTIONID>520</TRANSACTIONID>
  <CHARGEORCREDIT>CH</CHARGEORCREDIT>
  <AMOUNT CURRENCY="USD">50.00</AMOUNT>
  <BILLEDORPAIDDATE>2000-10-02</BILLEDORPAIDDATE>
  <DUEDATE>2000-10-02</DUEDATE>
  <OPCODE>PPL</OPCODE>
  <JURISCODE>PA</JURISCODE>
  <SERVICEPLAN UID="315">
</SERVICEPLAN>
</TRANSACTION>

```

### Post Budget Service Charge

```

<TRANSACTION>
  <ACCOUNT UID="387"></ACCOUNT>
  <TRANSACTIONID>1105</TRANSACTIONID>
  <REVENUEMONTH>2000-10-01</REVENUEMONTH>
  <CHARGEORCREDIT>CH</CHARGEORCREDIT>
  <AMOUNT CURRENCY="USD">100.00</AMOUNT>
  <BILLEDORPAIDDATE>10/03/2000</BILLEDORPAIDDATE>
  <DUEDATE>2000-10-13</DUEDATE>
  <RECEIVABLETYPE UID="10" NAME="BUDGET BILLING"/>
  <OPCODE>PPL</OPCODE>
  <JURISCODE>PA</JURISCODE>
  <SERVICEPLAN UID="315"/>
  <BUDGETPLAN UID="27"/>
</TRANSACTION>

```

### Post Budget Bill Charge

```

<TRANSACTION>
  <ACCOUNT UID="387">
</ACCOUNT>
  <TRANSACTIONID>1100</TRANSACTIONID>
  <TRANSACTIONTIME>2000-10-03T00:00:01</TRANSACTIONTIME>
  <REVENUEMONTH>2000-10-01</REVENUEMONTH>
  <CHARGEORCREDIT>CH</CHARGEORCREDIT>
  <AMOUNT CURRENCY="USD">100.00</AMOUNT>
  <BILLEDORPAIDDATE>2000-10-03</BILLEDORPAIDDATE>
  <DUEDATE>2000-10-13</DUEDATE>
  <RECEIVABLETYPE UID="10" NAME="BUDGET BILLING"/>
  <OPCODE>PPL</OPCODE>
  <JURISCODE>PA</JURISCODE>
  <SERVICEPLAN UID="315"/>
  <BUDGETPLAN UID="27"/>
</TRANSACTION>

```

### Post Budget Bill Trueup

```

<TRANSACTION>
  <ACCOUNT UID="387">
</ACCOUNT>
  <TRANSACTIONID>1120</TRANSACTIONID>
  <TRANSACTIONTIME>2000-10-03T00:00:01</TRANSACTIONTIME>
  <REVENUEMONTH>2000-10-01</REVENUEMONTH>
  <CHARGEORCREDIT>CR</CHARGEORCREDIT>
  <AMOUNT CURRENCY="USD">200.00</AMOUNT>
  <BILLEDORPAIDDATE>2000-10-03</BILLEDORPAIDDATE>
  <DUEDATE>2000-10-13</DUEDATE>
  <RECEIVABLETYPE UID="10" NAME="BUDGET BILLING"/>
  <OPCODE>PPL</OPCODE>
  <JURISCODE>PA</JURISCODE>
  <SERVICEPLAN UID="315"/>

```

```
<BUDGETPLAN UID="27" />
</TRANSACTION>
```

### Post Installment Charge

```
<TRANSACTION>
  <ACCOUNT UID="338">
  </ACCOUNT>
  <TRANSACTIONID>1510</TRANSACTIONID>
  <AMOUNT CURRENCY="USD">100.00</AMOUNT>
  <BILLEDORPAIDDATE>10/03/2000</BILLEDORPAIDDATE>
  <DUEDATE>10/13/2000</DUEDATE>
</TRANSACTION>
```

### Cancel Service Charge

```
<TRANSACTION UID="1987" />
```

**Note:** The xmlAcctTrans arguments used for Cancel methods contains the minimal information required to identify the transaction. They can also include additional information as desired, such as Cancel Reason, Cancel Note, or other information.

### Cancel Deferred Service Charge

```
<TRANSACTION UID="631" />
```

### Cancel Budget Service Charge

```
<TRANSACTION UID="1282" />
```

### Cancel Budget Bill Charge

```
<TRANSACTION UID="637" />
```

### Cancel Budget Bill Trueup

```
<TRANSACTION UID="644" />
```

### Cancel Installment Charge

```
<TRANSACTION UID="1322" />
```



# Chapter 18

---

## Oracle Utilities Receivables Component Remittance Interface

This chapter describes the methods/functions available to external systems through the Oracle Utilities Receivables Component Remittance interface (IRemittance). These methods allow users to perform a number of remittance functions available in the Remittance module of the Oracle Utilities Receivables Component to Oracle Utilities Billing Component from external systems. These functions include the following:

- Post/Cancel Payment
- Process Batch Payment
- Process Payment File
- Process/Cancel AutoPayment
- Batch AutoPayments.

See **Chapter 7: Remittance** for more information about these functions.

## Methods, Interfaces, and Syntax

The methods, interface objects, and syntax for the Oracle Utilities Receivables Component Remittance interface are as follows:

### Post Payment

**Description:** Used to post payments at the account level.

**Method Name:** PostPayment

**Interface:** IRemittance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Remittance

**Syntax:**

```
HRESULT PostPayment ( [in] BSTR xmlDataSource,  
                     [in] BSTR xmlPayment);
```

### Process Batch Payment

**Description:** Used to process batch payments containing one or more individual payments.

**Method Name:** ProcessBatchPayment

**Interface:** IRemittance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Remittance

**Syntax:**

```
HRESULT ProcessBatchPayment ([in] BSTR xmlDataSource,  
                             [in] BSTR xmlBatchPayment);
```

### Process Payment File

**Description:** Used to process payment files that contain one or more batch payments.

**Method Name:** ProcessPaymentFile

**Interface:** IRemittance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Remittance

**Syntax:**

```
HRESULT ProcessPaymentFile ([in] BSTR xmlDataSource,  
                            [in] BSTR xmlPaymentFile);
```

### Cancel Payment

**Description:** Used to cancel payments at the account level.

**Method Name:** CancelPayment

**Interface:** IRemittance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Remittance

**Syntax:**

```
HRESULT CancelPayment ( [in] BSTR xmlDataSource,  
                       [in] BSTR xmlPayment);
```



---

## Process AutoPayment

**Description:** Used to process an automatic payment for an account.

**Method Name:** ProcessAutoPayment

**Interface:** IRemittance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Remittance

**Syntax:**

```
HRESULT ProcessAutoPayment ([in] BSTR xmlDataSource,  
                             [in] BSTR xmlAcctBill);
```

## Cancel AutoPayment

**Description:** Used to cancel an automatic payment for an account.

**Method Name:** CancelAutoPayment

**Interface:** IRemittance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Remittance

**Syntax:**

```
HRESULT CancelAutoPayment ( [in] BSTR xmlDataSource,  
                             [in] BSTR xmlAcctBill);
```

## Batch AutoPayments

**Description:** Used to group all pending automatic payments by payment source and forward them to the appropriate outbound interface.

**Method Name:** BatchAutoPayments

**Interface:** IRemittance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Remittance

**Syntax:**

```
HRESULT BatchAutoPayments ([in] BSTR xmlDataSource);
```

## Interface Arguments

The methods available in the Oracle Utilities Receivables Component Remittance interface use the following arguments:

### **xmlDataSource Argument**

The xmlDataSource argument is an xml string that contains database connection and other related information. A DTD, xml example, and data element descriptions for the xmlDataSource argument can be found on page 15-7 in the *Oracle Utilities Energy Information Platform Configuration Guide*.

### **xmlPayment Argument**

The xmlPayment argument is an xml string that contains data necessary to post a payment. A DTD, xml example, and data element descriptions for the xmlPayment argument can be found on page 18-5.

### **xmlBatchPayment Argument**

The xmBatchPayment argument is an xml string that contains data necessary to process a batch payment. A DTD, xml example, and data element descriptions for the xmlBatchPayment argument can be found on page 18-9.

### **xmlPaymentFile Argument**

The xmlPaymentFile argument is an xml string that contains data necessary to process a payment file. A DTD, xml example, and data element descriptions for the xmlPaymentFile argument can be found on page 18-11.

### **xmlAcctBill Argument**

The xmlAcctBill argument is an xmlTransaction xml string containing the appropriate information to process or cancel an automatic payment. A DTD and data element descriptions for this argument can be found on page 16-6. An xml example for this argument can be found on page 18-13.

## Input Values

The Data Type Definition (DTD), an xml example, and data element descriptions used as input values for the Oracle Utilities Receivables Component Remittance interface (IRemittance) are provided below.

### xmlPayment

#### DTD - xmlPayment

```
<!DOCTYPE PAYMENT
[
<!ELEMENT PAYMENT (DEFAULTACCOUNTID, TRANSACTIONID?, REVENUEMONTH?,
NOTE?, ACCOUNTID?, SOURCECODE?, BATCHPAYMENT?, BATCHCANCEL?,
PAYMENTID?, DATE?, AMOUNT, METHODCODE?, INSTITUTION?, ACCOUNTNAME?,
ACCOUNTNO?, ACCOUNTZIP?, CHECKNO?, EXPDATE?, RTN?,
AUTOPAYMENTTIMESTAMP?, AUTOPAYMENTDATE?, BILLDATE?, INVOICEID?,
INVOICEDATE?, MISC1?, MISC2?, MISC3?, CANCELREVENUEMONTH?,
CANCELREASONCODE?, CANCELNOTE?, POSTPENALTY?, UIDASSISTPROGRAM?,
ASSISTPROGRAMID? RELATEDTRANSACTIONS?)>
<!ATTLIST PAYMENT
    UIDCDATA          #IMPLIED>
    APPLICATIONMETHOD (IMMEDIATE|
        (INVOICEID|
            (RECEIVABLETYPE| "IMMEDIATE">
<!ELEMENT DEFAULTACCOUNTID (#PCDATA)>
<!ELEMENT TRANSACTIONID (#PCDATA)>
<!ELEMENT REVENUEMONTH (#PCDATA)>
<!ELEMENT NOTE (#PCDATA)>
<!ELEMENT ACCOUNTID (#PCDATA)>
<!ELEMENT SOURCECODE (#PCDATA)>
<!ELEMENT BATCHPAYMENT EMPTY>
<!ATTLIST BATCHPAYMENT
    UIDCDATA          #IMPLIED>
<!ELEMENT BATCHCANCEL EMPTY>
<!ATTLIST BATCHCANCEL
    UIDCDATA          #IMPLIED>
<!ELEMENT PAYMENTID (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT AMOUNT (#PCDATA)>
<!ELEMENT METHODCODE (#PCDATA)>
<!ELEMENT INSTITUTION (#PCDATA)>
<!ELEMENT ACCOUNTNAME (#PCDATA)>
<!ELEMENT ACCOUNTNO (#PCDATA)>
<!ELEMENT ACCOUNTZIP (#PCDATA)>
<!ELEMENT CHECKNO (#PCDATA)>
<!ELEMENT EXPDATE (#PCDATA)>
<!ELEMENT RTN (#PCDATA)>
<!ELEMENT AUTOPAYMENTTIMESTAMP (#PCDATA)>
<!ELEMENT AUTOPAYMENTDATE (#PCDATA)>
<!ELEMENT BILLDATE (#PCDATA)>
<!ELEMENT INVOICEID (#PCDATA)>
<!ELEMENT INVOICEDATE (#PCDATA)>
<!ELEMENT MISC1 (#PCDATA)>
<!ELEMENT MISC2 (#PCDATA)>
<!ELEMENT MISC3 (#PCDATA)>
<!ELEMENT CANCELREVENUEMONTH (#PCDATA)>
<!ELEMENT CANCELREASONCODE (#PCDATA)>
<!ELEMENT CANCELNOTE (#PCDATA)>
<!ELEMENT POSTPENALTY EMPTY>
<!ELEMENT UIDASSISTPROGRAM (CDATA)>
```

```

<!ELEMENT ASSISTPROGRAMID (CDATA)>
<!ELEMENT RELATEDTRANSACTIONS (TRANSACTION)>
<!ELEMENT TRANSACTION EMPTY>
<!ATTLIST TRANSACTION
    UID CDATA          #IMPLIED>
]>

```

## XML Examples - xmlPayment

### Single Payment

```

<PAYMENT>
  <DEFAULTACCOUNTID>9999999999</DEFAULTACCOUNTID>
  <ACCOUNTID>RM-81568-IGNORE-AGE-2-REC-TYPES-CY01</ACCOUNTID>
  <SOURCECODE>CHASE</SOURCECODE>
  <PAYMENTID>PAYIDRM81568</PAYMENTID>
  <DATE>2000-05-21</DATE>
  <AMOUNT CURRENCY="USD">400.00</AMOUNT>
  <METHODCODE>LOCKBOX</METHODCODE>
  <CHECKNO>CHECKNORM81468</CHECKNO>
</PAYMENT>

```

### Single Payment posted against a single charge transaction

```

<PAYMENT>
  <DEFAULTACCOUNTID>9999999999</DEFAULTACCOUNTID>
  <ACCOUNTID>RM-81568-IGNORE-AGE-2-REC-TYPES-CY01</ACCOUNTID>
  <SOURCECODE>CHASE</SOURCECODE>
  <PAYMENTID>PAYIDRM81568</PAYMENTID>
  <DATE>2000-05-21</DATE>
  <AMOUNT CURRENCY="USD">400.00</AMOUNT>
  <METHODCODE>LOCKBOX</METHODCODE>
  <CHECKNO>CHECKNORM81468</CHECKNO>
  <RELATEDTRANSACTIONS>
    <TRANSACTION UID="28668"/>
  </RELATEDTRANSACTIONS>
</PAYMENT>

```

### Single Payment posted against multiple charge transactions

```

<PAYMENT>
  <DEFAULTACCOUNTID>9999999999</DEFAULTACCOUNTID>
  <ACCOUNTID>RM-81568-IGNORE-AGE-2-REC-TYPES-CY01</ACCOUNTID>
  <SOURCECODE>CHASE</SOURCECODE>
  <PAYMENTID>PAYIDRM81568</PAYMENTID>
  <DATE>2000-05-21</DATE>
  <AMOUNT CURRENCY="USD">400.00</AMOUNT>
  <METHODCODE>LOCKBOX</METHODCODE>
  <CHECKNO>CHECKNORM81468</CHECKNO>
  <RELATEDTRANSACTIONS>
    <TRANSACTION UID="28655"/>
    <TRANSACTION UID="28656"/>
    <TRANSACTION UID="28657"/>
  </RELATEDTRANSACTIONS>
</PAYMENT>

```

## Element Descriptions - xmlPayment

The use of each individual attribute and element in the xmlPayment argument is described below.

Payment Attributes:

**UID:** Unique identifier for payment.

**APPLICATIONMETHOD:** Indicates credit application method.

Elements:

**DEFAULTACCOUNTID:** Default account identifier for payment. Used if actual account identifier cannot be determined.

**TRANSACTIONID:** Transaction ID for payment. If not provided, the default Transaction ID for the PYMNT Transaction Type will be used.

**REVENUEMONTH:** Optional revenue month for post payment. If not provided, the current month will be used.

**NOTE:** Optional note associated with posted payment.

**ACCOUNTID:** Unique Account ID for payment.

**SOURCECODE:** Payment source code for payment.

**BATCHPAYMENT:** Batch payment from where this payment came.

Attributes:

**UID:** Unique identifier for batch payment. Required for posting and canceling.

**BATCHCANCEL:** Batch payment that cancelled this payment.

Attributes:

**UID:** Unique identifier for batch payment. Required for posting and canceling.

**PAYMENTID:** Unique ID for payment within batch or source.

**DATE:** Date of payment.

**AMOUNT:** Amount of payment. Required for posting and canceling.

Attributes:

**CURRENCY:** Currency code for the amount.

**METHODCODE:** Payment method code for payment.

**INSTITUTION:** Institution from which payment is drawn.

**ACCOUNTNAME:** Name on account.

**ACCOUNTNO:** Unique identifier of account within above institution.

**ACCOUNTZIP:** ZIP code of account.

**CHECKNO:** Check number for above account.

**EXPDATE:** Expiration date for credit card account.

**RTN:** Routing transit number for direct debit.

**AUTOPAYMENTTIMESTAMP:** Transaction time that associated auto payment record was created.

**AUTOPAYMENTDATE:** Scheduled payment date of associated auto payment.

**BILLDATE:** Bill date of associated bill transaction when autopayments are used.

**INVOICEID:** Associated invoice Id for payment.

**INVOICEDATE:** Associated invoice date for payment.

**MISC1:** Optional miscellaneous payment attribute.

**MISC2:** Optional miscellaneous payment attribute.

**MISC3:** Optional miscellaneous payment attribute.

**CANCELREVENUEMONTH:** Optional revenue month for cancelled payment. If not provided, the current month will be used.

**CANCELREASONCODE:** Optional reason code for canceling payment.

**CANCELNOTE:** Optional note associated with cancelled payment.

**POSTPENALTY:** The existence of this element for a cancelled payment indicates that a penalty should be posted.

**UIDASSISTPROGRAM:** The existence of this element indicates that the payment is an Assistance Payment. Identifies which Assistance Program this Payment is associated with. If provided, ASSISTPROGRAMID will be ignored.

**ASSISTPROGRAMID:** The existence of this element indicates that the payment is an Assistance Payment. Identifies which Assistance Program this Payment is associated with.

**RELATEDTRANSACTIONS:** Element that contains one or more related charge transactions against which the payment will be posted.

Elements:

**TRANSACTION:** Element containing a single related charge transaction against which the payment will be posted.

Attributes:

**UID:** Unique ID for related charge transaction against which the payment will be posted.

## xmlBatchPayment

### DTD - xmlBatchPayment

```

<!DOCTYPE BATCHPAYMENT
[
<!ELEMENT BATCHPAYMENT (DEFAULTACCOUNTID?, TRANSACTIONID?,
REVENUEMONTH?, SOURCECODE?, PAYMENTFILE?, DATE?, METHODCODE?,
NUMPAYMENTS?, AMOUNT?, CANCELREVENUEMONTH?, CANCELREASONCODE?,
CANCELNOTE?, POSTPENALTY?, MAXERRORS?, PAYMENT*)>
<!ATTLIST BATCHPAYMENT
    UID CDATA          #IMPLIED
    BATCHNO CDATA      #IMPLIED
    CANCEL{TRUE|FALSE} "FALSE"
    RESTART (TRUE|FALSE) "FALSE">
<!ELEMENT DEFAULTACCOUNTID (#PCDATA)>
<!ELEMENT TRANSACTIONID (#PCDATA)>
<!ELEMENT REVENUEMONTH (#PCDATA)>
<!ELEMENT SOURCECODE (#PCDATA)>
<!ELEMENT PAYMENTFILE EMPTY>
<!ATTLIST PAYMENTFILE
    NAMECDATA          #IMPLIED>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT METHODCODE (#PCDATA)>
<!ELEMENT NUMPAYMENTS (#PCDATA)>
<!ELEMENT AMOUNT (#PCDATA)>
<!ELEMENT MAXERRORS (#PCDATA)>
<!ELEMENT CANCELREVENUEMONTH (#PCDATA)>
<!ELEMENT CANCELREASONCODE (#PCDATA)>
<!ELEMENT CANCELNOTE (#PCDATA)>
<!ELEMENT POSTPENALTY (#PCDATA)>
]>

```

### XML Example - xmlBatchPayment

```

<BATCHPAYMENT BATCHNO="DVBATCHNO100" ISAUTOPAY="N">
  <SOURCECODE>CHASE</SOURCECODE>
  <DATE>2000-10-02</DATE>
  <NUMPAYMENTS>2</NUMPAYMENTS>
  <AMOUNT CURRENCY="USD">1168.50</AMOUNT>
  <PAYMENT>
    <DEFAULTACCOUNTID>999999999</DEFAULTACCOUNTID>
    <ACCOUNTID>PETER-TEST-ACCOUNT</ACCOUNTID>
    <SOURCECODE>CHASE</SOURCECODE>
    <PAYMENTID>PAYIDPTA100</PAYMENTID>
    <DATE>2000-10-02</DATE>
    <AMOUNT CURRENCY="USD">1100.00</AMOUNT>
    <METHODCODE>LOCKBOX</METHODCODE>
    <CHECKNO>CHECKNOPTA100</CHECKNO>
  </PAYMENT>
  <PAYMENT>
    <DEFAULTACCOUNTID>999999999</DEFAULTACCOUNTID>
    <ACCOUNTID>BE-DARIA</ACCOUNTID>
    <SOURCECODE>CHASE</SOURCECODE>
    <PAYMENTID>PAYIDDV100</PAYMENTID>
    <DATE>2000-10-02</DATE>
    <AMOUNT CURRENCY="USD">68.50</AMOUNT>
    <METHODCODE>LOCKBOX</METHODCODE>
    <CHECKNO>CHECKNODV100</CHECKNO>
  </PAYMENT>
</BATCHPAYMENT>

```

## Element Descriptions - xmlBatchPayment

The use of each individual attribute and element in the xmlBatchPayment argument is described below.

Batch Payment Attributes:

**UID:** Unique identifier for batch payment.

**BATCHNO:** Unique number of batch payment within payment file or source.

**CANCEL:** Indicates that the batch should be processed in cancel mode.

**RESTART:** Indicates that the batch should be processed in restart mode.

Elements:

**DEFAULTACCOUNTID:** Optional default account ID for payments. Forwarded to individual payments if not already provided.

**TRANSACTIONID:** Optional Transaction ID for payments. Forwarded to individual payments if not already provided.

**REVENUEMONTH:** Optional revenue month for payments. Forwarded to individual payments if not already provided.

**SOURCECODE:** Payment source code for batch. Required for processing. Forwarded to individual payments if not already provided.

**PAYMENTFILE:** Payment file that the batch came from. Optional for processing.

Attributes:

**NAME:** Unique file path name of payment file.

**DATE:** Date of batch payment. Optional for processing. Forwarded to individual payments if not already provided.

**METHODCODE:** Payment method code for batch. Optional for processing. Forwarded to individual payments if not already provided.

**NUMPAYMENTS:** Total number of payments in the batch. Optional for processing.

**AMOUNT:** Total amount of all payments in the batch. Optional for processing.

Attributes:

**CURRENCY:** Currency code for the amount.

**CANCELREVENUEMONTH:** Optional revenue month for cancelled payment. If not provided then the current month will be used.

**CANCELREASONCODE:** Optional reason code for canceling payment.

**CANCELNOTE:** Optional note associated with cancelled payment.

**POSTPENALTY:** The existence of this element for a cancelled payment indicates that a penalty should be posted.

**MAXERRORS:** Maximum number of payment errors allowed prior to stopping the process. Optional for processing.

**PAYMENT\*:** Individual payments in the batch. Required for processing.



## xmlPaymentFile

### DTD - xmlPaymentFile

```

<!DOCTYPE PAYMENTFILE
[
<!ELEMENT PAYMENTFILE (DEFAULTACCOUNTID?, TRANSACTIONID?,
REVENUEMONTH?, SOURCECODE?, DATE?, METHODCODE?, NUMBATCHES?,
NUMPAYMENTS?, AMOUNT?, CANCELREVENUEMONTH?, CANCELREASONCODE?,
CANCELNOTE?, POSTPENALTY?, MAXERRORS?, MAXERRORSPERBATCH?,
BATCHPAYMENT*)>
<!ATTLIST PAYMENTFILE
    NAMECDATA          #IMPLIED>
    CANCEL (TRUE|FALSE) "FALSE">
    RESTART (TRUE|FALSE) "FALSE">
<!ELEMENT DEFAULTACCOUNTID (#PCDATA)>
<!ELEMENT TRANSACTIONID (#PCDATA)>
<!ELEMENT REVENUEMONTH (#PCDATA)>
<!ELEMENT SOURCECODE (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT METHODCODE (#PCDATA)>
<!ELEMENT NUMBATCHES (#PCDATA)>
<!ELEMENT NUMPAYMENTS (#PCDATA)>
<!ELEMENT AMOUNT (#PCDATA)>
<!ELEMENT CANCELREVENUEMONTH (#PCDATA)>
<!ELEMENT CANCELREASONCODE (#PCDATA)>
<!ELEMENT CANCELNOTE (#PCDATA)>
<!ELEMENT POSTPENALTY (#PCDATA)>
<!ELEMENT MAXERRORS (#PCDATA)>
<!ELEMENT MAXERRORSPERBATCH (#PCDATA)>
]>

```

### XML Example - xmlPaymentFile

```

<PAYMENTFILE NAME="F:\XMLSCRIPTS\REMITTANCE\DVPAYMENTFILE">
  <SOURCECODE>CHASE</SOURCECODE>
  <DATE>2000-10-02</DATE>
  <METHODCODE>LOCKBOX</METHODCODE>
  <NUMBATCHES>2</NUMBATCHES>
  <NUMPAYMENTS>3</NUMPAYMENTS>
  <AMOUNT CURRENCY="USD">300.00</AMOUNT>
  <BATCHPAYMENT BATCHNO="RMBATCHNODV101" ISAUTOPAY="N">
    <SOURCECODE>CHASE</SOURCECODE>
    <PAYMENTFILE NAME="F:\XMLSCRIPTS\REMITTANCE\DVPAYMENTFILE">
      <EMPTY/>
    </PAYMENTFILE>
    <DATE>2000-10-02</DATE>
    <NUMPAYMENTS>1</NUMPAYMENTS>
    <AMOUNT CURRENCY="USD">100.00</AMOUNT>
    <PAYMENT>
      <DEFAULTACCOUNTID>999999999</DEFAULTACCOUNTID>
      <ACCOUNTID>BE-DARIA</ACCOUNTID>
      <SOURCECODE>CHASE</SOURCECODE>
      <PAYMENTID>PAYIDDV101</PAYMENTID>
      <DATE>2000-10-02</DATE>
      <AMOUNT CURRENCY="USD">100.00</AMOUNT>
      <METHODCODE>LOCKBOX</METHODCODE>
      <CHECKNO>CHECKNODV101</CHECKNO>
    </PAYMENT>
  </BATCHPAYMENT>
  <BATCHPAYMENT BATCHNO="RMBATCHNODV102" ISAUTOPAY="N">
    <SOURCECODE>CHASE</SOURCECODE>
    <PAYMENTFILE NAME="F:\XMLSCRIPTS\REMITTANCE\DVPAYMENTFILE">
      <EMPTY/>
    </PAYMENTFILE>
  </BATCHPAYMENT>

```

```

<DATE>2000-10-02</DATE>
<NUMPAYMENTS>2</NUMPAYMENTS>
<AMOUNT CURRENCY="USD">200.00</AMOUNT>
<PAYMENT>
  <DEFAULTACCOUNTID>999999999</DEFAULTACCOUNTID>
  <ACCOUNTID>PETER-TEST-ACCOUNT</ACCOUNTID>
  <SOURCECODE>CHASE</SOURCECODE>
  <PAYMENTID>PAYIDPTA101</PAYMENTID>
  <DATE>2000-10-02</DATE>
  <AMOUNT CURRENCY="USD">100.00</AMOUNT>
  <METHODCODE>LOCKBOX</METHODCODE>
  <CHECKNO>CHECKNOPTA101</CHECKNO>
</PAYMENT>
<PAYMENT>
  <DEFAULTACCOUNTID>999999999</DEFAULTACCOUNTID>
  <ACCOUNTID>TRNSFR-TST-81760</ACCOUNTID>
  <SOURCECODE>CHASE</SOURCECODE>
  <PAYMENTID>PAYIDTT101</PAYMENTID>
  <DATE>2000-10-02</DATE>
  <AMOUNT CURRENCY="USD">100.00</AMOUNT>
  <METHODCODE>LOCKBOX</METHODCODE>
  <CHECKNO>CHECKNOTT101</CHECKNO>
</PAYMENT>
</BATCHPAYMENT>
</PAYMENTFILE>

```

## Element Descriptions - xmlPaymentFile

Each of the data elements and attributes used by the xmlPaymentFile argument is described below.

Payment File Attributes:

**NAME:** Unique file path name of payment file. Required for processing.

**CANCEL:** Indicates that the file should be processed in cancel mode.

**RESTART:** Indicates that the batch should be processed in restart mode.

Elements:

**DEFAULTACCOUNTID:** Optional default account ID for batch payments. Forwarded to individual batch payments if not already provided.

**TRANSACTIONID:** Optional Transaction ID for batch payments. Forwarded to individual batch payments if not already provided.

**REVENUEMONTH:** Optional revenue month for batch payments. Forwarded to individual batch payments if not already provided.

**SOURCECODE:** Required payment source code for the payment file. Forwarded to individual batch payments if not already provided.

**DATE:** Optional date of payment file. Forwarded to individual batch payments if not already provided.

**METHODCODE:** Optional payment method code for payment file. Forwarded to individual batch payments if not already provided.

**NUMBATCHES:** Total number of batches in the file. Optional for processing.

**NUMPAYMENTS:** Total number of payments in the file. Optional for processing.

**AMOUNT:** Total amount of all payments (or batch payments) in the file. Optional for processing.

Attributes:

**CURRENCY:** Currency code for the amount.

**CANCELREVENUEMONTH:** Optional revenue month for cancelled payment. If not provided then the current month will be used.

**CANCELREASONCODE:** Optional reason code for canceling payment.

**CANCELNOTE:** Optional note associated with cancelled payment.

**POSTPENALTY:** The existence of this element for a cancelled payment indicates that a penalty should be posted.

**MAXERRORS:** Maximum number of batch payment errors allowed prior to stopping the process. Optional for processing.

**MAXERRORSPERBATCH:** Maximum number of payment errors allowed per batch payment prior to stopping the batch payment process. Forwarded to individual batch payments if not already provided. Optional for processing.

**BATCHPAYMENT\*:** Individual batch payments in the file. Required for processing.

## xmlAcctBill

### XML Examples - xmlAcctBill

#### Process AutoPayment

```
<TRANSACTION UID="127">
  <AMOUNT CURRENCY="USD">15.0000</AMOUNT>
  <BILLEDORPAIDDATE>2000-06-14</BILLEDORPAIDDATE>
  <TRANSACTIONTIME>2000-09-19T14:47:22</TRANSACTIONTIME>
</TRANSACTION>
```

#### Cancel AutoPayment

```
<TRANSACTION>
  <UIDAUTOPAYPLAN>28</UIDAUTOPAYPLAN>
  <CANCELTIME>2000-08-17T11:46:05</CANCELTIME>
  <UIDTRANSACTION>967</UIDTRANSACTION>
</TRANSACTION>
```



# Chapter 19

---

## Oracle Utilities Receivables Component Maintenance Interface

This chapter describes the methods/functions available to external systems through the Oracle Utilities Receivables Component Maintenance interface (IMaintenance). These methods allow users to perform a number of maintenance functions available in the Maintenance module of the Oracle Utilities Receivables Component to Oracle Utilities Billing Component from external systems. These functions include the following:

- Post/Cancel Payment Transfer
- Post/Cancel Adjustment
- Post/Cancel Refund
- Post/Process Pending Refunds
- Process Time Voided Refunds
- Update Refund Status
- Post/Cancel Writeoff
- Write Off/Write On Account
- Post/Cancel Transaction
- Process Batch Transaction/Transaction File
- Post/Cancel Balance Transfer
- Transfer/Return Account Balance

See **Chapter 8: Maintenance** for more information about these functions.

## Methods, Interfaces, and Syntax

The methods, interface objects, and syntax for the Oracle Utilities Receivables Component Maintenance interface are as follows:

### Post Payment Transfer

**Description:** Used to post payment transfers between accounts.

**Method Name:** PostPaymentTransfer

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT PostPaymentTransfer ( [in] BSTR xmlDataSource,  
                             [in] BSTR xmlTransaction);
```

### Cancel Payment Transfer

**Description:** Used to cancel payment transfers between accounts.

**Method Name:** CancelPaymentTransfer

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT CancelPaymentTransfer ([in] BSTR xmlDataSource,  
                              [in] BSTR xmlTransaction);
```

### Post Adjustment

**Description:** Used to post an adjustment transaction credit or charge against an account.

**Method Name:** PostAdjustment

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT PostAdjustment ([in] BSTR xmlDataSource,  
                       [in] BSTR xmlTransaction);
```

### Cancel Adjustment

**Description:** Used to cancel an adjustment transaction credit or charge against an account.

**Method Name:** CancelAdjustment

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT CancelAdjustment ([in] BSTR xmlDataSource,  
                          [in] BSTR xmlTransaction);
```

## Post Refund

**Description:** Used to post a refund transaction charge against an account.

**Method Name:** PostRefund

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT PostRefund ([in] BSTR xmlDataSource,
                    [in] BSTR xmlTransaction);
```

## Cancel Refund

**Description:** Used to cancel a refund transaction charge against an account.

**Method Name:** CancelRefund

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT CancelRefund ([in] BSTR xmlDataSource,
                      [in] BSTR xmlTransaction);
```

## Post Pending Refunds

**Description:** Scans the Oracle Utilities Data Repository for all closed accounts with an unapplied credit, determines if the credit has been active on the account for the specified time period, and creates a Refund record in the database.

**Method Name:** PostPendingRefunds

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT PostPendingRefunds ([in] BSTR xmlDataSource,
                             [in] BSTR xmlBatchRefund);
```

## Process Pending Refunds

**Description:** Scans the Oracle Utilities Data Repository for all Refund records with a Status of PENDING, converts them into the Batch Refund, and stores them in a user-defined file to be used for issuing the refunds via checks.

**Method Name:** ProcessPendingRefunds

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT ProcessPendingRefunds ([in] BSTR xmlDataSource,
                                [in] BSTR xmlBatchRefund);
```

## Process Time Voided Refunds

**Description:** Scans the Oracle Utilities Data Repository for all Refund records with a Status of ISSUED and determines whether the checks have been cashed before the void date.

**Method Name:** ProcessTimeVoidedRefunds

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT ProcessTimeVoidedRefunds ([in] BSTR xmlDataSource,  
                                  [in] BSTR xmlBatchRefund);
```

## Update Refund Status

**Description:** Updates records in the Oracle Utilities Data Repository with refund status information.

**Method Name:** UpdateRefundStatus

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT UpdateRefundStatus ([in] BSTR xmlDataSource,  
                             [in] BSTR xmlBatchRefund);
```

## Post Writeoff

**Description:** Used to post a writeoff transaction credit or charge against an account.

**Method Name:** PostWriteOff

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT PostWriteOff ([in] BSTR xmlDataSource,  
                     [in] BSTR xmlTransaction);
```

## Cancel Writeoff

**Description:** Used to cancel a writeoff transaction credit or charge against an account.

**Method Name:** CancelWriteOff

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT CancelWriteOff ([in] BSTR xmlDataSource,  
                        [in] BSTR xmlTransaction);
```



**Writeoff Account**

**Description:** Used to post a write-off transaction for each outstanding (balance > 0) previously posted transaction for the account.

**Method Name:** WriteOffAccount

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT WriteOffAccount ([in] BSTR xmlDataSource,
                        [in] BSTR xmlTransaction);
```

**Writeon Account**

**Description:** Used to to cancel all previous write-off transactions for the account.

**Method Name:** WriteOnAccount

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT WriteOnAccount ([in] BSTR xmlDataSource,
                       [in] BSTR xmlTransaction);
```

**Post Transaction**

**Description:** Used to post a transaction of any type.

**Method Name:** PostTransaction

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT PostTransaction ( [in] BSTR xmlDataSource,
                          [in] BSTR xmlTransaction);
```

**Cancel Transaction**

**Description:** Used to cancel a transaction of any type.

**Method Name:** CancelTransaction

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT CancelTransaction ( [in] BSTR xmlDataSource,
                            [in] BSTR xmlTransaction);
```

### **Process Batch Transaction**

**Description:** Used to process batch transactions containing one or more individual transactions.

**Method Name:** ProcessBatchTransaction

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT ProcessBatchTransaction ( [in] BSTR xmlDataSource,  
                                  [in] BSTR xmlBatchTransaction);
```

### **Process Transaction File**

**Description:** Used to process transaction files containing one or more batch transactions.

**Method Name:** ProcessTransactionFile

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT ProcessTransactionFile ( [in] BSTR xmlDataSource,  
                                 [in] BSTR xmlTransactionFile);
```

### **Post Balance Transfer**

**Description:** Used to used transfer the balance of a single previously posted transaction from one account to another.

**Method Name:** PostBalanceTransfer

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT PostBalanceTransfer ([in] BSTR xmlDataSource,  
                             [in] BSTR xmlTransaction);
```

### **Cancel Balance Transfer**

**Description:** Used to to cancel a previous balance transfer transaction.

**Method Name:** CancelBalanceTransfer

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT CancelBalanceTransfer ([in] BSTR xmlDataSource,  
                               [in] BSTR xmlTransaction);
```

---

## Transfer Account Balance

**Description:** Used to post a balance transfer transaction for each outstanding (balance > 0) previously posted transaction for the account.

**Method Name:** TransferAccountBalance

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT TransferAccountBalance ([in] BSTR xmlDataSource,  
                                [in] BSTR xmlTransaction);
```

## Return Account Balance

**Description:** Used to cancel all previous balance transfer transactions for the account.

**Method Name:** ReturnAccountBalance

**Interface:** IMaintenance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Maintenance

**Syntax:**

```
HRESULT ReturnAccountBalance ([in] BSTR xmlDataSource,  
                               [in] BSTR xmlTransaction);
```

## Interface Arguments

The methods available in the Oracle Utilities Receivables Component Maintenance interface use the following arguments:

### **xmlDataSource Argument**

The xmlDataSource argument is an xml string that contains database connection and other related information. A DTD, xml example, and data element descriptions for this argument can be found on page 15-7 in the *Oracle Utilities Energy Information Platform Configuration Guide*.

### **xmlTransaction Argument**

The xmlTransaction argument is an xml string that contains data necessary to perform the appropriate function. A DTD and data element descriptions for this argument can be found on page 16-6. XML examples for this argument can be found on page 19-9.

### **xmlBatchRefund Argument**

The xmlBatchRefund argument is an xml string that contains batch refund data. An xml schema and data element descriptions for the xmlBatchRefund argument can be found on page 19-10.

### **xmlBatchTransaction Argument**

The xmlBatchTransaction argument is an xml string that contains data necessary to process a batch transaction. A DTD and data element descriptions for the xmlBatchTransaction argument can be found on page 19-13.

### **xmlTransactionFile Argument**

The xmlTransactionFile argument is an xml string that contains data necessary to process a transaction file. A DTD and data element descriptions for the xmlTransactionFile argument can be found on page 19-14.

# Input Values

XML examples used as input values for the Oracle Utilities Receivables Component Maintenance interface (IMaintenance) are provided below.

## xmlTransaction

### XML Examples - xmlTransaction

#### Post Payment Transfer

```
<TRANSACTION>
  <ACCOUNT UID="357"/>
  <RELATEDTRANSACTIONS>
    <TRANSACTION UID="1772"/>
  </RELATEDTRANSACTIONS>
</TRANSACTION>
```

#### Cancel Payment Transfer

```
<TRANSACTION UID="1742"/>
```

**Note:** The xmlTransaction arguments used for Cancel methods contains the minimal information required to identify the transaction. They can also include additional information as desired, such as Cancel Reason, Cancel Note, or other information.

#### Post Adjustment

```
<TRANSACTION>
  <ACCOUNT UID="361"/>
  <CHARGEORCREDIT>CR</CHARGEORCREDIT>
  <AMOUNT CURRENCY="USD">550.00</AMOUNT>
</TRANSACTION>
```

#### Cancel Adjustment

```
<TRANSACTION UID="1746"/>
```

#### Post Refund

```
<TRANSACTION>
  <ACCOUNT UID="334"/>
  <AMOUNT CURRENCY="USD">100.00</AMOUNT>
</TRANSACTION>
```

#### Cancel Refund

```
<TRANSACTION UID="1764"/>
```

#### Post WriteOff

```
<TRANSACTION>
  <ACCOUNT UID="364"/>
  <RELATEDTRANSACTION>"1743"</RELATEDTRANSACTION>
</TRANSACTION>
```

#### Cancel WriteOff

```
<TRANSACTION UID="1605"/>
```

#### WriteOff Account

```
<TRANSACTION>
  <ACCOUNT UID="364"/>
</TRANSACTION>
```

**WriteOn Account**

```
<TRANSACTION>
  <ACCOUNT UID="364"/>
</TRANSACTION>
```

**Cancel Transaction**

```
<TRANSACTION UID="1743"/>
```

**Post Balance Transfer**

```
<TRANSACTION>
  <TRANSACTIONTYPE>BALTXFR</TRANSACTIONTYPE>
  <ACCOUNT ACCOUNTID="12345"/>
  <RELATEDTRANSACTIONS>
    <TRANSACTION>
      <ACCOUNT ACCOUNTID="98765"/>
    </TRANSACTION>
  </RELATEDTRANSACTIONS>
</TRANSACTION>
```

**Cancel Balance Transfer**

```
<TRANSACTION UID="1764"/>
```

**Transfer Account Balance**

```
<TRANSACTION>
  <ACCOUNT UID="334"/>
</TRANSACTION>
```

**Return Account Balance**

```
<TRANSACTION>
  <ACCOUNT UID="364"/>
</TRANSACTION>
```

**xmlBatchRefund****XML Schema - xmlBatchRefund**

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/08/XMLSchema">
  <xsd:element name="BATCHREFUND" type="BATCHREFUNDTYPE" />
  <xsd:complexType name="BATCHREFUNDTYPE">
    <xsd:sequence>
      <xsd:element name="REFUND" type="REFUNDTYPE" minOccurs="0"
maxOccurs="unbounded"/>
      <xsd:element name="RTN" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="INSTITUTION" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="LASTCHECKNO" type="xsd:integer" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name=" NUMREFUNDS " type="xsd:integer" use ="optional">
    <xsd:attribute name=" TOTALAMOUNT " type="xsd:double" use ="optional"/>
    <xsd:attribute name=" NUMERRORS " type="xsd:integer" use ="optional">
    <xsd:attribute name=" MAXERRORS " type="xsd:integer" use ="optional">
    <xsd:attribute name="REFUNDWAITDAYS" type="xsd:integer" use ="optional"/>
    <xsd:attribute name="VOIDOFFSETDAYS" type="xsd:integer" use ="optional"/>
    <xsd:attribute name="WRITEOFFREASONCODE" type="xsd:string" use
="optional"/>
    <xsd:attribute name="FILENAME" type="xsd:string" use ="optional"/>
    <xsd:attribute name="REFUNDREASONCODE" type="xsd:string" use ="optional"/>
    <xsd:attribute name="ACCOUNTNO" type="xsd:string" use ="optional"/>
  </xsd:complexType>
  <xsd:complexType name=" REFUNDTYPE ">
    <xsd:sequence>
      <xsd:element name="ACCOUNTID" type="xsd:string"/>
      <xsd:element name="UIDREFUND" type="xsd:integer"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

```

        <xsd:element name="UIDTRANSACTION" type="xsd:integer" minOccurs="0"
maxOccurs="1"/>
        <xsd:element name="UIDACCOUNT" type="xsd:integer" minOccurs="0"
maxOccurs="1"/>
        <xsd:element name="UIDCHECKINGACCOUNT" type="xsd:integer" minOccurs="0"
maxOccurs="1"/>
        <xsd:element name="AMOUNT" type="xsd:double" minOccurs="0"
maxOccurs="1"/>
        <xsd:element name="CHECKNO" type="xsd:integer" minOccurs="0"
maxOccurs="1"/>
        <xsd:element name="ISSUEDATE" type="xsd:date" minOccurs="0"
maxOccurs="1"/>
        <xsd:element name="CASHEDDATE" type="xsd:date" minOccurs="0"
maxOccurs="1"/>
        <xsd:element name="REASON" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
        <xsd:element name="STATUS" type="STATUSTYPE" minOccurs="0"
maxOccurs="1"/>
        <xsd:element name="SENDTO" type="ADDRESS" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:simpleType name=" STATUSTYPE " >
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="ISSUED"/>
        <xsd:enumeration value="CASHED"/>
        <xsd:enumeration value="VOIDED"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="ADDRESS">
    <xsd:sequence>
        <xsd:element name="CUSTOMERNAME" type="xsd:string"/>
        <xsd:element name="ADDRESS1" type="xsd:string"/>
        <xsd:element name="ADDRESS2" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
        <xsd:element name="ADDRESS3" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
        <xsd:element name="CITY" type="xsd:string"/>
        <xsd:element name="STATE" type="xsd:string"/>
        <xsd:element name="COUNTY" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
        <xsd:element name="POSTALCODE" type="xsd:decimal"/>
    </xsd:sequence>
    <xsd:attribute name="COUNTRY" type="xsd:NMTOKEN" use="default" value="USA"/
>
</xsd:complexType>
</xsd: schema >

```

## Element Descriptions - xmlBatchRefund

Batch Refund Attributes:

**NUMREFUNDS:** Total number of refunds processed in the file. This is an output attribute

**TOTALAMOUNT:** Total amount of all refunds processed in the file. This is an output attribute.

Attributes:

**CURRENCY:** Currency code for the amount.

**NUMERRORS:** Total number of errors occurred while refunds processing. This is an output attribute

**MAXERRORS:** The maximum no of errors that are allowed. This is an optional input attribute for UpdateRefundStatus process.

**REFUNDWAITDAYS:** The no of days the unapplied outstanding credit has to remain on a closed account. This is a required input attribute for PostPendingRefunds process.

**REFUNDREASONCODE:** This is the reason code for which the refunds are being posted to the REFUND table. This is an optional input attribute for PostPendingRefunds process.

**FILENAME:** The user specified fully qualified file name in which xmlBatchRefund will be stored. This is an required input attribute for ProcessPendingRefunds process.

**ACCOUNTNO:** The financial institution's account no. This is a required input attribute for ProcessPendingRefunds process.

**VOIDOFFSETDAYS:** The no of days after which the issued check would become void. This is a required input attribute for ProcessTimeVoidedRefunds process.

**WRITEOFFREASONCODE:** The reason code for writing off the account. This is a required input attribute for ProcessTimeVoidedRefunds process.

**COUNTRY:** The optional country name for the address. The default is USA.

Elements:

**ACCOUNTID:** Alternate identifier for account.

**UIDREFUND:** Alternate unique identifier for refund.

**UIDTRANSACTION:** Alternate unique identifier for refund transaction.

**UIDACCOUNT:** Alternate unique identifier for account.

**UIDCHECKINGACCOUNT:** Alternate unique identifier for checking account.

**AMOUNT:** The amount being refunded. This is a required input field when updating the status.

Attributes:

**CURRENCY:** Currency code for the amount.

**CHECKNO:** The Check number in the checking account.

**ISSUEDATE:** The date on which the check was issued.

**CASHEDDATE:** The date on which the check was cashed.

**REASON:** The description of the refund reason.

**STATUS:** The status of the refund. The allowed states are ISSUED, CASHED. This is a required input field when updating the status.

**CUSTOMERNAME:** The name of the customer to whom the amount is being refunded.

**ADDRESS1, ADDRESS2, and ADDRESS3:** The house number, apartment number, and street name of the customer.

**CITY:** The name of the town where the customer lives.

**STATE:** The state in which the city is located.

**COUNTY:** The county in which the city is located.



## xmlBatchTransaction

### DTD - xmlBatchTransaction

```

<!DOCTYPE BATCHTRANSACTION
[
<!ELEMENT BATCHTRANSACTION (TRANSACTIONID?, REVENUEMONTH?,
TRANSACTIONFILE?, DATE?, NUMTRANSACTIONS?, AMOUNT?,
CANCELREVENUEMONTH?, CANCELREASONCODE?, CANCELNOTE?, MAXERRORS?,
TRANSACTION*)>
<!ATTLIST BATCHTRANSACTION
    UID CDATA          #IMPLIED
    BATCHNO CDATA      #IMPLIED
    CANCEL{TRUE|FALSE} "FALSE"
    RESTART (TRUE|FALSE) "FALSE">
<!ELEMENT TRANSACTIONID    (#PCDATA)>
<!ELEMENT REVENUEMONTH     (#PCDATA)>
<!ELEMENT TRANSACTIONFILE  EMPTY>
<!ATTLIST TRANSACTIONFILE
    NAMECDATA          #IMPLIED>
<!ELEMENT DATE             (#PCDATA)>
<!ELEMENT NUMTRANSACTIONS  (#PCDATA)>
<!ELEMENT AMOUNT           (#PCDATA)>
<!ELEMENT CANCELREVENUEMONTH (#PCDATA)>
<!ELEMENT CANCELREASONCODE (#PCDATA)>
<!ELEMENT CANCELNOTE       (#PCDATA)>
<!ELEMENT MAXERRORS        (#PCDATA)>
]>

```

### Element Descriptions - xmlBatchTransaction

Batch Transaction Attributes:

**UID:** Unique identifier for the transaction.

**BATCHNO:** Unique number of batch transaction within transaction file or source.

**CANCEL:** Indicates that the batch should be processed in cancel mode.

**RESTART:** Indicates that the batch should be processed in restart mode.

Elements:

**TRANSACTIONID:** Optional Transaction ID for transactions. Forwarded to individual transactions if not already provided.

**REVENUEMONTH:** Optional revenue month for transactions. Forwarded to individual transactions if not already provided.

**TRANSACTIONFILE:** Transaction file from where batch came. Optional for processing.

Attributes:

**NAME:** Unique file path name of transaction file.

**DATE:** Date of batch transaction. Optional for processing.

**NUMTRANSACTIONS:** Total number of transactions in the batch. Optional for processing.

**AMOUNT:** Total amount of all transactions in the batch. Optional for processing.

Attributes:

**CURRENCY:** Currency code for the amount.

**CANCELREVENUEMONTH:** Optional revenue month for cancelled transaction. If not provided then the current month will be used.

**CANCELREASONCODE:** Optional reason code for canceling transaction.

**CANCELNOTE:** Optional note associated with cancelled transaction.

**MAXERRORS:** Maximum number of transaction errors allowed prior to stopping the process. Optional for processing.

**TRANSACTION\*:** Individual transactions in the batch. Required for processing.

## xmlTransactionFile

### DTD - xmlTransactionFile

```
<!DOCTYPE TRANSACTIONFILE
[
<!ELEMENT TRANASCTIONFILE (TRANSACTIONID?, REVENUEMONTH?, DATE?,
NUMBATCHES?, NUMTRANSACTIONS?, AMOUNT?, CANCELREVENUEMONTH?,
CANCELREASONCODE?, CANCELNOTE?, MAXERRORS?, MAXERRORSPERBATCH?,
BATCHTRANSACTION*)>
<!ATTLIST TRANSACTIONFILE
    NAMECDATA          #IMPLIED
    CANCEL{TRUE|FALSE} "FALSE"
    RESTART (TRUE|FALSE) "FALSE">
<!ELEMENT TRANSACTIONID (#PCDATA)>
<!ELEMENT REVENUEMONTH (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT NUMBATCHES (#PCDATA)>
<!ELEMENT NUMTRANSACTIONS (#PCDATA)>
<!ELEMENT AMOUNT (#PCDATA)>
<!ELEMENT CANCELREVENUEMONTH (#PCDATA)>
<!ELEMENT CANCELREASONCODE (#PCDATA)>
<!ELEMENT CANCELNOTE (#PCDATA)>
<!ELEMENT MAXERRORS (#PCDATA)>
<!ELEMENT MAXERRORSPERBATCH (#PCDATA)>
]>
```

### Element Descriptions - xmlTransactionFile

Transaction File Attributes:

**NAME:** Unique file path name of transaction file. Required for processing.

**CANCEL:** Indicates that the file should be processed in cancel mode.

**RESTART:** Indicates that the file should be processed in restart mode.

Elements:

**TRANSACTIONID:** Optional Transaction ID for batch transactions. Forwarded to individual batch transactions if not already provided.

**REVENUEMONTH:** Optional revenue month for batch transactions. Forwarded to individual batch transactions if not already provided.

**DATE:** Optional date of transaction file. Forwarded to individual batch transactions if not already provided.

**NUMBATCHES:** Total number of batches in the file. Optional for processing.

**NUMTRANSACTIONS:** Total number of transactions in the file. Optional for processing.

**AMOUNT:** Total amount of all transactions (or batch transactions) in the file. Optional for processing.

Attributes:

**CURRENCY:** Currency code for the amount.

**CANCELREVENUEMONTH:** Optional revenue month for cancelled transaction. If not provided then the current month will be used.

**CANCELREASONCODE:** Optional reason code for canceling transaction.

**CANCELNOTE:** Optional note associated with cancelled transaction.

**MAXERRORS:** Maximum number of batch transaction errors allowed prior to stopping the process. Optional for processing.

**MAXERRORSPERBATCH:** Maximum number of transaction errors allowed per batch transaction prior to stopping the batch transactions process. Forwarded to individual batch transactions if not already provided. Optional for processing.

**BATCHTRANSACTION\*:** Individual batch transactions in the file. Required for processing.



# Chapter 20

---

## Oracle Utilities Receivables Component Collections Interface

This chapter describes the methods/functions available to external systems through the Oracle Utilities Receivables Component Collections Interface (ICollections) and the CollectionsObjects interface. These methods allow users to review the status of collection arrangements from external systems, and to create a process context for collections processes.

See **Chapter 9: Collections** for more information about these functions.

## Method, Interface, and Syntax

The methods, interface objects, and syntax for the Oracle Utilities Receivables Component Collections interface are as follows:

### Review Collection Arrangement

**Description:** Used to periodically review the terms of a Collection Arrangement to make sure the arrangement is not in DEFAULT.

**Method Name:** reviewColArrangement

**Interface:** IRemittance

**DLL Name:** LSACCT.DLL

**Program ID:** LSACCT.Remittance

**Syntax:**

```
HRESULT reviewColArrangement ([in] BSTR xmlDataSource,  
                              [in] BSTR xmlColArrangementIn);  
                              [out, retval] BSTR xmlColArrangementOut);
```

### Create Process Context

**Description:** Used to create process context values for collections processes initiated from external systems.

**Method Name:** createProcessContext

**Interface:** CollectionObjects

**DLL Name:** CollectionObjects.DLL

**Program ID:** CollectionObjects.ProcessXML

**Syntax:**

```
HRESULT createProcessContext ([in] BSTR xmlDataSource,  
                              [in] STR UIDProcess);  
                              [in] STR UIDAccount);  
                              [in] STR ContextVar);  
                              [out, retval] BSTR xmlContext);
```

### Collection Information

**Description:** Used to gather collections-related information for a specified account.

**Method Name:** CollectionInfo

**Interface:** CollectionObjects

**DLL Name:** CollectionObjects.DLL

**Program ID:** CollectionObjects.CollectionInfo

**Syntax:**

```
HRESULT CollectionInfo ([in] BSTR xmlDataSource,  
                       [in] STR UIDAccount);  
                       [out, retval] BSTR xmlCollInfo;
```

## Collection History Insert

**Description:** Inserts a record in the Collection History Table.

**Method Name:** ColHistoryInsert

**Interface:** CollectionObjects

**DLL Name:** CollectionObjects.DLL

**Program ID:** CollectionObjects.ColHistoryInsert

**Syntax:**

```
HRESULT ColHistoryInsert ([in] BSTR xmlDataSource,
                          [in] BSTR xmlTransaction);
                          [out, retval] BSTR xmlTransactionOut);
```

## Update Account Statuses

**Description:** Updates the Collections Status, Receiveable Status, or both values for a given Account

**Method Name:** UpdateAccount

**Interface:** CollectionObjects

**DLL Name:** CollectionObjects.DLL

**Program ID:** CollectionObjects.UpdateAccount

**Syntax:**

```
HRESULT ColHistoryInsert ([in] BSTR xmlDataSource,
                          [in] BSTR xmlAccount);
                          [out, retval] BSTR xmlAccountOut);
```

## Add Exemption

**Description:** Inserts an exemption into the Oracle Utilities Data Repository and triggers any Collections processing that should occur based on this exemption being created. This includes logging the activity to the Collection Object History table, suspending existing Workflow processes if the exemption causes the account to no longer be past due, and posting the necessary Workflow events to indicate an exemption has been created for the account. This function mimics what occurs when a user adds an exemption using the Collection Exemptions screens of the Oracle Utilities Receivables Component User Interface.

**Method Name:** Add

**Interface:** CollectionObjects

**DLL Name:** CollectionObjects.DLL

**Program ID:** CollectionObjects.Exemptions

**Syntax:**

```
HRESULT Add ([in] BSTR xmlDataSource,
             [in] BSTR xmlExemptionIn);
             [out, retval] BSTR xmlExemptionOut);
```

## Edit Exemption

**Description:** Modifies the data elements of an existing exemption. Either the UID or the logical key (Account (UID or ID), STARTDATE, and type) of the record that you wish to edit. In addition to modifying the appropriate database record, all other required Collections activity will be done as well. This includes checking to see if the account needs to be put in or taken out of collections based on the modifications made, and finally notify the Workflow engine that a modification has been made by posting the appropriate Workflow event type code. This function mimics what occurs when a user edits an exemption using Collection Exemptions screens of the Oracle Utilities Receivables Component User Interface.

**Method Name:** Edit

**Interface:** CollectionObjects

**DLL Name:** CollectionObjects.DLL

**Program ID:** CollectionObjects.Exemptions

**Syntax:**

```
HRESULT Edit ([in] BSTR xmlDataSource,  
             [in] BSTR xmlExemptionIn);  
            [out, retval] BSTR xmlExemptionOut);
```

## Delete Exemption

**Description:** Deletes an exemption record. This method accepts either a UID or the logical key of the exemption to be deleted. Once the exemption is deleted, the appropriate Collections processing occurs, including logging the action to the Collection Object History table for this account, checking to see if deleting this exemption will cause the account to go back into collections, and finally, notifying the workflow engine that this account has an exemption that has been deleted. Unlike when an exemption is added or edited, this is done by starting a workflow called Activity Event. This system workflow actually handles posting the workflow Activity Event to ensure that this happens after all the Collections workflows have been resumed and can properly process this event. This function mimics what occurs when a user deletes an exemption using the Collection Exemptions screens of the Oracle Utilities Receivables Component User Interface.

**Method Name:** Delete

**Interface:** CollectionObjects

**DLL Name:** CollectionObjects.DLL

**Program ID:** CollectionObjects.Exemptions

**Syntax:**

```
HRESULT Delete ([in] BSTR xmlDataSource,  
              [in] BSTR xmlExemptionIn);  
             [out, retval] BSTR xmlExemptionOut);
```

## Add Arrangement

**Description:** Inserts an arrangement into the Oracle Utilities Data Repository and triggers any Collections processing that should occur based on this arrangement being created. This includes logging the activity to the Collection Object History table, suspending existing workflow processes if the exemption created with the arrangement causes the account to no longer be past due, creating arrangement payments, and posting the necessary workflow events to indicate an exemption has been created for this account. This function mimics what occurs when a user adds an exemption using the Collection Arrangements screens of the Oracle Utilities Receivables Component User Interface.

**Method Name:** Add

**Interface:** CollectionObjects



**DLL Name:** CollectionObjects.DLL

**Program ID:** CollectionObjects.Arrangements

**Syntax:**

```
HRESULT Add ([in] BSTR xmlDataSource,
             [in] BSTR xmlArrangementIn);
           [out, retval] BSTR xmlArrangementOut);
```

## Edit Arrangement

**Description:** Modifies the data elements of an already existing arrangement. Either the UID or the logical key (Account (UID or ID), and STARTDATE) of of the arrangement that you wish to edit. In addition to modifying the appropriate arrangement record, any other required Collections activity will be done as well. This includes checking to see if the account needs to be put in or taken out of collections based on the modifications made, or to resume the workflow if the account is past due. This function mimics what occurs when a user edits an arrangement using Collection Arrangements screens of the Oracle Utilities Receivables Component User Interface.

**Method Name:** Edit

**Interface:** CollectionObjects

**DLL Name:** CollectionObjects.DLL

**Program ID:** CollectionObjects.Arrangements

**Syntax:**

```
HRESULT Edit ([in] BSTR xmlDataSource,
             [in] BSTR xmlArrangementIn);
           [out, retval] BSTR xmlArrangementOut);
```

## Delete Arrangement

**Description:** Deletes an arrangement. This method accepts either a UID or the logical key of an arrangement to be deleted. Once the arrangement is deleted, the appropriate Collections processing occurs, including logging the action to the Collection Object History table for this account, checking to see if deleting the arrangement's exemption will cause the account to go back into collections, and finally, notifying the workflow engine that this account has an exemption that has been deleted. Unlike when an arrangement is added or edited, this is done by starting a workflow called Activity Event. This system workflow actually handles posting the workflow Activity Event to ensure that this happens after all the Collections workflows have been resumed and can properly process this event. This function mimics what occurs when a user deletes an arrangement using the Collection Arrangements screens of the Oracle Utilities Receivables Component User Interface.

**Method Name:** Delete

**Interface:** CollectionObjects

**DLL Name:** CollectionObjects.DLL

**Program ID:** CollectionObjects.Arrangements

**Syntax:**

```
HRESULT Delete ([in] BSTR xmlDataSource,
               [in] BSTR xmlArrangementIn);
             [out, retval] BSTR xmlArrangementOut);
```

## Update Arrangement Status

**Description:** Updates the status of an arrangement, and if desired, the status of the arrangement's related exemption. This method accepts either a UID or the logical key of an arrangement to be updated.

**Method Name:** UpdateStatus

**Interface:** CollectionObjects

**DLL Name:** CollectionObjects.DLL

**Program ID:** CollectionObjects.Arrangements

**Syntax:**

```
HRESULT UpdateStatus ([in] BSTR xmlDataSource,  
                     [in] BSTR xmlArrangementIn);  
[out, retval] BSTR xmlArrangementOut);
```

## Add Payment

**Description:** Inserts an arrangement payment into the Oracle Utilities Data Repository and triggers any Collections processing that should occur based on the modification to the arrangement. This includes suspending existing workflow processes if the exemption created with the arrangement causes the account to no longer be past due, resuming suspended processes, and posting the necessary workflow events to indicate an exemption has been created for this account. This function mimics what occurs when a user adds an arrangement payment using the Collection Arrangements screens of the Oracle Utilities Receivables Component User Interface.

**Method Name:** Add

**Interface:** CollectionObjects

**DLL Name:** CollectionObjects.DLL

**Program ID:** CollectionObjects.Payments

**Syntax:**

```
HRESULT Add ([in] BSTR xmlDataSource,  
            [in] BSTR xmlPaymentsIn);  
[out, retval] BSTR xmlPaymentsOut);
```

## Edit Payment

**Description:** Modifies the data elements of an already existing arrangement payment. Either the UID or the logical key (Account (UID or ID), and STARTDATE) of the arrangement for which you wish to edit the payment must be provided. In addition to modifying the appropriate database record, all other required Collections activity will be done as well. This includes checking to see if the account needs to be put in or taken out of collections based on the modifications made, or to resume the workflow if the account is past due. This function mimics what occurs when a user edits an arrangement payment using Collection Arrangements screens of the Oracle Utilities Receivables Component User Interface.

**Method Name:** Edit

**Interface:** CollectionObjects

**DLL Name:** CollectionObjects.DLL

**Program ID:** CollectionObjects.Payments

**Syntax:**

```
HRESULT Edit ([in] BSTR xmlDataSource,  
             [in] BSTR xmlPaymentsIn);  
[out, retval] BSTR xmlPaymentsOut);
```

## Delete Payment

**Description:** Deletes an existing arrangement payment. This method accepts either a UID or the logical key of the arrangement related to the payment to be deleted. Once the payment is deleted, the appropriate Collections processing occurs. This includes checking to see if deleting the payment will cause the account to go back into collections. This function mimics what occurs when a user deletes an arrangement payment using the Collection Arrangements screens of the Oracle Utilities Receivables Component User Interface.

**Method Name:** Delete

**Interface:** CollectionObjects

**DLL Name:** CollectionObjects.DLL

**Program ID:** CollectionObjects.Payments

**Syntax:**

```
HRESULT Delete ([in] BSTR xmlDataSource,  
               [in] BSTR xmlPaymentsIn);  
               [out, retval] BSTR xmlPaymentsOut);
```

## Interface Arguments

The methods available in the Oracle Utilities Receivables Component Collections interface use the following arguments:

### **xmlDataSource Argument**

The xmlDataSource argument is an xml string that contains database connection and other related information. A DTD, xml example, and data element descriptions for this argument can be found on page 15-7 in the *Oracle Utilities Energy Information Platform Configuration Guide*.

### **xmlColArrangementIn Argument**

The xmlColArrangementIn argument is an xml string that contains the information necessary to identify a collections arrangement and associated payments. A DTD, xml example, and data element descriptions for this argument can be found on page 20-10.

### **xmlColArrangementOut Argument**

The xmlColArrangementOut argument is an xml string that contains return values from the Review Collection Arrangement method. A DTD, xml example, and data element descriptions for this argument can be found on page 20-17.

### **UIDProcess**

The UIDProcess argument is the UID of the process that you are creating the context for. This defines the context values to be created by the createProcessContext method.

### **UIDAccount**

The UIDAccount argument is the UID of the account for which the collections process is being initiated.

### **ContextVar**

The ContextVar argument is a semi-colon delimited string value that contains the values needed for any queries used to populate the context values created by the createProcessContext method. If values are to be populated with literals, this argument defines how these literals are passed to the function.

For example, to populate the process context for the Active/Inactive Collections processes, the ContextVar value might be "UIDACCOUNT= 23;". For processes where the process begins at the third activity in the process, the ContextVar might be "UIDACCOUNT= 23;ACTIVITYSTART=3=LITERAL".

In both examples, the UIDAccount is passed in as a value that is used to retrieve other values from the Context Value and Process Context Value tables. In the second example, the ACTIVITYSTART value is used as is in the ContextVar string because the string 'LITERAL' is included after the actual ACTIVITYSTART value. This flexibility provides additional means for dynamically creating process context xml strings with literal values, values from the database based on parameters passed into the function, or a mixture of the two.

### **xmlContext**

The xmlContext output is an xml string that contains context values based on the process initiated. The specific elements in the xmlContext string are based on the records in the Context Value and Process Context Value tables. See **Chapter 12: Setting Up Workflow Management Database Tables** for more information about setting up and configuring these tables.

### **xmlCollInfo Argument**

The xmlCollInfo argument is an xml string that contains return values from the Collection Information method. An xml example of this argument can be found on page 20-17.

**xmlTransaction Argument**

The xmlTransaction argument is an xml string that contains data necessary to perform the appropriate function. A DTD and data element descriptions for this argument can be found on page 16-6.

**xmlTransactionOut Argument**

The xmlTransactionOut argument is an xml string that contains return values from the Collection History Insert method. An xml example of this argument can be found on page 20-18.

**xmlAccount Argument**

The xmlAccount argument is an xml string that contains information about how the account is to be updated. An xml example for this argument can be found on page 20-11.

**xmlAccountOut Argument**

The xmlAccountOut argument is an xml string that contains return values from the Update Account Statuses method. An xml example of this argument can be found on page 20-18.

**xmlExemptionIn Argument**

The xmlExemptionIn argument is an xml string that contains information about an exemption. Xml examples and data element descriptions for this argument can be found on page 20-11.

**xmlExemptionOut Argument**

The xmlExemptionOut argument is an xml string that contains return values from the Add Exemption, Edit Exemption, and Delete Exemption methods. Xml examples and data element descriptions of this argument can be found on page 20-18.

**xmlArrangementIn Argument**

The xmlArrangementIn argument is an xml string that contains information about an arrangement. Xml examples for this argument can be found on page 20-11.

**xmlArrangementOut Argument**

The xmlArrangementOut argument is an xml string that contains return values from the Add Arrangement, Edit Arrangement, and Delete Arrangement methods. Xml examples of this argument can be found on page 20-18.

**xmlPaymentIn Argument**

The xmlPaymentIn argument is an xml string that contains information about an arrangement payment. Xml examples for this argument can be found on page 20-11.

**xmlPaymentOut Argument**

The xmlPaymentOut argument is an xml string that contains return values from the Add Payment, Edit Payment, and Delete Payment methods. Xml examples of this argument can be found on page 20-18.

## Input Values

The Data Type Definition (DTD), xml example, and data element descriptions for the input values of the Oracle Utilities Receivables Component Collections interface are provided below.

### xmlColArrangementIn

#### DTD - xmlColArrangementIn

```
<!DOCTYPE COLARRANGEMENT
[
<!ELEMENT COLARRANGEMENT
  (UIDCOLARRANGEMENT?, UIDACCOUNT?, STARTDATE?, STOPDATE?,
  NUMPAYMENTS?, TOTALAMOUNT?, STATUSCODE?, MODTIME?, DOCTIME?,
  COLARRANGEMENTPAYMENT+) >
<!ELEMENT UIDCOLARRANGEMEN (CDATA) >
<!ELEMENT UIDACCOUNT (CDATA) >
<!ELEMENT STARTDATE (CDATA) >
<!ELEMENT STOPDATE (CDATA) >
<!ELEMENT NUMPAYMENTS (CDATA) >
<!ELEMENT TOTALAMOUNT (CDATA) >
<!ELEMENT STATUSCODE (CDATA) >
<!ELEMENT MODTIME (CDATA) >
<!ELEMENT DOCTIME (CDATA) >
<!ELEMENT COMPLIANCETIME (CDATA) >
<!COLARRANGEMENTPAYMENT
  (UIDCOLARRANGEMENT?, DUEDATE, DUEAMOUNT) >
<!ELEMENT UIDCOLARRANGEMENT (CDATA) >
<!ELEMENT DUEDATE (CDATA) >
<!ELEMENT DUEAMOUNT (CDATA) >
]>
```

#### XML Example - xmlColArrangementIn

```
<COLARRANGEMENT>
  <UIDACCOUNT>25571</UIDACCOUNT>
  <STARTDATE>2001-05-05</STARTDATE>
  <STOPDATE>2001-08-05</STOPDATE>
  <NUMPAYMENTS>3</NUMPAYMENTS>
  <TOTALAMOUNT CURRENCY="USD">300.00</TOTALAMOUNT>
  <STATUSCODE>CURRENT</STATUSCODE>
  <COMPLIANCEDATE>2001-08-05</COMPLIANCEDATE>
  <COLARRANGEMENTPAYMENT>
    <DUEDATE>2001-06-05</DUEDATE>
    <DUEAMOUNT CURRENCY="USD">100.00</DUEAMOUNT>
  </COLARRANGEMENTPAYMENT>
  <COLARRANGEMENTPAYMENT>
    <DUEDATE>2001-07-05</DUEDATE>
    <DUEAMOUNT CURRENCY="USD">100.00</DUEAMOUNT>
  </COLARRANGEMENTPAYMENT>
  <COLARRANGEMENTPAYMENT>
    <DUEDATE>2001-08-05</DUEDATE>
    <DUEAMOUNT CURRENCY="USD">100.00</DUEAMOUNT>
  </COLARRANGEMENTPAYMENT>
</COLARRANGEMENT>
```

#### Element Descriptions - xmlColArrangementIn

The use of each individual attribute and element in the xmlColArrangement argument is described below.

Elements:

**UIDCOLARRANGEMENT:** Optional Unique ID for the Collection Arrangement

**UIDACCOUNT:** Unique ID for the Account

**STARTDATE:** Start Date for the Collection Arrangement

**STOPDATE:** Stop Date for the Collection Arrangement

**NUMPAYMENTS:** Number of Payments associated with this Collection Arrangement.

**TOTALAMOUNT:** Total Amount of all Collection Arrangement Payments associated with this Collection Arrangement.

Attributes:

**CURRENCY:** Currency code for the amount.

**STATUSCODE:** Status Code for the Collection Arrangement. Valid values for STATUSCODE include CURRENT, DEFAULT, COMPLETE

**MODTIME:** Modification Date for the Collection Arrangement

**DOCTIME:** Optional Documentation Date for the Collection Arrangement

**COMPLIANCEDATE:** Optional Compliance Date to be used by reviewColArrangement().

**COLARRANGEMENTPAYMENT:** Payment Schedule Information for the Collection Arrangement

Elements:

**UIDCOLARRANGEMENT:** Unique ID for the Collection Arrangement associated to the payments.

**DUEDATE:** Due Date for the Collection Arrangement Payment

**DUEAMOUNT:** Due Amount for the Collection Arrangement Payment

Attributes:

**CURRENCY:** Currency code for the amount.

## xmlAccount

### XML Example - xmlAccount

```
<ACCOUNT>
  <UIDACCOUNT>25571</UIDACCOUNT>
  <UPDATETYPE>RECEIVABLE</UPDATETYPE>
  <RECEIVABLESTATUS>COLLECTIONS</RECEIVABLESTATUS>
</ACCOUNT>
```

## xmlExemptionIn

### XML Example - xmlExemptionIn - Add

```
<EXEMPTION SOURCE="LODESTAR CCS API">
  <UIDACCOUNT>81</UIDACCOUNT>
  <ACCOUNTID/>
  <STARTDATE>01/05/2004</STARTDATE>
  <STOPDATE/>
  <COLEXEMPTYPECODE/>
  <UIDCOLEXEMPTYPE/>
  <NOTE/>
  <EXEMPTIONSTATUS/>
  <EXEMPTAMT/>
</EXEMPTION>
```

### XML Example - xmlExemptionIn - Edit

```
<EXEMPTION SOURCE="LODESTAR CCS API">
  <UIDACCOUNT>81</UIDACCOUNT>
  <ACCOUNTID/>
  <STARTDATE>01/05/2004</STARTDATE>
  <NEWSTARTDATE/>
  <STOPDATE/>
  <NEWCOLEXEMPTYPECODE/>
  <COLEXEMPTYPECODE/>
  <NEWUIDCOLEXEMPTYPE/>
  <UIDCOLEXEMPTYPE>1</UIDCOLEXEMPTYPE>
  <NOTE/>
  <EXEMPTIONSTATUS/>
  <EXEMPTAMT/>
</EXEMPTION>
```

### XML Example - xmlExemptionIn - Delete

```
<EXEMPTION SOURCE="LODESTAR CCS API">
  <UIDCOLEXEMPTION/>
  <UIDACCOUNT>81</UIDACCOUNT>
  <ACCOUNTID/>
  <STARTDATE/>
  <COLEXEMPTYPECODE/>
  <UIDCOLEXEMPTYPE/>
</EXEMPTION>
```

### Element Descriptions - xmlExemptionIn

The use of each individual attribute and element in the xmlExemptionIn argument is described below.

Elements:

**EXEMPTION:** Root element that contains information about an exemption.

Attributes:

**SOURCE:** The value that will be logged to the Source column in the Collection Object History table when this method is executed. If not provided it will default to "LODESTAR CCS API".

**UIDACCOUNT:** Unique ID of the account related to the exemption. Required if ACCOUNTID not provided.

**ACCOUNTID:** Account ID of the account related to the exemption. Required if UIDACCOUNT not provided.

**STARTDATE:** Date exemption is to start. Required

**STOPDATE:** Optional. If not provided this exemption will never expire



**COLEXEMPTYPECODE:** Code for the exemption type, from the Exemption Type table in the Oracle Utilities Data Repository. Required if no UIDCOLEXEMPTYPE is provided.

**UIDCOLEXEMPTYPE:** Unique ID of the exemption type, from the Exemption Type table in the Oracle Utilities Data Repository. Required if no COLEXEMPTYPECODE is provided.

**NOTE:** Optional note, can be up to 254 characters in length.

**EXEMPTIONSTATUS:** Optional. The current status of the exemption. Defaults to "CURRENT" if not is provided. Valid statuses are "CURRENT" and "EXPIRED".

**EXEMPTAMT:** Optional. The maximum amount that account will be exempt for. If left blank, the account will be fully exempt and will never enter collections.

**UIDCOLEXEMPTION:** Unique ID of an existing exemption. Can be used with the Edit Exemption and Delete Exemption methods. Required if the UIDACCOUNT or logical key (comprised of ACCOUNTID, COLEXEMPTYPECODE or UIDCOLEXEMPTYPE, and STARTDATE) are not provided.

**NEWSTARTDATE:** New start date for the exemption. Required to change the start time using the Edit Exemption method.

**NEWCOLEXEMPTYPECODE:** Optional. New code for the exemption type, from the Exemption Type table in the Oracle Utilities Data Repository. Required to change the exemption type using the Edit Exemption method if no NEWUIDCOLEXEMPTYPE is provided.

**NEWUIDCOLEXEMPTYPE:** Optional. New unique ID the exemption type, from the Exemption Type table in the Oracle Utilities Data Repository. Required to change the exemption type using the Edit Exemption if no NEWCOLEXEMPTYPECODE is provided.

## xmlArrangementIn

### XML Example - xmlArrangementIn - Add

```
<COLARRANGEMENTS SOURCE="LODESTAR CCS API">
  <COLARRANGEMENT>
    <UIDACCOUNT>200</UIDACCOUNT>
    <STARTDATE>1/1/04</STARTDATE>
    <NUMBEROFPAYMENTS>2</NUMBEROFPAYMENTS>
    <FIRSTPAYMENTDATE>July 5, 2004</FIRSTPAYMENTDATE>
    <TOTALAMOUNT>-500</TOTALAMOUNT>
    <ARRANGEMENTTYPE>Collections Payment Plan</ARRANGEMENTTYPE>
    <PAYMENTS>
      <PAYMENT>
        <DUEDATE>01/04/2004</DUEDATE>
        <DUEAMOUNT>60</DUEAMOUNT>
      </PAYMENT>
      <PAYMENT>
        <DUEDATE>1-5-04</DUEDATE>
        <DUEAMOUNT>62</DUEAMOUNT>
      </PAYMENT>
    </PAYMENTS>
  </COLARRANGEMENT>
</COLARRANGEMENTS>
```

### XML Example - xmlArrangementIn - Edit

```
<COLARRANGEMENTS SOURCE="LODESTAR CCS API">
  <COLARRANGEMENT>
    <ACCOUNTID>1234567</ACCOUNTID>
    <STARTDATE>1/01/04</STARTDATE>
    <NEWSTARTDATE>1/5/2004</NEWSTARTDATE>
    <NUMBEROFPAYMENTS>3</NUMBEROFPAYMENTS>
    <TOTALAMOUNT>500</TOTALAMOUNT>
    <FIRSTPAYMENTDATE/>
    <ARRANGEMENTTYPE>Collections Payment Plan</ARRANGEMENTTYPE>
  </COLARRANGEMENT>
</COLARRANGEMENTS>
```

```

    </COLARRANGEMENT>
  </COLARRANGEMENTS>

```

### XML Example - xmlArrangementIn - Delete

```

<COLARRANGEMENTS SOURCE="LODESTAR CCS API">
  <COLARRANGEMENT>
    <ACCOUNTID>1234567</ACCOUNTID>
    <STARTDATE>1/5/04</STARTDATE>
  </COLARRANGEMENT>
</COLARRANGEMENTS>

```

### XML Example - xmlArrangementIn - UpdateStatus

```

<COLARRANGEMENTS>
  <COLARRANGEMENT>
    <UIDCOLARRANGEMENT>23</UIDCOLARRANGEMENT>
    <ARRANGEMENTSTATUS>CURRENT</ARRANGEMENTSTATUS>
  </COLARRANGEMENT>
</COLARRANGEMENTS>

```

### Element Descriptions - xmlArrangementIn

The use of each individual attribute and element in the xmlArrangementIn argument is described below.

Elements:

**COLARRANGEMENTS:** Root element containing a COLARRANGEMENT element.

Attributes:

**SOURCE:** The value that will be logged to the Source column in the Collection Object History table when this method is executed. If not provided it will default to "LODESTAR CCS API".

**COLARRANGEMENT:** Element containing a single arrangement. Note: Only one COLARRANGEMENT element can be present.

**UIDACCOUNT:** Unique ID of the account related to the arrangement. Required if ACCOUNTID not provided (unless UIDCOLARRANGEMENT is provided).

**ACCOUNTID:** Account ID of the account related to the arrangement. Required if UIDACCOUNT not provided (unless UIDCOLARRANGEMENT is provided).

**STARTDATE:** Date arrangement is to start. Required for the Add Arrangement method. Required for Edit Arrangement and Delete Arrangement methods, unless UIDCOLARRANGEMENT is provided.

**NUMBEROFPAYMENTS:** The number of payments in the arrangement.

**FIRSTPAYMENTDATE:** Optional. The date on which the first payment in the arrangement is due. If not provided, the default is the STARTDATE of the arrangement.

**TOTALAMOUNT:** Optional. The total amount of the arrangement. If not provided, the default is the total collectible balance for the account.

**ARRANGEMENTTYPE:** The arrangement type, from the Arrangement Type table in the Oracle Utilities Data Repository.

**PAYMENTS:** Optional. Element containing one or more arrangement payments. If specified, specified payments are created. If not provided, payments are automatically created based on the FIRSTPAYMENTDATE, TOTALAMOUNT, and NUMBEROFPAYMENTS elements.

Elements:

**PAYMENT:** Element containing Due Date and Amount Due for a single payment.

**DUEDATE:** Due date for the payment.

**DUEAMOUNT:** Amount due for the payment.

**UIDCOLARRANGEMENT:** Unique ID of an existing arrangement. Can be used with the Edit Arrangement and Delete Arrangement methods. Required if the UIDACCOUNT or logical key (comprised of (ACCOUNTID or UIDACCOUNT) and STARTDATE) is not provided.

**NEWSTARTDATE:** New start date for the arrangement. Required to change the start time using the Edit Arrangement method.

## xmlPaymentIn

### XML Example - xmlPaymentIn - Add

```
<COLARRANGEMENTS LANGUAGE="ENU" SOURCE="LODESTAR CCS API">
  <COLARRANGEMENT>
    <UIDACCOUNT>200</UIDACCOUNT>
    <STARTDATE>1/05/04</STARTDATE>
    <PAYMENTS>
      <PAYMENT>
        <DUEDATE>6/25/2004</DUEDATE>
        <DUEAMOUNT>2</DUEAMOUNT>
      </PAYMENT>
      <PAYMENT>
        <DUEDATE>7/30/04</DUEDATE>
        <DUEAMOUNT>35</DUEAMOUNT>
      </PAYMENT>
    </PAYMENTS>
  </COLARRANGEMENT>
</COLARRANGEMENTS>
```

### XML Example - xmlPaymentIn - Edit

```
<COLARRANGEMENTS LANGUAGE="ENU" SOURCE="LODESTAR CCS API">
  <COLARRANGEMENT>
    <UIDCOLARRANGEMENT>1143</UIDCOLARRANGEMENT>
    <PAYMENTS>
      <PAYMENT>
        <DUEDATE>6/25/2004</DUEDATE>
        <NEWDUEDATE>6/27/04</NEWDUEDATE>
        <DUEAMOUNT>90</DUEAMOUNT>
      </PAYMENT>
      <PAYMENT>
        <DUEDATE>7/30/04</DUEDATE>
        <DUEAMOUNT>95</DUEAMOUNT>
      </PAYMENT>
    </PAYMENTS>
  </COLARRANGEMENT>
</COLARRANGEMENTS>
```

### XML Example - xmlPaymentIn - Delete

```
<COLARRANGEMENTS LANGUAGE="ENU" SOURCE="LODESTAR CCS API">
  <COLARRANGEMENT>
    <UIDACCOUNT>200</UIDACCOUNT>
    <STARTDATE>1/05/04</STARTDATE>
    <PAYMENTS>
      <PAYMENT>
        <DUEDATE>6/27/2004</DUEDATE>
      </PAYMENT>
    </PAYMENTS>
  </COLARRANGEMENT>
</COLARRANGEMENTS>
```

### Element Descriptions - xmlPaymentIn

The use of each individual attribute and element in the xmlPaymentIn argument is described below.

Elements:

**COLARRANGEMENTS:** Root element containing a COLARRANGEMENT element.

Attributes:

**SOURCE:** The value that will be logged to the Source column in the Collection Object History table when this method is executed. If not provided it will default to "LODESTAR CCS API".

**COLARRANGEMENT:** Element containing a single arrangement. Note: Only one COLARRANGEMENT element can be present.

**UIDACCOUNT:** Unique ID of the account related to the arrangement. Required if ACCOUNTID and STARTDATE or UIDCOLARRANGEMENT not provided.

**ACCOUNTID:** Account ID of the account related to the arrangement. Required if UIDACCOUNT or UIDCOLARRANGEMENT not provided.

**STARTDATE:** Date arrangement is to start. Required if UIDACCOUNT or UIDCOLARRANGEMENT not provided.

**UIDCOLARRANGEMENT:** Unique ID of an existing arrangement. Required if the UIDACCOUNT or logical key (comprised of (ACCOUNTID or UIDACCOUNT) and STARTDATE) is not provided.

**PAYMENTS:** Element containing one or more arrangement payments.

Elements:

**PAYMENT:** Element containing Due Date, Amount Due, and optional New Due Date for a single payment.

Elements:

**DUEDATE:** Due date for the payment. Required.

**DUEAMOUNT:** Amount due for the payment. Required for the Add Payment method, optional for the Edit Payment method.

**NEWDUEDATE:** New due date for the payment. Used with the Edit Payment method.

## Return Values

This section describes the data returned from the Oracle Utilities Receivables Component Collections interface.

### xmlColArrangementOut

The DTD, and data elements for the xmlColArrangementOut argument are the same as the xmlColArrangementIn argument.

### xmlContext

#### XML Example - xmlContext - Pass

```
<PROCESSINSTANCE>
  <UID>1</UID>
  <PROCESSVERSION>41</PROCESSVERSION>
  <ACCOUNT>81</ACCOUNT>
  <CONTEXT>
    <JURISCODE>TX</JURISCODE>
    <OPCODE>LODESTAR</OPCODE>
    <REVENUECODE>R</REVENUECODE>
    <SPCLHANDLECODE/>
  </CONTEXT>
  <STATUS>PASS</STATUS>
</PROCESSINSTANCE>
```

#### XML Example - xmlContext - Error

```
<PROCESSINSTANCE>
  <UID>1</UID>
  <PROCESSVERSION>41</PROCESSVERSION>
  <ACCOUNT>81</ACCOUNT>
  <CONTEXT>
    <JURISCODE>TX</JURISCODE>
    <OPCODE>LODESTAR</OPCODE>
    <REVENUECODE>R</REVENUECODE>
    <SPCLHANDLECODE/>
  </CONTEXT>
  <STATUS>
    <ERRORS>
      <ERROR NUMBER="91" DESC="Object variable or With block variable not set"
SOURCE="BuildXMLFrom">
        <PARAMETERS>
          <PARAMETER><![CDATA[]]></PARAMETER>
          <PARAMETER><![CDATA[]]></PARAMETER>
        </PARAMETERS>
      </ERROR>
    </ERRORS>
  </STATUS>
</PROCESSINSTANCE>
```

### xmlCollInfo

#### XML Example - xmlCollInfo

```
<COLLECTIONINFO>
  <COLLECTIONSTATUS />
  <RECEIVABLESTATUS>COLLECTIONS</RECEIVABLESTATUS>
  <TOTALAMOUNTDUE>142</TOTALAMOUNTDUE>
  <PASTDUEAMOUNT>142</PASTDUEAMOUNT>
  <TOTALEXEMPTAMOUNT>42</TOTALEXEMPTAMOUNT>
  <COLLECTABLEBALANCE>100</COLLECTABLEBALANCE>
  <LASTPAYMENTDATE>10/01/2002</LASTPAYMENTDATE>
```

```
<LASTPAYMENTAMOUNT>50</LASTPAYMENTAMOUNT>
</COLLECTIONINFO>
```

## xmlTransactionOut

### XML Example - xmlTransactionOut

```
<TRANSACTION>
  <UIDACCOUNT>81</UIDACCOUNT>
  <TYPE>TYPE</TYPE>
  <NAME>NAME</NAME>
  <SOURCE>Test</SOURCE>
  <MESSAGE>MESSAGE</MESSAGE>
  <STATUS>PASS</STATUS>
</TRANSACTION>
```

## xmlAccountOut

### XML Example - xmlAccountOut

```
<ACCOUNT>
  <UIDACCOUNT>81</UIDACCOUNT>
  <COLLECTIONSTATUS>200</COLLECTIONSTATUS>
  <RECEIVABLESTATUS />
  <UPDATETYPE>BOTH</UPDATETYPE>
  <STATUS>PASS</STATUS>
</ACCOUNT>
```

## xmlExemptionOut

### XML Example - xmlExemptionOut - Pass

```
<EXEMPTION SOURCE="LODESTAR CCS API">
  <UIDACCOUNT>81</UIDACCOUNT>
  <ACCOUNTID/>
  <STARTDATE>01/06/2004</STARTDATE>
  <STOPDATE/>
  <COLEXEMPTYPECODE/>
  <UIDCOLEXEMPTYPE>1</UIDCOLEXEMPTYPE>
  <NOTE>1</NOTE>
  <EXEMPTIONSTATUS/>
  <EXEMPTAMT/>
  <STATUS>PASS</STATUS>
</EXEMPTION>
```

### XML Example - xmlExemptionOut - Fail

```
<EXEMPTION SOURCE="LODESTAR CCS API">
  <UIDACCOUNT>81</UIDACCOUNT>
  <ACCOUNTID/>
  <STARTDATE>01/05/2004</STARTDATE>
  <STOPDATE/>
  <COLEXEMPTYPECODE/>
  <UIDCOLEXEMPTYPE>1</UIDCOLEXEMPTYPE>
  <NOTE>1</NOTE>
  <EXEMPTIONSTATUS/>
  <EXEMPTAMT/>
  <STATUS>ERROR</STATUS>
  <ERRORS>
    <ERROR NUMBER="429" DESC="ActiveX component can't create object"
SOURCE="CreateConnection">
      <PARAMETERS>
        <PARAMETER>
          <DATASOURCE>
            <NAME/>
```

```

        <CONNECTSTRING>UID=otsedom;PWD=otsedom;DSN=vega</CONNECTSTRING>
        <USERID>otsedom</USERID>
        <PASSWORD>otsedom</PASSWORD>
        <QUALIFIER>otsedom</QUALIFIER>
    </DATASOURCE>
</PARAMETER>
</PARAMETERS>
</ERROR>
</ERRORS>
</EXEMPTION>

```

## Element Descriptions - xmlExemptionOut

Data elements for the xmlExemptionOut argument are the same as the xmlExemptionIn argument, with the following additions.

Elements:

**STATUS:** The status of the operation. Can be either “PASS” or “ERROR”.

**ERRORS:** Element containing one or more individual errors, each within an ERROR element.

Elements:

**ERROR:** An individual error.

Attributes:

**NUMBER:** The error number.

**SOURCE:** The source of the error.

Elements:

**PARAMETERS:** Element containing one or more parameters (each within a PARAMETER element) defining specific parameters related to the error.

## xmlArrangementOut

### XML Example - xmlArrangementOut - Pass

```

<COLARRANGEMENTS LANGUAGE="ENU" SOURCE="LODESTAR CCS API">
  <COLARRANGEMENT>
    <UIDACCOUNT>200</UIDACCOUNT>
    <STARTDATE>1/1/04</STARTDATE>
    <NUMBEROFPAYMENTS>2</NUMBEROFPAYMENTS>
    <FIRSTPAYMENTDATE>July 5, 2004</FIRSTPAYMENTDATE>
    <TOTALAMOUNT>-500</TOTALAMOUNT>
    <ARRANGEMENTTYPE>Collections Payment Plan</ARRANGEMENTTYPE>
    <PAYMENTS>
      <PAYMENT>
        <DUEDATE>01/04/2004</DUEDATE>
        <DUEAMOUNT>60</DUEAMOUNT>
      </PAYMENT>
      <PAYMENT>
        <DUEDATE>1-5-04</DUEDATE>
        <DUEAMOUNT>62</DUEAMOUNT>
      </PAYMENT>
    </PAYMENTS>
  </COLARRANGEMENT>
  <STATUS>PASS</STATUS>
</COLARRANGEMENTS>

```

### XML Example - xmlArrangementOut - Fail

```

<COLARRANGEMENTS LANGUAGE="ENU" SOURCE="LODESTAR CCS API">
  <COLARRANGEMENT>
    <UIDACCOUNT>200</UIDACCOUNT>
    <STARTDATE>1/1/04</STARTDATE>

```

```

<NUMBEROFPAYMENTS>2</NUMBEROFPAYMENTS>
<FIRSTPAYMENTDATE>July 5, 2004</FIRSTPAYMENTDATE>
<TOTALAMOUNT>-500</TOTALAMOUNT>
<ARRANGEMENTTYPE>Collections Payment Plan</ARRANGEMENTTYPE>
<PAYMENTS>
  <PAYMENT>
    <DUEDATE>01/04/2004</DUEDATE>
    <DUEAMOUNT>60</DUEAMOUNT>
  </PAYMENT>
  <PAYMENT>
    <DUEDATE>1-5-04</DUEDATE>
    <DUEAMOUNT>62</DUEAMOUNT>
  </PAYMENT>
</PAYMENTS>
</COLARRANGEMENT>
<STATUS>ERROR</STATUS>
<ERRORS>
  <ERROR NUMBER="429" DESC="ActiveX component can't create object"
SOURCE="CreateConnection">
    <PARAMETERS>
      <PARAMETER>
        <DATASOURCE>
          <NAME/>
          <CONNECTSTRING>UID=otsedom;PWD=otsedom;DSN=vega</CONNECTSTRING>
          <USERID>otsedom</USERID>
          <PASSWORD>otsedom</PASSWORD>
          <QUALIFIER>otsedom</QUALIFIER>
        </DATASOURCE>
      </PARAMETER>
    </PARAMETERS>
  </ERROR>
</ERRORS>
</COLARRANGEMENTS>

```

### XML Example - xmlArrangementOut - UpdateStatus

```

<COLARRANGEMENTS>
  <COLARRANGEMENT>
    <UIDCOLARRANGEMENT>23</UIDCOLARRANGEMENT>
    <ARRANGEMENTSTATUS>CURRENT</ARRANGEMENTSTATUS>
  </COLARRANGEMENT>
  <STATUS>PASS</STATUS>
</COLARRANGEMENTS>

```

### Element Descriptions - xmlArrangementOut

Data elements for the xmlArrangementOut argument are the same as the xmlArrangementIn argument, with the following additions.

Elements:

**STATUS:** The status of the operation. Can be either “PASS” or “ERROR”.

**ERRORS:** Element containing one or more individual errors, each within an ERROR element.

Elements:

**ERROR:** An individual error.

Attributes:

**NUMBER:** The error number.

**SOURCE:** The source of the error.

Elements:

**PARAMETERS:** Element containing one or more parameters (each within a PARAMETER element) defining specific parameters related to the error.



## xmlPaymentOut

### XML Example - xmlPaymentOut - Pass

```
<COLARRANGEMENTS LANGUAGE="ENU" SOURCE="LODESTAR CCS API">
  <COLARRANGEMENT>
    <UIDACCOUNT>200</UIDACCOUNT>
    <STARTDATE>1/05/04</STARTDATE>
    <PAYMENTS>
      <PAYMENT>
        <DUEDATE>6/25/2004</DUEDATE>
        <DUEAMOUNT>2</DUEAMOUNT>
      </PAYMENT>
      <PAYMENT>
        <DUEDATE>7/30/04</DUEDATE>
        <DUEAMOUNT>35</DUEAMOUNT>
      </PAYMENT>
    </PAYMENTS>
  </COLARRANGEMENT>
  <STATUS>PASS</STATUS>
</COLARRANGEMENTS>
```

### XML Example - xmlPaymentOut - Fail

```
<COLARRANGEMENTS LANGUAGE="ENU" SOURCE="LODESTAR CCS API">
  <COLARRANGEMENT>
    <UIDACCOUNT>200</UIDACCOUNT>
    <STARTDATE>1/05/04</STARTDATE>
    <PAYMENTS>
      <PAYMENT>
        <DUEDATE>6/25/2004</DUEDATE>
        <DUEAMOUNT>2</DUEAMOUNT>
      </PAYMENT>
      <PAYMENT>
        <DUEDATE>7/30/04</DUEDATE>
        <DUEAMOUNT>35</DUEAMOUNT>
      </PAYMENT>
    </PAYMENTS>
  </COLARRANGEMENT>
  <STATUS>ERROR</STATUS>
  <ERRORS>
    <ERROR NUMBER="429" DESC="ActiveX component can't create object"
SOURCE="CreateConnection">
      <PARAMETERS>
        <PARAMETER>
          <DATASOURCE>
            <NAME/>
            <CONNECTSTRING>UID=otsedom;PWD=otsedom;DSN=vega</CONNECTSTRING>
            <USERID>otsedom</USERID>
            <PASSWORD>otsedom</PASSWORD>
            <QUALIFIER>otsedom</QUALIFIER>
          </DATASOURCE>
        </PARAMETER>
      </PARAMETERS>
    </ERROR>
  </ERRORS>
</COLARRANGEMENTS>
```

### Element Descriptions - xmlPaymentOut

Data elements for the xmlPaymentOut argument are the same as the xmlPaymentIn argument, with the following additions.

Elements:

**STATUS:** The status of the operation. Can be either “PASS” or “ERROR”.

**ERRORS:** Element containing one or more individual errors, each within an ERROR element.

Elements:

**ERROR:** An individual error.

Attributes:

**NUMBER:** The error number.

**SOURCE:** The source of the error.

Elements:

**PARAMETERS:** Element containing one or more parameters (each within a PARAMETER element) defining specific parameters related to the error.

# Chapter 21

---

## Messaging Interface

This chapter describes the methods/functions available to external systems through the Messaging interface (IMessage). These methods allow users to perform a number of messaging functions available in the Messaging module from external systems. These functions include the following:

- Post Message
- Peek Message
- Remove Message
- List Messages
- Hold Message
- Release Message
- Close Message
- Reopen Message
- Reopen Message
- Update Message

See **Appendix C: Messaging** for more information about these functions.

## Methods, Interfaces, and Syntax

The methods, interface objects, and syntax for the Messaging interface are as follows:

### Post Message

**Description:** Used to post a single message to zero or more message queues, based on the message's message type.

**Method Name:** Post

**Interface:** IMessage

**DLL Name:** LSDB.DLL

**Program ID:** LSDB.Message

**Syntax:**

```
HRESULT Post([in] BSTR xmlDataSource,  
             [in] BSTR xmlMessageIn,  
             [out, retval] BSTR* xmlMessageOut);
```

### Peek Message

**Description:** Used to retrieve a single message from a single message queue.

**Method Name:** Peek

**Interface:** IMessage

**DLL Name:** LSDB.DLL

**Program ID:** LSDB.Message

**Syntax:**

```
HRESULT Peek([in] BSTR xmlDataSource,  
             [in] BSTR xmlMessageIn,  
             [out, retval] BSTR* xmlMessageOut);
```

### Remove Message

**Description:** Used to retrieve and remove a single message from a single message queue.

**Method Name:** Remove

**Interface:** IMessage

**DLL Name:** LSDB.DLL

**Program ID:** LSDB.Message

**Syntax:**

```
HRESULT Remove([in] BSTR xmlDataSource,  
              [in] BSTR xmlMessageIn,  
              [out, retval] BSTR* xmlMessageOut);
```

## List Message

**Description:** Used to retrieve a list of messages from a single message queue.

**Method Name:** List

**Interface:** IMessage

**DLL Name:** LSDB.DLL

**Program ID:** LSDB.Message

**Syntax:**

```
HRESULT List([in] BSTR xmlDataSource,  
             [in] BSTR xmlMessageListIn,  
             [out, retval] BSTR* xmlMessageListOut);
```

## Hold Message

**Description:** Used to hold a message in a work queue in order to prevent two or more people from working on the same work queue item at the same time.

**Method Name:** Hold

**Interface:** IMessage

**DLL Name:** LSDB.DLL

**Program ID:** LSDB.Message

**Syntax:**

```
HRESULT Hold([in] BSTR xmlDataSource,  
             [in] BSTR xmlMessageIn,  
             [out, retval] BSTR* xmlMessageOut);
```

## Release Message

**Description:** Used to release a previously held message.

**Method Name:** Rel

**Interface:** IMessage

**DLL Name:** LSDB.DLL

**Program ID:** LSDB.Message

**Syntax:**

```
HRESULT Rel([in] BSTR xmlDataSource,  
            [in] BSTR xmlMessageIn,  
            [out, retval] BSTR* xmlMessageOut);
```

## Close Message

**Description:** Used to close a previously held message.

**Method Name:** Close

**Interface:** IMessage

**DLL Name:** LSDB.DLL

**Program ID:** LSDB.Message

**Syntax:**

```
HRESULT Close([in] BSTR xmlDataSource,  
              [in] BSTR xmlMessageIn,  
              [out, retval] BSTR* xmlMessageOut);
```

## Reopen Message

**Description:** Used to reopen a previously closed message.

**Method Name:** Reopen

**Interface:** IMessage

**DLL Name:** LSDB.DLL

**Program ID:** LSDB.Message

**Syntax:**

```
HRESULT Reopen([in] BSTR xmlDataSource,  
               [in] BSTR xmlMessageIn,  
               [out, retval] BSTR* xmlMessageOut);
```

## Update Message

**Description:** Used to update the note or data fields of a message.

**Method Name:** Update

**Interface:** IMessage

**DLL Name:** LSDB.DLL

**Program ID:** LSDB.Message

**Syntax:**

```
HRESULT Update([in] BSTR xmlDataSource,  
               [in] BSTR xmlMessageIn,  
               [out, retval] BSTR* xmlMessageOut);
```

---

## Interface Arguments

The methods available in the Messaging interface use the following arguments:

### **xmlDataSource Argument**

The `xmlDataSource` argument is an xml string that contains database connection and other related information. A DTD, xml example, and data element descriptions for this argument can be found on page 15-7 in the *Oracle Utilities Energy Information Platform Configuration Guide*.

### **xmlMessageIn Argument**

The `xmlMessageIn` argument is an xml string that contains the parameters for a specific message. A DTD, xml example, and data element descriptions for this argument can be found on page 21-6.

### **xmlMessageOut Argument**

The `xmlMessageOut` argument is an xml string that contains return values from the appropriate Message function. A DTD, xml example, and data element descriptions for this argument can be found on page 21-10.

### **xmlMessageListIn Argument**

The `xmlMessageListIn` argument is an xml string that contains the parameters for a list of messages. A DTD, xml example, and data element descriptions for this argument can be found on page 21-8. This argument is used by the List Messages function only.

### **xmlMessageListOut Argument**

The `xmlMessageListOut` argument is an xml string that contains return values from the List Messages function. A DTD, xml example, and data element descriptions for this argument can be found on page 21-10. This argument is used by the List Messages function only.

## Input Values

The Data Type Definition (DTD), an xml example, and data element descriptions used as input values for the Messaging interface (IMessage) are provided below.

### xmlMessageIn

#### DTD - xmlMessageIn

```
<!DOCTYPE MESSAGE
[
<!ELEMENT MESSAGE (TYPECODE, POSTEDTIME?, SCHEDULEDTIME?, SOURCE?,
POSTEDBYUSERID?, PROCESSEDBYUSERID?, CLOSEDTIME?, OPCOCODE?,
JURISCODE?, UIDACCOUNT?, ACCOUNTID?, AMOUNT?, NOTE?, FILENAME?,
DATA?)>
<!ATTLIST MESSAGE
  QUEUECODECDATA #IMPLIED
  UIDCDATA #IMPLIED>
  STATE (OPEN|HELD|CLOSED) "OPEN">
<!ELEMENT TYPECODE (#PCDATA)>
<!ELEMENT POSTEDTIME (#PCDATA)>
<!ELEMENT SCHEDULEDTIME (#PCDATA)>
<!ELEMENT SOURCE (#PCDATA)>
<!ELEMENT POSTEDBYUSERID (#PCDATA)>
<!ELEMENT PROCESSEDBYUSERID (#PCDATA)>
<!ELEMENT CLOSEDTIME (#PCDATA)>
<!ELEMENT OPCOCODE (#PCDATA)>
<!ELEMENT JURISCODE (#PCDATA)>
<!ELEMENT UIDACCOUNT (#PCDATA)>
<!ELEMENT ACCOUNTID (#PCDATA)>
<!ELEMENT AMOUNT (#PCDATA)>
<!ELEMENT NOTE (#PCDATA)>
<!ELEMENT FILENAME (#PCDATA)>
<!ELEMENT DATA (DATAROOTNODE)>
]>
```

#### XML Examples - xmlMessageIn

##### Post Message

```
<MESSAGE>
  <TYPECODE>WORKQUEUE</TYPECODE>
  <DATA/>
</MESSAGE>
```

##### Peek Message

```
<MESSAGE QUEUECODE="WORKQUEUE" UID="48" />
```

**Note:** The xmlMessageIn arguments used for the Peek, Remove, Hold, Release, Close, and Reopen methods contain the minimal information required to identify the message. They can also include additional information as desired.

##### Remove Message

```
<MESSAGE QUEUECODE="WORKQUEUE" UID="48" />
```

##### Hold Message

```
<MESSAGE QUEUECODE="WORKQUEUE" UID="48" >
  <PROCESSEDBYUSERID>user1</PROCESSEDBYUSERID>
  <CLOSEDTIME>2000-01-01T10:53:54</CLOSEDTIME>
</MESSAGE>
```



**Release Message**

```
<MESSAGE QUEUECODE="WORKQUEUE" UID="48" >
  <PROCESSEDBYUSERID>user1</PROCESSEDBYUSERID>
  <CLOSEDTIME>2000-01-01T10:53:54</CLOSEDTIME>
</MESSAGE>
```

**Close Message**

```
<MESSAGE QUEUECODE="WORKQUEUE" UID="48" >
  <PROCESSEDBYUSERID>user1</PROCESSEDBYUSERID>
  <CLOSEDTIME>2000-01-01T10:53:54</CLOSEDTIME>
</MESSAGE>
```

**Reopen Message**

```
<MESSAGE QUEUECODE="WORKQUEUE" UID="48" >
  <PROCESSEDBYUSERID>user1</PROCESSEDBYUSERID>
  <CLOSEDTIME>2000-01-01T10:53:54</CLOSEDTIME>
</MESSAGE>
```

**Element Descriptions - xmlMessageIn**

Each of the data elements used by the xmlMessageIn argument is described below.

Elements:

**TYPECODE:** Associated message type.

**POSTEDTIME:** Time the message was posted.

**SCHEDULEDTIME:** Time the message was scheduled to be handled (for appropriate message types)

**SOURCE:** Source of the message.

**POSTEDBYUSERID:** User ID of user who posted the message.

**PROCESSEDBYUSERID:** User ID of user who processed the message.

**CLOSEDTIME:** Time the message was closed.

**OPCODE:** Associated operating company.

**JURISCODE:** Associated jurisdiction.

**UIDACCOUNT:** Unique ID of account associated with the message.

**ACCOUNTID:** Account ID of account associated with the message.

**AMOUNT:** Amount of transaction associated with the message.

Attributes:

**CURRENCY:** Currency code for the amount.

**NOTE:** Optional note associated with the message.

**FILENAME:** Indicates the name of an associated file with additional XML data for the message.

**DATA:** Message data.

## xmlMessageListIn

### DTD - xmlMessageListIn

```

<!DOCTYPE MESSAGELIST
[
<!ELEMENT MESSAGELIST (TYPECODE?, STARTTIME?, STOPTIME?, OPCOCODE?,
JURISCODE?, UIDACCOUNT?, ACCOUNTID?, SORTBY?, LIST?)>
<!ATTLIST MESSAGELIST
    QUEUECODECDATA #REQUIRED
    PAGESIZECDATA #IMPLIED
    PAGENOCDATA #IMPLIED >
<!ELEMENT TYPECODE (#PCDATA)>
<!ELEMENT STARTTIME (#PCDATA)>
<!ELEMENT STOPTIME (#PCDATA)>
<!ELEMENT OPCOCODE (#PCDATA)>
<!ELEMENT JURISCODE (#PCDATA)>
<!ELEMENT UIDACCOUNT (#PCDATA)>
<!ELEMENT ACCOUNTID (#PCDATA)>
<!ELEMENT SORTBY EMPTY>
<!ATTLIST SORTBY
    MESSAGETYPECODE (ASC | DESC) #IMPLIED
    SOURCE (ASC | DESC) #IMPLIED
    POSTEDBYUSERID (ASC | DESC) #IMPLIED
    PROCESSEDBYUSERID (ASC | DESC) #IMPLIED
    OPCOCODE (ASC | DESC) #IMPLIED
    JURISCODE (ASC | DESC) #IMPLIED
    ACCOUNTID (ASC | DESC) #IMPLIED
    AMOUNT (ASC | DESC) #IMPLIED>
<!ELEMENT LIST (MESSAGE*)>
]>

```

### XML Example - xmlMessageListIn

```

<MESSAGELIST QUEUECODE="CONTROL">
  <TYPECODE/>
  <OPCOCODE/>
  <JURISCODE/>
  <UIDACCOUNT/>
</MESSAGELIST>

```

### Element Descriptions - xmlMessageListIn

Each of the data elements used by the xmlMessageListIn argument is described below.

MessageList Attributes:

**QUEUECODE:** Associated message queue code for the message list.

**PAGESIZE/PAGENO:** Specify a limited number of messages to be returned starting at the specified page number given the specified page size.

Elements:

**TYPECODE:** Associated message type.

**STARTTIME:** Start time of messages to be included in list.

**STOPTIME:** Stop time of messages to be included in list.

**OPCOCODE:** Associated operating company.

**JURISCODE:** Associated jurisdiction.

**UIDACCOUNT:** Unique ID of account associated with the message.

**ACCOUNTID:** Account ID of account associated with the message.

**SORTBY:** Attributes indicating how messages in the list are to be sorted.

Attributes:

**TYPECODE:** Associated message type

**SOURCE:** Source of message

**POSTEDBYUSERID:** User ID of user who posted the message.

**PROCESSEDBYUSERID:** User ID of user who processed the message.

**OPCOCODE:** Associated operating company.

**JURISCODE:** Associated jurisdiction.

**UIDACCOUNT:** Unique ID of account associated with the message.

**ACCOUNTID:** Account ID of account associated with the message.

**AMOUNT:** Amount of transaction associated with the message.

Attributes:

**CURRENCY:** Currency code for the amount.

## Return Values

The data returned from the Messaging interfaces is described in the following DTD, xml example, and data element descriptions.

### xmlMessageOut/xmlMessageListOut

#### DTD - xmlMessageOut/xmlMessageListOut

The DTDs for the MessageOut and MessageListOut arguments are the same as the MessageIn and MessageListIn arguments.

#### Element Descriptions - xmlMessageOut/xmlMessageListOut

The data elements for the MessageOut argument are the same as the MessageIn argument with the following additional attributes:

Message Attributes:

**QUEUECODE:** Associated message queue code for the message.

**UID:** Unique ID of the message.

The data elements for the MessageListOut arguments are the same as the MessageListIn argument.

# Part Four

---

## Workflow Management Interfaces

Part Four describes the COM interfaces available with the workflow management functionality of Oracle Utilities Billing Component, and contains the following chapters:

- **Chapter 22: Workflow Management Activity Implementations Interface**
- **Chapter 23: Workflow Management Process Versions Interface**
- **Chapter 24: Workflow Management Process Instance Interface**

---

# Chapter 22

---

---

## Workflow Management Activity Implementations Interface

This chapter describes the methods/functions available to external systems through the Activity Implementation interface (IActivityImplementation). These methods allow users to create and maintain Activity Implementations used by Workflow Management from external systems. These functions include the following:

- Insert
- Update
- Delete
- Select
- List

See **Chapter 23: Workflow Activity Implementations** in the *Oracle Utilities Billing Component User's Guide* for more information about these functions.

## Methods, Interfaces, and Syntax

The methods, interface objects, and syntax for the Activity Implementation interface are as follows:

### Insert

**Description:** Used to insert a new activity implementation record into the database.

**Method Name:** Insert

**Interface:** IActivityImplementation

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ActivityImplementation

**Syntax:**

```
HRESULT Insert([in] BSTR xmlDataSource,  
              [in] BSTR xmlActivityImplIn,  
              [out, retval] BSTR* xmlActivityImplOut);
```

### Update

**Description:** Used to update an existing activity implementation record in the database.

**Method Name:** Update

**Interface:** IActivityImplementation

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ActivityImplementation

**Syntax:**

```
HRESULT Update([in] BSTR xmlDataSource,  
              [in] BSTR xmlActivityImpl,
```

### Delete

**Description:** Used to delete an existing activity implementation record in the database.

**Method Name:** Delete

**Interface:** IActivityImplementation

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ActivityImplementation

**Syntax:**

```
HRESULT Delete([in] BSTR xmlDataSource,  
              [in] BSTR xmlActivityImpl,
```



## Select

**Description:** Used to select an existing activity implementation record in the database.

**Method Name:** Select

**Interface:** IActivityImplementation

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ActivityImplementation

**Syntax:**

```
HRESULT List([in] BSTR xmlDataSource,  
             [in] BSTR xmlActivityImplIn,  
             [out, retval] BSTR* xmlActivityImplOut);
```

## List

**Description:** Used to list existing activity implementation records in the database.

**Method Name:** List

**Interface:** IActivityImplementation

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ActivityImplementation

**Syntax:**

```
HRESULT Select([in] BSTR xmlDataSource,  
              [in] BSTR xmlActivityImplListIn,  
              [out, retval] BSTR* xmlActivityImplListOut);
```

## Interface Arguments

The methods available in the Activity Implementation interface use the following arguments:

### **xmlDataSource Argument**

The xmlDataSource argument is an xml string that contains database connection and other related information. A DTD, xml example, and data element descriptions for this argument can be found on page 15-7 in the *Oracle Utilities Energy Information Platform Configuration Guide*.

### **xmlActivityImplIn Argument**

The xmlActivityImplIn argument is an xml string that contains the parameters for a specific activity implementation. When used with the Update and Delete functions, this argument is called xmlActivityImpl. An xml example and data element descriptions for this argument can be found on page 22-5.

### **xmlActivityImplOut Argument**

The xmlActivityImplOut argument is an xml string that contains return values from the appropriate Activity Implementation function. An xml example and data element descriptions for this argument can be found on page 22-7.

### **xmlActivityImplListIn Argument**

The xmlActivityImplListIn argument is an xml string that contains the parameters for a list of activity implementations. An xml example and data element descriptions for this argument can be found on page 12-5. This argument is used by the List function only.

### **xmlActivityImplListOut Argument**

The xmlActivityImplListOut argument is an xml string that contains return values from the List function. An xml example and data element descriptions for this argument can be found on page 12-7. This argument is used by the List function only.

## Input Values

Xml examples and data element descriptions used as input values for the Activity Implementation interface (IAActivityImplementation) are provided below.

### xmlActivityImplementationIn

#### XML Example - xmlActivityImplIn

```
<ACTIVITYIMPL>
  <UID>1</UID>
  <NAME>Activity Implementation Name</NAME>
  <DESCRIPTION>Activity Implementation Description</DESCRIPTION>
  <TYPE>MESSAGE</TYPE>
  <MESSAGETYPE>ACTVIMPL_MSG_CODE</MESSAGETYPE>
  <CONTEXTSCHEMA>
    <Schema name=' Schema Name'
      xmlns='urn:schemas-microsoft-com:xml-data'
    </Schema>
  </CONTEXTSCHEMA>
</ACTIVITYIMPL>
```

#### Element Descriptions - xmlActivityImplIn

Each of the data elements used by the xmlActivityImplIn argument is described below.

**ACTIVITYIMPL:** The root document element of the activity implementation XML structure. Represents a single activity implementation record.

Elements:

**UID:** UID of the activity implementation record.

**NAME:** Name of the activity implementation.

**DESCRIPTION:** Optional description for the activity implementation.

**TYPE:** Type of activity implementation. Valid values are "MESSAGE", "EVENT", "PROCESS", and "RATEFORM".

**MESSAGETYPE:** Associated message type code if the activity implementation type is "MESSAGE".

**EVENTTYPE:** Associated event type code if the activity implementation type is "EVENT".

**PROCESS:** Associated process UID if the activity implementation type is "PROCESS".

**RATEFORM:** Associated rate form UID if the activity implementation type is "RATEFORM".

**CONTEXTSCHEMA:** Optional context schema for the activity implementation.

### xmlMessageListIn

#### XML Example - xmlActivityImplListIn

```
<ACTIVITYIMPLLIST>
  <ACTIVITYIMPL>
    <UID>1</UID>
    <NAME>Activity Implementation Name</NAME>
    <DESCRIPTION>Activity Implementation Description</DESCRIPTION>
    <TYPE>MESSAGE</TYPE>
    <MESSAGETYPE>ACTVIMPL_MSG_CODE</MESSAGETYPE>
    <CONTEXTSCHEMA>
      <Schema name=' Schema Name'
        xmlns='urn:schemas-microsoft-com:xml-data'
      </Schema>
    </CONTEXTSCHEMA>
  </ACTIVITYIMPL>
</ACTIVITYIMPLLIST>
```

```
</ACTIVITYIMPL>  
<ACTIVITYIMPL></ACTIVITYIMPL>  
</ACTIVITYIMPLLIST>
```

### Element Descriptions - xmlActivityImplListIn

Each of the data elements used by the xmlActivityImplListIn argument is described below.

**ACTIVITYIMPLLIST:** Root document element of activity implementation list XML structure. Represents a list of zero or more activity implementation records.

**ACTIVITYIMPL:** Root element of activity implementation record (see above). Zero or more of these elements may exist.

## Return Values

This section describes the data returned from the Activity Implementation interface.

### **xmlActivityImplOut/xmlActivityImplListOut**

#### **XML Schema and Element Descriptions**

##### **- xmlActivityImplOut/xmlActivityImplListOut**

The data elements for the ActivityImplOut and ActivityImplListOut arguments are the same as the ActivityImplIn and ActivityImplListIn arguments.



# Chapter 23

---

## Workflow Management Process Versions Interface

This chapter describes the methods/functions available to external systems through the Process Version interface (IProcessVersion). These methods allow users to create and maintain Processes, Process Versions, and Process Activities used by Workflow Management from external systems. These functions include the following:

- Insert Process Version
- Update Process Version
- Delete Process Version
- Select Process Version
- Copy Process Version
- Validate Process Version
- List Process Versions
- Insert Process
- Update Process
- Delete Process
- Select Process
- List Processes
- Insert Activity
- Update Activity
- Delete Activity
- Select Activity
- List Activities

See **Chapter 24: Workflow Processes and Process Versions** in the *Oracle Utilities Billing Component User's Guide* for more information about these functions.

## Methods, Interfaces, and Syntax

The methods, interface objects, and syntax for the Process Version interface are as follows:

### Insert

**Description:** Used to insert a new process version record into the database.

**Method Name:** Insert

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT Insert([in] BSTR xmlDataSource,  
              [in] BSTR xmlProcessVersionIn,  
              [out, retval] BSTR* xmlProcessVersionOut);
```

### Update

**Description:** Used to update an existing process version record in the database.

**Method Name:** Update

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT Update([in] BSTR xmlDataSource,  
              [in] BSTR xmlProcessVersion,
```

### Delete

**Description:** Used to delete an existing process version record in the database.

**Method Name:** Delete

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT Delete([in] BSTR xmlDataSource,  
              [in] BSTR xmlProcessVersion,
```

### Select

**Description:** Used to select an existing process version record in the database.

**Method Name:** Select

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT Select([in] BSTR xmlDataSource,  
              [in] BSTR xmlProcessVersionIn,  
              [out, retval] BSTR* xmlProcessVersionOut);
```



## Copy

**Description:** Used to copy an existing process version record from the database, along with all its child process activity records, and inserts the copies into the database.

**Method Name:** Copy

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT Copy([in] BSTR xmlDataSource,
             [in] BSTR xmlProcessVersionIn,
             [out, retval] BSTR* xmlProcessVersionOut);
```

## Validate

**Description:** Used to validate an existing process version record in the database.

**Method Name:** Select

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT Validate([in] BSTR xmlDataSource,
                 [in] BSTR xmlProcessVersion,
```

## List

**Description:** Used to list existing process version records in the database.

**Method Name:** List

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT List([in] BSTR xmlDataSource,
             [in] BSTR xmlProcessVersionListIn,
             [out, retval] BSTR* xmlProcessVersionListOut);
```

## Insert Process

**Description:** Used to insert a new process record into the database.

**Method Name:** InsertProcess

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT InsertProcess([in] BSTR xmlDataSource,
                     [in] BSTR xmlProcessIn,
                     [out, retval] BSTR* xmlProcessOut);
```

## Update Process

**Description:** Used to update an existing process record in the database.

**Method Name:** UpdateProcess

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT UpdateProcess([in] BSTR xmlDataSource,  
                      [in] BSTR xmlProcess,
```

## Delete Process

**Description:** Used to delete an existing process record in the database.

**Method Name:** DeleteProcess

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT DeleteProcess([in] BSTR xmlDataSource,  
                     [in] BSTR xmlProcess,
```

## Select Process

**Description:** Used to select an existing process record in the database.

**Method Name:** SelectProcess

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT SelectProcess([in] BSTR xmlDataSource,  
                     [in] BSTR xmlProcessIn,  
                     [out, retval] BSTR* xmlProcessOut);
```

## List Processes

**Description:** Used to list existing process records in the database.

**Method Name:** ListProcesses

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT ListProcesses([in] BSTR xmlDataSource,  
                     [in] BSTR xmlProcessListIn,  
                     [out, retval] BSTR* xmlProcessListOut);
```

## Insert Activity

**Description:** Used to insert a new process activity record into the database.

**Method Name:** InsertActivity

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT InsertActivity([in] BSTR xmlDataSource,  
                       [in] BSTR xmlProcessActivityIn,  
                       [out, retval] BSTR* xmlProcessActivityOut);
```

## Update Activity

**Description:** Used to update an existing process activity record in the database.

**Method Name:** UpdateActivity

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT UpdateActivity([in] BSTR xmlDataSource,  
                      [in] BSTR xmlProcessActivity,
```

## Delete Activity

**Description:** Used to delete an existing process activity record in the database.

**Method Name:** DeleteActivity

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT DeleteActivity([in] BSTR xmlDataSource,  
                      [in] BSTR xmlProcessActivity,
```

## Select Activity

**Description:** Used to select an existing process activity record in the database.

**Method Name:** SelectActivity

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT SelectActivity([in] BSTR xmlDataSource,  
                      [in] BSTR xmlProcessActivityIn,  
                      [out, retval] BSTR* xmlProcessActivityOut);
```

## List Activities

**Description:** Used to list existing process activity records in the database.

**Method Name:** ListActivities

**Interface:** IProcessVersion

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessVersion

**Syntax:**

```
HRESULT ListActivities([in] BSTR xmlDataSource,  
                      [in] BSTR xmlProcessActivityListIn,  
                      [out, retval] BSTR* xmlProcessActivityListOut);
```

---

## Interface Arguments

The methods available in the Process Version interface use the following arguments:

### **xmlDataSource Argument**

The `xmlDataSource` argument is an xml string that contains database connection and other related information. A DTD, xml example, and data element descriptions for this argument can be found on page 15-7 in the *Oracle Utilities Energy Information Platform Configuration Guide*.

### **xmlProcessVersionIn Argument**

The `xmlProcessVersionIn` argument is an xml string that contains the parameters for a specific process version. When used with the Update and Delete functions, this argument is called `xmlProcessVersion`. An xml example and data element descriptions for this argument can be found on page 23-9.

### **xmlProcessVersionOut Argument**

The `xmlProcessVersionOut` argument is an xml string that contains return values from the appropriate Process Version function. An xml example and data element descriptions for this argument can be found on page 23-14.

### **xmlProcessVersionListIn Argument**

The `xmlProcessVersionListIn` argument is an xml string that contains the parameters for a list of process versions. An xml example and data element descriptions for this argument can be found on page 23-10. This argument is used by the List function only.

### **xmlProcessVersionListOut Argument**

The `xmlProcessVersionListOut` argument is an xml string that contains return values from the List function. An xml example and data element descriptions for this argument can be found on page 23-14. This argument is used by the List function only.

### **xmlProcessIn Argument**

The `xmlProcessIn` argument is an xml string that contains the parameters for a specific process. When used with the Update Process and Delete Process functions, this argument is called `xmlProcess`. An xml example and data element descriptions for this argument can be found on page 23-10.

### **xmlProcessOut Argument**

The `xmlProcessOut` argument is an xml string that contains return values from the appropriate Process function. An xml example and data element descriptions for this argument can be found on page 23-14.

### **xmlProcessListIn Argument**

The `xmlProcessVersionListIn` argument is an xml string that contains the parameters for a list of processes. An xml example and data element descriptions for this argument can be found on page 23-11. This argument is used by the List function only.

### **xmlProcessListOut Argument**

The `xmlProcessVersionListOut` argument is an xml string that contains return values from the List function. An xml example and data element descriptions for this argument can be found on page 23-14. This argument is used by the List function only.

### **xmlProcessActivityIn Argument**

The `xmlProcessActivityIn` argument is an xml string that contains the parameters for a specific process activity. When used with the Update Activity and Delete Activity functions, this argument is called `xmlProcessActivity`. An xml example and data element descriptions for this argument can be found on page 23-11.

### **xmlProcessActivityOut Argument**

The `xmlProcessActivityOut` argument is an xml string that contains return values from the appropriate Process Activity function. An xml example and data element descriptions for this argument can be found on page 23-14.

### **xmlProcessActivityListIn Argument**

The `xmlProcessActivityListIn` argument is an xml string that contains the parameters for a list of process activities. An xml example and data element descriptions for this argument can be found on page 23-13. This argument is used by the List function only.

### **xmlProcessActivityListOut Argument**

The `xmlProcessActivityListOut` argument is an xml string that contains return values from the List function. An xml example and data element descriptions for this argument can be found on page 23-14. This argument is used by the List function only.

## Input Values

Xml examples and data element descriptions used as input values for the Process Version interface (IProcessVersion) are provided below.

### xmlProcessVersionIn

#### XML Example - xmlProcessVersionIn

```
<PROCESSVERSION>
  <UID>1</UID>
  <PROCESS>1</PROCESS>
  <MAJORVERSION>2</MAJORVERSION>
  <MINORVERSION>11</MINORVERSION>
  <NOTE>Process Version Note</NOTE>
  <VALIDATED>FALSE</VALIDATED>
  <CONTEXTSCHEMA>
    <Schema name='Schema Name'
      xmlns='urn:schemas-microsoft-com:xml-data'>
    </Schema>
  </CONTEXTSCHEMA>
</PROCESSVERSION>
```

#### Element Descriptions - xmlProcessVersionIn

Each of the data elements used by the xmlProcessVersionIn argument is described below.

**PROCESSVERSION:** The root document element of the process version XML structure. Represents a single process version record.

Elements:

**UID:** UID of the process version record.

**PROCESS:** UID of the parent process record.

**MAJORVERSION:** Major version number of process version.

**MINORVERSION:** Minor version number of process version.

**NOTE:** Optional note for process version.

**VALIDATED:** Validated flag for process version. Automatically set to FALSE when process or any child process activities are edited. Only set to TRUE by the Validate function, if successful.

**CONTEXTSCHEMA:** Optional context schema for the activity implementation.

## xmlProcessVersionListIn

### XML Example - xmlProcessVersionListIn

```
<PROCESSVERSIONLIST>
  <PROCESSVERSION>
    <UID>1</UID>
    <PROCESS>1</PROCESS>
    <MAJORVERSION>2</MAJORVERSION>
    <MINORVERSION>11</MINORVERSION>
    <NOTE>Process Version Note</NOTE>
    <VALIDATED>FALSE</VALIDATED>
    <CONTEXTSCHEMA>
      <Schema name='Schema Name'
        xmlns='urn:schemas-microsoft-com:xml-data'>
      </Schema>
    </CONTEXTSCHEMA>
  </PROCESSVERSION>
</PROCESSVERSIONLIST>
```

### Element Descriptions - xmlProcessVersionListIn

Each of the data elements used by the xmlProcessVersionListIn argument is described below.

**PROCESSVERSIONLIST:** Root document element of process version list XML structure. Represents a list of zero or more process version records.

**PROCESSVERSION:** Root element of process version record (see above). Zero or more of these elements may exist.

## xmlProcessIn

### XML Example - xmlProcessIn

```
<PROCESS>
  <UID>1</UID>
  <NAME>Process Name</NAME>
  <OPCO>OPCO_CODE</OPCO>
  <JURIS>JURIS_CODE</JURIS>
  <DESCRIPTION>Process Description</DESCRIPTION>
</PROCESS>
```

### Element Descriptions - xmlProcessIn

Each of the data elements used by the xmlProcessIn argument is described below.

**PROCESS:** The root document element of the process XML structure. Represents a singleprocess record.

Elements:

**UID:** UID of the process record.

**NAME:** Name of the process.

**OPCO:** Optional operating company code for the process.

**JURIS:** Optional jurisdiction code for the process.

**DESCRIPTION:** Optional description for the process.



## xmlProcessListIn

### XML Example - xmlProcessListIn

```
<PROCESSLIST>
  <PROCESS>
    <UID>1</UID>
    <NAME>Process Name</NAME>
    <OPCO>OPCO_CODE</OPCO>
    <JURIS>JURIS_CODE</JURIS>
    <DESCRIPTION>Process Description</DESCRIPTION>
  </PROCESS>
</PROCESSLIST>
```

### Element Descriptions - xmlProcessListIn

Each of the data elements used by the xmlProcessListIn argument is described below.

**PROCESSLIST:** Root document element of process list XML structure. Represents a list of zero or more process records.

**PROCESS:** Root element of process record (see above). Zero or more of these elements may exist.

## xmlProcessActivityIn

### XML Example - xmlProcessActivityIn

```
<PROCESSACTIVITY>
  <UID>1</UID>
  <PROCESSVERSION>1</PROCESSVERSION>
  <ID>B</ID>
  <ACTIVITYIMPL>1</ACTIVITYIMPL>
  <PROPERTIES STARTCOND='ALL'
    EXITCOND='TRUE'
    WAITFOR='TRUE'>
    <EXPTIME RELATIVE='TRUE'
      USEBUSINESSDAYS='TRUE'
      ONEXPIRE='EXIT'>0000-00-05 00:00:00</EXPTIME>
  </PROPERTIES>
  <INPUTMAP>
    <ITEM FROM='/ACCOUNTID' TO='/ACCOUNTID' />
    <ITEM FROM='10' TO='/COUNT' />
  </INPUTMAP>
  <OUTPUTMAP>
    <ITEM FROM='Literal' TO='/ACTIVITY_B/RESULT' />
  </OUTPUTMAP>
  <PATH SOURCEID='A' TRANSCOND='TRUE' />
</PROCESSACTIVITY>
```

### Element Descriptions - xmlProcessActivityIn

Each of the data elements used by the xmlProcessActivityIn argument is described below.

**PROCESSACTIVITY:** Root document element of process activity XML structure. Represents a single process activity record.

Elements:

**UID:** UID of the process activity record.

**PROCESSVERSION:** UID of the parent process version record.

**ID:** Unique id of the process activity within process version.

**ACTIVITYIMPL:** UID of the associated activity implementation.

**PROPERTIES:** Root element for additional process activity properties.

**@STARTCOND:** Start condition for the process activity. Valid values are:

"ALL": all input path transition conditions must evaluate to true (default)

"ANY": one or more input path transition conditions must evaluate to true

**@EXITCOND:** Exit condition for the process activity. Should be a valid XPath expression against the activity implementation context. May also be the literal value "TRUE" (default).

**@WAITFOR:** Wait For flag for the process activity. Valid values are:

"TRUE": wait for the activity to finish before moving on (default)

"FALSE": do not wait for activity to finish

**EXPTIME:** Optional expiration time for the activity. Only valid if Wait For Flag is set to TRUE. Time is expressed as YYYY-MM-DD HH:MM:SS, and may be absolute or relative.

**@RELATIVE:** Indicates whether expiration time is relative ("TRUE") or absolute ("FALSE") (default). Relative times are relative to the time that the activity is started.

**@USEBUSINESSDAYS:** Indicates whether the days value of a relative expiration time should include only business days ("TRUE") or all days ("FALSE") (default).

**@ONEXPIRE:** Indicates behavior of process when activity expires. Valid values are:

"ERROR": place activity in ERROR state (default)

"EXIT": place activity in COMPLETED state

**INPUTMAP:** Root element for optional input context map for the activity.

**ITEM:** Root element for an individual context map item.

**@FROM:** From expression. Should be valid XPath expression against the process version context or a literal value enclosed in quotes.

**@TO:** To expression. Should be valid XPath expression against the activity implementation context.

**OUTPUTMAP:** Root element for optional output context map for the activity.

**ITEM:** Root element for an individual context map item.

**@FROM:** From expression. Should be valid XPath expression against the activity implementation context or a literal value enclosed in quotes.

**@TO:** To expression. Should be valid XPath expression against the process version context.

**PATH:** Root element of optional input path for the activity. Zero or more may exist.

**@SOURCEID:** Id of the source process activity within the parent process version.

**@TRANSCOND:** Transition condition for the path. Should be a valid Boolean XPath expression against the process version context. May also be the literal value "TRUE" (default).

## xmlProcessActivityListIn

### XML Example - xmlProcessActivityListIn

```

<PROCESSACTIVITYLIST>
  <PROCESSACTIVITY>
    <UID>1</UID>
    <PROCESSVERSION>1</PROCESSVERSION>
    <ID>B</ID>
    <ACTIVITYIMPL>1</ACTIVITYIMPL>
    <PROPERTIES STARTCOND='ALL'
      EXITCOND='TRUE'
      WAITFOR='TRUE'>
    <EXPTIME RELATIVE='TRUE'
      USEBUSINESSDAYS='TRUE'
      ONEXPIRE='EXIT'>0000-00-05 00:00:00</EXPTIME>
    <INPUTMAP>
      <ITEM FROM='/ACCOUNTID' TO='/ACCOUNTID' />
      <ITEM FROM='10' TO='/COUNT' />
    </INPUTMAP>
    <OUTPUTMAP>
      <ITEM FROM='"Literal"' TO='/ACTIVITY_B/RESULT' />
    </OUTPUTMAP>
    <PATH SOURCEID='A' TRANSCOND='"TRUE"' />
  </PROPERTIES>
</PROCESSACTIVITY>
<PROCESSACTIVITY></PROCESSACTIVITY>
</PROCESSACTIVITYLIST>

```

### Element Descriptions - xmlProcessActivityListIn

Each of the data elements used by the xmlProcessActivityListIn argument is described below.

**PROCESSACTIVITYLIST:** Root document element of process activity list XML structure. Represents a list of zero or more process activity records.

**PROCESSACTIVITY:** Root element of process activity record (see above). Zero or more of these elements may exist.

## Return Values

This section describes the data returned from the Process Version interface.

### **xmlProcessVersionOut/xmlProcessVersionListOut**

#### **XML Schema and Element Descriptions**

##### **- xmlProcessVersionOut/xmlProcessVersionListOut**

The data elements for the ProcessVersionOut and ProcessVersionListOut arguments are the same as the ProcessVersionIn and ProcessVersionListIn arguments.

### **xmlProcessOut/xmlProcessListOut**

#### **XML Schema and Element Descriptions**

##### **- xmlProcessOut/xmlProcessListOut**

The Xdata elements for the ProcessOut and ProcessListOut arguments are the same as the ProcessIn and ProcessListIn arguments.

### **xmlProcessActivityOut/xmlProcessActivityListOut**

#### **XML Schema and Element Descriptions**

##### **- xmlProcessActivityOut/xmlProcessActivityListOut**

The data elements for the ProcessActivityOut and ProcessActivityListOut arguments are the same as the ProcessActivityIn and ProcessActivityListIn arguments.

# Chapter 24

---

## Workflow Management Process Instance Interface

This chapter describes the methods/functions available to external systems through the Process Instance interface (IProcessInstance). These methods allow users to work with process and activity instances used by Workflow Management from external systems. These functions include the following:

- List Process Instances
- Select Process Instance
- Start Process Instance
- Suspend Process Instance
- Resume Process Instance
- Repair Process Instance
- Terminate Process Instance
- List Activity Instances
- Select Activity Instance
- Repair Activity Instance
- Post Activity Event
- Activity Finished
- Activity Expired
- Activity InError

See **Chapter 25: Workflow Process Activities** in the *Oracle Utilities Billing Component User's Guide* for more information about these functions.

## Methods, Interfaces, and Syntax

The methods, interface objects, and syntax for the Process Instance interface are as follows:

### List

**Description:** Used to return a list of zero or more process instances from the Workflow System.

**Method Name:** List

**Interface:** IProcessInstance

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessInstance

**Syntax:**

```
HRESULT List([in] BSTR xmlDataSource,  
             [in] BSTR xmlProcessInstanceListIn,  
             [out, retval] BSTR* xmlProcessInstanceListOut);
```

### Select

**Description:** Used to select a specified process instance from the Workflow System.

**Method Name:** Select

**Interface:** IProcessInstance

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessInstance

**Syntax:**

```
HRESULT Select([in] BSTR xmlDataSource,  
              [in] BSTR xmlProcessInstanceIn,  
              [out, retval] BSTR* xmlProcessInstanceOut);
```

### Start

**Description:** Used to create a new process instance record in the Workflow System.

**Method Name:** Start

**Interface:** IProcessInstance

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessInstance

**Syntax:**

```
HRESULT Start([in] BSTR xmlDataSource,  
             [in] BSTR xmlProcessInstanceIn,  
             [out, retval] BSTR* xmlProcessInstanceOut);
```

### Suspend

**Description:** Used to suspend an existing running process instance in the Workflow System.

**Method Name:** Suspend

**Interface:** IProcessInstance

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessInstance

**Syntax:**

```
HRESULT Suspend([in] BSTR xmlDataSource,
```

```
[in] BSTR xmlProcessInstance,
```

## Resume

**Description:** Used to resume an existing suspended or in error process instance in the Workflow System.

**Method Name:** Resume

**Interface:** IProcessInstance

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessInstance

**Syntax:**

```
HRESULT Resume([in] BSTR xmlDataSource,
               [in] BSTR xmlProcessInstance,
```

## Repair

**Description:** Used to update the context of an existing suspended or in error process instance in the Workflow System.

**Method Name:** Repair

**Interface:** IProcessInstance

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessInstance

**Syntax:**

```
HRESULT Repair([in] BSTR xmlDataSource,
               [in] BSTR xmlProcessInstance,
```

## Terminate

**Description:** Used to terminate an existing running, suspended, or in error process instance in the Workflow System.

**Method Name:** Terminate

**Interface:** IProcessInstance

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessInstance

**Syntax:**

```
HRESULT Terminate([in] BSTR xmlDataSource,
                  [in] BSTR xmlProcessInstance,
```

## List Activities

**Description:** Used to return a list of zero or more activity instances in the Workflow System.

**Method Name:** ListActivities

**Interface:** IProcessInstance

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessInstance

**Syntax:**

```
HRESULT ListActivities([in] BSTR xmlDataSource,
                      [in] BSTR xmlActivityInstanceListIn,
                      [out, retval] BSTR* xmlActivityInstanceListOut);
```

## Select Activity

**Description:** Used to select an existing activity instance from the Workflow System.

**Method Name:** SelectActivity

**Interface:** IProcessInstance

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessInstance

**Syntax:**

```
HRESULT SelectActivity([in] BSTR xmlDataSource,  
                      [in] BSTR xmlActivityInstanceIn,  
                      [out, retval] BSTR* xmlActivityInstanceOut);
```

## Repair Activity

**Description:** Used to update the state (to INACTIVE) and the context of an existing in error activity instance in the Workflow System.

**Method Name:** RepairActivity

**Interface:** IProcessInstance

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessInstance

**Syntax:**

```
HRESULT RepairActivity([in] BSTR xmlDataSource,  
                      [in] BSTR xmlActivityInstance,
```

## Post Activity Event

**Description:** Used to post an event related to zero or more existing activity instances in the Workflow System. Only the EVENTTYPE, ACCOUNT, and EVENTDATA elements of the Activity Event xml argument (p. 14-9) are used.

**Method Name:** PostActivityEvent

**Interface:** IProcessInstance

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessInstance

**Syntax:**

```
HRESULT PostActivityEvent([in] BSTR xmlDataSource,  
                          [in] BSTR xmlActivityEvent,
```

## Activity Finished

**Description:** Used to notify the Workflow System that a running activity has finished.

**Method Name:** ActivityFinished

**Interface:** IProcessInstance

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessInstance

**Syntax:**

```
HRESULT ActivityFinished([in] BSTR xmlDataSource,  
                        [in] BSTR xmlActivityInstance,
```



## Activity Expired

**Description:** Used to notify the Workflow System that a running activity has expired.

**Method Name:** ActivityExpired

**Interface:** IProcessInstance

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessInstance

**Syntax:**

```
HRESULT ActivityExpired([in] BSTR xmlDataSource,
                        [in] BSTR xmlActivityInstance,
```

## Activity InError

**Description:** Used to notify the Workflow System that a running activity is in error.

**Method Name:** ActivityInError

**Interface:** IProcessInstance

**DLL Name:** LSWRKFLW.DLL

**Program ID:** LSWRKFLW.ProcessInstance

**Syntax:**

```
HRESULT ActivityInError([in] BSTR xmlDataSource,
                        [in] BSTR xmlActivityInstance,
```

## Interface Arguments

The methods available in the Process Instance interface use the following arguments:

### xmlDataSource Argument

The xmlDataSource argument is an xml string that contains database connection and other related information. A DTD, xml example, and data element descriptions for this argument can be found on page 15-7 in the *Oracle Utilities Energy Information Platform Configuration Guide*.

### xmlProcessInstanceIn Argument

The xmlProcessInstanceIn argument is an xml string that contains the parameters for a specific process instance. When used with the Suspend, Resume, Repair and Terminate functions, this argument is called xmlProcessInstance. An xml example and data element descriptions for this argument can be found on page 24-7.

### xmlProcessInstanceOut Argument

The xmlProcessInstanceOut argument is an xml string that contains return values from the appropriate Process Instance function. An xml example and data element descriptions for this argument can be found on page 24-10.

### xmlProcessInstanceListIn Argument

The xmlProcessInstanceListIn argument is an xml string that contains the parameters for a list of process instances. An xml example and data element descriptions for this argument can be found on page 24-8. This argument is used by the List function only.

### xmlProcessInstanceListOut Argument

The xmlProcessInstanceListOut argument is an xml string that contains return values from the List function. An xml example and data element descriptions for this argument can be found on page 24-10. This argument is used by the List function only.

### **xmlActivityInstanceIn Argument**

The `xmlActivityInstanceIn` argument is an xml string that contains the parameters for a specific activity instance. When used with the `Activity Finished`, `Activity Expired`, and `Activity InError` functions, this argument is called `xmlActivityInstance`. An xml example and data element descriptions for this argument can be found on page 24-8.

### **xmlActivityInstanceOut Argument**

The `xmlActivityInstanceOut` argument is an xml string that contains return values from the appropriate Activity Instance function. An xml example and data element descriptions for this argument can be found on page 24-10.

### **xmlActivityInstanceListIn Argument**

The `xmlActivityInstanceListIn` argument is an xml string that contains the parameters for a list of process instances. An xml example and data element descriptions for this argument can be found on page 24-9. This argument is used by the `List` function only.

### **xmlActivityInstanceListOut Argument**

The `xmlActivityInstanceListOut` argument is an xml string that contains return values from the `List` function. An xml example and data element descriptions for this argument can be found on page 24-10. This argument is used by the `List` function only.

### **xmlActivityEvent Argument**

The `xmlActivityEventIn` argument is an xml string that contains the parameters for a specific activity event. An xml example and data element descriptions for this argument can be found on page 24-9.

## Input Values

XML examples and data element descriptions used as input values for the Process Instance interface (IProcessInstance) are provided below.

### xmlProcessInstanceIn

#### XML Example - xmlProcessInstanceIn

```
<PROCESSINSTANCE>
  <UID>1</UID>
  <PROCESSVERSION>
  </PROCESSVERSION>
  <STATE>RUNNING</STATE>
  <STARTTIME>2001-06-15T12:30:00</STARTTIME>
  <STOPTIME></STOPTIME>
  <ACCOUNT>1234</ACCOUNT>
  <ACTIVITYINSTANCE></ACTIVITYINSTANCE>
  <NOTE></NOTE>
  <CONTEXT></CONTEXT>
</PROCESSINSTANCE>
```

#### Element Descriptions - xmlProcessInstanceIn

Each of the data elements used by the xmlProcessInstanceIn argument is described below.

**PROCESSINSTANCE:** Root document element of process instance XML structure. Represents a single process instance record.

Elements

**UID:** UID of process instance.

**PROCESSVERSION:** Associated process version element (see description above).

**STATE:** State of the process. Valid values include "RUNNING", "SUSPENDED", "INERROR", "TERMINATED", and "COMPLETED".

**STARTTIME:** Time that process was started.

**STOPTIME:** Time that process was suspended, in error, terminated, or completed.

**ACCOUNT:** Optional uid of related account or

Elements

**ID:** Optional id of related account.

**UID:** Optional uid of related account.

**ACTIVITYINSTANCE:** Optional uid of activity instance for which the process instance is an implementation (sub-process).

**NOTE:** Optional note for the process.

**CONTEXT:** Optional context for the process. This element's schema is defined by the context schema of the associated process version.

## xmlProcessInstanceListIn

### XML Example - xmlProcessVersionListIn

```
<PROCESSINSTANCELIST>
  <PROCESSINSTANCE>
    <UID>1</UID>
    <PROCESSVERSION>
    </PROCESSVERSION>
    <STATE>RUNNING</STATE>
    <STARTTIME>2001-06-15T12:30:00</STARTTIME>
    <STOPTIME></STOPTIME>
    <ACCOUNT>1234</ACCOUNT>
    <ACTIVITYINSTANCE></ACTIVITYINSTANCE>
    <NOTE></NOTE>
    <CONTEXT></CONTEXT>
  </PROCESSINSTANCE>
</PROCESSINSTANCELIST>
```

### Element Descriptions - xmlProcessInstanceListIn

Each of the data elements used by the xmlProcessInstanceListIn argument is described below.

**PROCESSINSTANCELIST:** Root document element of process instance list XML structure. Represents a list of zero or more process instance records.

**PROCESSINSTANCE:** Root element of process instance record (see above). Zero or more of these elements may exist.

## xmlActivityInstanceIn

### XML Examples - xmlActivityInstanceIn

```
<ACTIVITYINSTANCE>
  <UID>1</UID>
  <PROCESSINSTANCE>1</PROCESSINSTANCE>
  <PROCESSACTIVITY>
  </PROCESSACTIVITY>
  <STATE>RUNNING</STATE>
  <STARTTIME>2001-06-15T12:30:00</STARTTIME>
  <STOPTIME></STOPTIME>
  <CONTEXT></CONTEXT>
</ACTIVITYINSTANCE>
```

### Element Descriptions - xmlActivityInstanceIn

Each of the data elements used by the xmlActivityInstanceIn argument is described below.

**ACTIVITYINSTANCE:** Root document element of activity instance XML structure. Represents a single activity instance record.

Elements:

**UID:** UID of activity instance.

**PROCESSINSTANCE:** Uid of parent process instance record.

**PROCESSACTIVITY:** Associated process activity element (see description above).

**STATE:** State of the activity. Valid values include "INACTIVE", "RUNNING", "SKIPPED", "FINISHED", "EXPIRED", "INERROR", "TERMINATED", and "COMPLETED".

**STARTTIME:** Time that activity was started.

**STOPTIME:** Time that activity was finished, in error, or terminated.

**CONTEXT:** Optional context for the activity. This element's schema is defined by the context schema of the associated activity implementation.

## xmlActivityInstanceListIn

### XML Example - xmlActivityInstanceListIn

```
<ACTIVITYINSTANCELIST>
  <ACTIVITYINSTANCE>
    <UID>1</UID>
    <PROCESSINSTANCE>1</PROCESSINSTANCE>
    <PROCESSACTIVITY>
    </PROCESSACTIVITY>
    <STATE>RUNNING</STATE>
    <STARTTIME>2001-06-15T12:30:00</STARTTIME>
    <STOPTIME></STOPTIME>
    <CONTEXT></CONTEXT>
  </ACTIVITYINSTANCE>
</ACTIVITYINSTANCELIST>
```

### Element Descriptions - xmlActivityInstanceListIn

Each of the data elements used by the xmlActivityInstanceListIn argument is described below.

**ACTIVITYINSTANCELIST:** Root document element of activity instance list XML structure. Represents a list of zero or more activity instance records.

**ACTIVITYINSTANCE:** Root element of activity instance record (see above). Zero or more of these elements may exist.

## xmlActivityEvent

### XML Examples - xmlActivityEvent

```
<ACTIVITYEVENT>
  <ACTIVITYINSTANCE>1</ACTIVITYINSTANCE>
  <EVENTTYPE>SPEEDPAY</EVENTTYPE>
  <ACCOUNT>1234</ACCOUNT>
  <WAITING>FALSE</WAITING>
  <EVENTTIME></EVENTTIME>
  <EVENTDATA></EVENTDATA>
</ACTIVITYEVENT>
```

### Element Descriptions - xmlActivityEvent

Each of the data elements used by the xmlActivityEvent argument is described below.

**ACTIVITYEVENT:** Root document element of activity event XML structure. Represents a single activity event record.

Elements:

**ACTIVITYINSTANCE:** UID of associated activity instance.

**EVENTTYPE:** Code of associated event type.

**ACCOUNT:** Optional uid of associated account.

**WAITING:** Waiting flag of event. If true then associated activity has started.

**EVENTTIME:** Time that event was posted.

**EVENTDATA:** Optional data associated with the event. This element's schema is defined by the context schema of the associated activity implementation.

## Return Values

This section describes the data returned from the Process Instance interface.

### **xmlProcessInstanceOut/xmlProcessInstanceListOut**

#### **XML Schema and Element Descriptions**

##### **- xmlProcessInstanceOut/xmlProcessInstanceListOut**

The data elements for the ProcessInstanceOut and ProcessInstanceListOut arguments are the same as the ProcessInstanceIn and ProcessInstanceListIn arguments.

### **xmlActivityInstanceOut/xmlActivityInstanceListOut**

#### **XML Schema and Element Descriptions**

##### **- xmlActivityInstanceOut/xmlActivityInstanceListOut**

The data elements for the ActivityInstanceOut and ActivityInstanceListOut arguments are the same as the ActivityInstanceIn and ActivityInstanceListIn arguments.

# Part Five

---

## Appendices

Part Five contains the following appendices:

- 
- 
-

---



# Appendix A

---

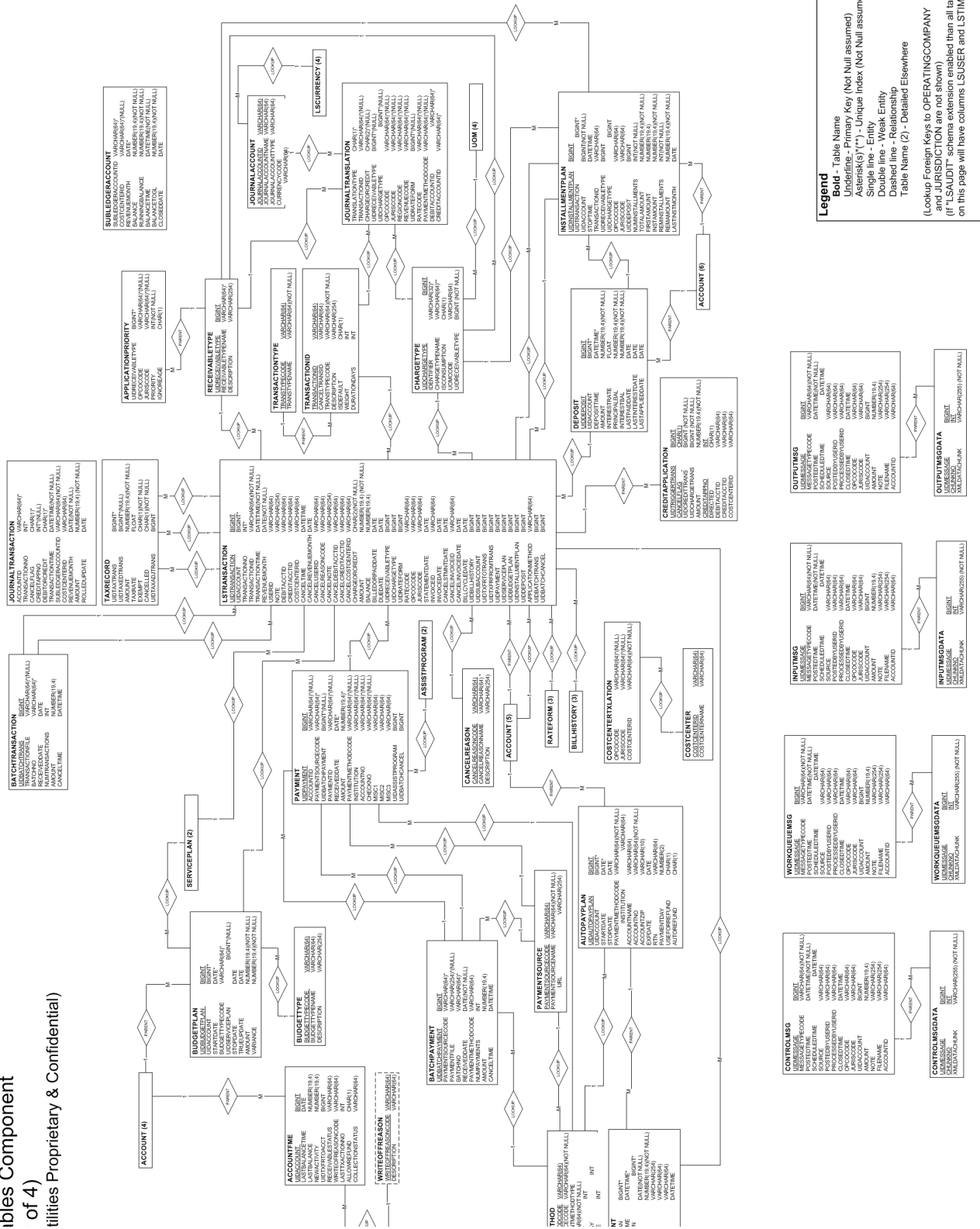
## Oracle Utilities Data Repository Receivables Component Database Schema

This appendix includes a diagram of the Oracle Utilities Data Repository Receivables Component database schema (v1.6.1.0.0) that provides details regarding the table and columns in the receivables management schema, as well as the relationships between these tables in the Oracle Utilities Data Repository. This information is very useful when writing Rules Language statements or constructing database queries. This includes:

- **Oracle Utilities Receivables Component Database Schema**
- **Oracle Utilities Receivables Component-Collections Database Schema**

# Oracle Utilities Receivables Component Database Schema

Jtilities Billing Component V1.6.0.0.0  
 ibles Component  
 of 4)  
 tilities Proprietary & Confidential)







# Appendix B

---

## Oracle Utilities Data Repository Workflow Management Database Schema

This appendix includes a diagram of the Oracle Utilities Data Repository Workflow Management database schema (v1.6.0.0.0) that provides details regarding the table and columns in the database schema, as well as the relationships between these tables in the Oracle Utilities Data Repository. This appendix includes:

- **Oracle Utilities Billing Component - Workflow Management / Reports Database Schema**









# Appendix C

---

## Messaging

This chapter provides an overview of the Messaging System, including:

- **The Purpose of the Messaging System**
- **Messaging Tables**
- **Message Types**
- **Messaging Functions**

## The Purpose of the Messaging System

During the processing of various business functions, it will be necessary to create messages for a number of reasons, such as notifying external systems that a given function has been performed, logging activity for control purposes, or maintaining work queues. The Messaging System is designed to support external interfacing, control activities, work queues, and internal messages (such as Collections processing).

An example of the Messaging System might be as follows. Whenever a financial transaction is posted against an account, a corresponding message is posted so that the activity is logged and any external systems can be notified of the update. This is an example of a message being used as both an external interface and a control activity.

---

## Messaging Tables

The Messaging System makes use of a number of tables in the Oracle Utilities Data Respository to track various message types, message queues, and messages. The following tables are used to manage information about message types and message queues in the system and the relationships between them.

### **Message Type Table (MESSAGETYPE)**

The Message Type Table contains records for each unique message type in the system.

### **Message Queue Table (MESSAGEQUEUE)**

The Message Queue Table contains records representing each available message queue or container in the system. The MESSAGEQUEUECODE attribute uniquely identifies the queue, and the MESSAGEQUEUEUETYPE attribute defines the type of the queue. The system can be extended to support many different types of message queues, including database tables, IBM MQSeries, and Microsoft MSMQ. The MESSAGEQUEUEUENAME and SERVERNAME attributes are used by the various message queue type implementations to identify the queue internally. The default message queue type supported is “LSDB”, which is a relational database table implementation in the Oracle Utilities database.

### **Message Type Queue Table (MESSAGETYPEQUEUE)**

The Message Type Queue Table is used to specify which message queues are used for each message type. A message type can have zero or more message queues. Each time a message is posted, the system will post the message to every message queue related to the message’s message type. This allows the system to be configured to post different message types to different queues depending on the requirements of the system. If a message type is not required, then it will not be related to any message queues, and the system will effectively “drop” the message.

## Message Types

The specific message types used by the Messaging System are dependent on the specific needs of the implementation. The system can handle any number of message types. The following lists the default message types used by the Oracle Utilities Receivables Component.

<b>Message Type</b>	<b>Message Type Code</b>
Account Balance Error	ACCOUNT_BALANCE_ERROR
Account Balanced	ACCOUNTS_BALANCED
Adjustment Transaction Posted	ADJ_TXACTION_POSTED
Autopayments Processed	AUTOPAYMENTS_PROCESSED
Balance Accounts Error	BALANCE_ACCOUNTS_ERROR
Batch Payment Error	BATCHPAYMENT_ERROR
Batch Payment Processed	BATCHPAYMENT_PROCESSED
Budget Charge Transaction Posted	BDGTCHG_TXACTION_POSTED
Budget Service Charge Transaction Posted	BDGTSRVCHG_TXACTION_POSTED
Budget Trueup Transaction Posted	BDGTTRP_TXACTION_POSTED
Bill Transaction Posted	BILL_TXACTION_POSTED
Cancel Transaction Error	CANCEL_TXACTION_ERROR
Collections Payment Posted	COLLECTIONS_PAYMENT
Deferred Service Charge Transaction Posted	DEFSRVCHG_TXACTION_POSTED
Installment Charge Transaction Posted	INSTCHG_TXACTION_POSTED
Journal Balance Error	JOURNAL_BALANCE_ERROR
Journal Balanced	JOURNAL_BALANCED
Payment Exception Error	PAYMENT_EXCEPTION_ERROR
Payment File Error	PAYMENTFILE_ERROR
Payment File Processed	PAYMENTFILE_PROCESSED
Payment File In	PAYMENTFILEIN
Payment File Out	PAYMENTFILEOUT
Post Transaction Error	POST_TXACTION_ERROR
Payment Transaction Posted	PYMNT_TXACTION_POSTED
Refund Transaction Posted	RFND_TXACTION_POSTED
Service Charge Transaction Posted	SRVCHG_TXACTION_POSTED
Statement Transaction Posted	STMT_TXACTION_POSTED

Message Type	Message Type Code
Sub-Ledger Rolled Up	SUBLEDGER_ROLLEDUP
Transaction Cancelled	TXACTION_CANCELLED
Unpostable Payment	UNPOSTABLE_PAYMENT
Unpostable Transaction	UNPOSTABLE_TRANSACTION
Write-Off Transaction Posted	WRTOFF_TXACTION_POSTED

## Message Queues

Message Queues can be thought of as containers for types of messages related to specific uses, such as input messages, output messages, control messages, etc. The system can support any number of message queues, but includes a number of default queues as part of the system's basic functionality.

### Default Message Queues

The default Message queues used by the Messaging System are described below.

#### Input Messages (INPUTMSG)

The Input Messages queue is used for potential messages coming into the Oracle Utilities system from external sources. This includes AutoPayment files that are required to be posted at a later date.

#### Output Messages (OUTPUTMSG)

The Output Messages queue is used for all messages heading out of the Oracle Utilities system to external systems.

#### Control Messages (CONTROLMSG)

The Control Messages queue is used for all control activities.

### **Work Queue Messages (WORKQUEUEMSG)**

The Work Queue Messages queue is used for all messages that need to be handled by a back-office user via a work queue.

### **Collection Messages (COLLECTIONMSG)**

Used for all messages related to automated collections processing.

### **Collection Events Messages (COLLEVENTMSG)**

Used to store all collection events triggered by automated collections processing. Messages in this queue will not be handled, they will simply be stored here for the purpose of maintaining a history of collection events for each account.

## **Message Queue Table Templates**

Each message queue has corresponding tables (for the “LSDB” queue type) that store the individual records associated with each message. These consist of a Message Table and a Message Data Table for each queue. For instance, there is a Message Table and Message Data Table for each of the default message queues described above.

### **Message Table**

Records in the Message Table represent actual messages. Records in the Message Table contain the following information:

- A unique ID for the message,
- Message Type (from the Message Type Table),
- Posted and Scheduled Time,
- Source of the message,
- Posted by/Processed by UserID,
- Closed Time,
- Associated Operating Company and Jurisdiction,
- Account,
- Amount (optional),
- Note, and
- Filename (optional name of a file containing the message).

### **Message Data**

One or more records in the Message Data Table store the XML data for a given message.

---

# Messaging Functions

The Messaging System provides a number of Messaging functions to support the ability to administer messages flowing through the system. These include the ability to post, retrieve, remove, and list messages, as well as other functions designed to support work queues. These functions are triggered through COM interfaces. The Messaging functions include:

## **Post Message**

The Post Message function is used to post a single message to zero or more message queues, based on the message's message type.

## **Peek Message**

The Peek Message function is used to retrieve a single message from a single message queue.

## **Remove Message**

The Remove Message function is identical to the Peek method described above, except that it will also remove the returned message from the message queue.

## **List Message**

The List Message function is used to retrieve a list of messages from a single message queue.

## **Hold, Release, Close, Reopen Message**

The Hold, Release, Close, and Reopen Message functions are used to support special message functionality required for work queues. A user “holds” a message in a work queue to indicate that the work queue item is currently being investigated. This prevents two or more people from accidentally working on the same work queue item at the same time. A held message can be subsequently “released”, or if the work queue item has been resolved then the message can be “closed”. A closed message can later be “reopened” if necessary.

## **Update Message**

The Update function is used to update an existing message in a single message queue. This function can only be used on message that have been locked via the Hold Message function.

## How the Messaging Functions Work

The Messaging “Post” function operates ‘behind the scenes’, meaning that the function is triggered automatically when certain operations are performed by one of the Oracle Utilities Receivables Component modules (such as the AR Engine) or WorkFlow Manager. For instance, when a charge or credit transaction is posted, the AR Engine triggers the Post Message function and sends a message of the appropriate message type to the appropriate message queues.

The other Messaging functions are used by the Oracle Utilities Receivables Component and WorkFlow Manager user interfaces. These include the Peek, Remove, List, Hold, Release, Close, and Reopen functions (essentially every function except Post).

In addition, the Messaging functions can be accessed by external systems through an interface.



# Appendix D

---

## Financial Management Rules Language Statements

This appendix provides detailed explanations of the Financial Management statements available in the Oracle Utilities Rules Language. Financial Management statements are used to post transactions to the Oracle Utilities Receivables Component, including:

- **Using the Financial Management Statements**
- **Deprecated Statements**

## Using the Financial Management Statements

The Financial Management statements are used to post charges or credits to the Oracle Utilities Receivables Component's Financial Engine. Each statement takes a transaction identifier as a single argument. The transaction identifier is a stem that should contain several tail attributes, as described below. Attributes marked with an asterisk (\*) are required.

Attribute	Description
ACCOUNTID	An account ID that identifies the account for posting or cancelling a transaction. If not provided, the rate schedule account context is used. It is an error if no account ID is provided and the rate schedule is not run within the context of an account.
UID	Unique ID of a posted transaction. Used with the CANCEL_TRAN statement.
TRANSACTIONID	A transaction ID for the transaction. If not provided, the default transaction ID for the transaction type is used.
REVENUEMONTH	The revenue month for the transaction. If not provided, the rate schedule bill month is used.
NOTE	A note for the transaction.
CANCELREVENUEMONTH	The revenue month for a cancelled transaction. Optional attribute used with the CANCEL_TRAN statement.
CANCELREASONCODE	The reason for cancelling a transaction. Optional attribute used with the CANCEL_TRAN statement.
CANCELNOTE	A note for a cancelled transaction. Optional attribute used with the CANCEL_TRAN statement.
CHARGEORCREDIT	Indicates whether transaction is a charge (CH) or a credit (CR). The default is charge ("CH") unless otherwise indicated.
DEFERBALANCE	Indicates whether transaction balance is deferred ("TRUE") or not ("FALSE"). The default is "FALSE" unless otherwise indicated.
AMOUNT*	The amount of the transaction.
CURRENCY	The currency code for the currency associated with the account for the transaction. This is required if the LS Currency column is populated in the Account table.
BILLEDDATE*	The billed date for the charge transaction. This is only required for charge transactions.
DUEDATE*	The due date for the charge transaction. This is only required for charge transactions.
RECEIVABLETYPENAME	The receivable type name for transactions (required for all charge type transactions, except for POST BILL, if CHARGETYPEID is not provided).

<b>Attribute</b>	<b>Description</b>
CHARGETYPEID	The charge type identifier for transactions (required for all charge type transactions, except for POST BILL, if RECEIVABLETYPENAME is not provided).
OPCOCODE	The operating company code associated with the transaction.
JURISCODE	The jurisdiction code associated with the transaction.
STATEMENTDATE	This attribute is used only with the POST STATEMENT statement. The statement date associated with the transaction.
INVOICEID	The invoice ID associated with the transaction.
INVOICEDATE	The invoice date associated with the transaction.
BILLCYCLEDATE	The bill cycle date for the transaction. If not provided, the rate schedule read date is used. It is an error if no bill cycle date is provided and the rate schedule does not have an associated read date.
BILLSTARTTIME	This attribute is used only with the POST BILL statement. The bill start time for the transaction. If not provided, the transaction time of the previous BILL transaction with the same transaction ID is used.
BILLSTOPTIME	This attribute is used only with the POST BILL statement. The bill stop time for the transaction. If not provided, the transaction time is used.
SUSPENDAUTOPAYMENT	This attribute is used only with the POST BILL statement. Indicates that automatic payments for the bill transaction should be suspended.
SERVICEPLAN*	This attribute is required for the POST SERVICE CHARGE and POST BUDGET SERVICE CHARGE statements. The service plan attribute is a stem itself that requires both STARTDATE and SERVICETYPECODE attributes; optional attributes are ADDRESS1, ADDRESS2, ADDRESS3, CITY, COUNTY, STATE, ZIP (to identify the associated premise), and LDCACCOUNTNO.
BUDGETPLAN*	This attribute is used only with the POST BUDGET SERVICE CHARGE, POST BUDGET BILL CHARGE, and POST BUDGET BILL TRUEUP statements. The budget plan attribute is a stem itself that requires STARTDATE and BUDGETTYPECODE attributes; the SERVICEPLAN attribute (to identify any associated service plan) is optional.

<b>Attribute</b>	<b>Description</b>
TAXRATE	The tax rate associated with either a TAX transaction or one or more individual TAXEDTRANSACTIONS.
TAXEDTRANSACTION<ID>	One of the taxed transactions associated with a POST TAX statement. The taxed transaction attribute is a stem itself that may contain the following attributes: UIDTRANSACTION or TRANSACTIONNO (at least one of which is required), AMOUNT, TAXAMOUNT, TAXRATE, TAXEXEMPT ("TRUE" or "FALSE").
UIDINSTALLMENTPLAN	Unique ID of associated installment plan. Either this or INSTALLMENTPLANNO below is required for the POST INSTALLMENT statement.
INSTALLMENTPLANNO	The transaction number of deferred charge transaction associated with installment plan. Either this or UIDINSTALLMENTPLAN above is required for the POST INSTALLMENT statement.
UIDDEPOSIT	Unique ID of associated deposit. Either this or DEPOSITTIME below is required for the POST DEPOSIT INTEREST and POST DEPOSIT APPLICATION statements.
DEPOSITTIME	Time of associated deposit. Either this or UIDDEPOSIT above is required for the POST DEPOSIT INTEREST and POST DEPOSIT APPLICATION statements.
DEPINTRATE	Optional interest rate for deposit. This is used by the POST DEPOSIT statement.
APPLICATIONMETHOD	Indicates how to apply the transaction against outstanding charges or credits. Valid values are "DEFERRED", "IMMEDIATE", and "INVOICEID". Default is "DEFERRED" unless otherwise indicated.
DEFACCOUNTID	Required default account id used by the POST PAYMENT statement.
SOURCECODE	Required payment source code used by the POST PAYMENT statement.
PAYMENTID	Optional payment id used by the POST PAYMENT statement.
METHODCODE	Optional payment method code used by the POST PAYMENT statement.
INSTITUTION	Optional institution name from which payment is drawn; used by the POST PAYMENT statement.
ACCOUNTNO	Optional account number from which payment is drawn; used by the POST PAYMENT statement.

Attribute	Description
CHECKNO	Optional payment check number used by the POST PAYMENT statement.
RELATEDTRANSACTIONn	Optional related transaction(s) to which credits are applied when using the POST CHARGEORCREDIT statement. If multiple related transactions are specified, credits are applied in the order specified in the Rules Language. For example, RELATEDTRANSACTION1 first, RELATEDTRANSACTION2 second, etc.
MISC1	Optional user-defined miscellaneous attribute used by the POST PAYMENT statement.
MISC2	Optional user-defined miscellaneous attribute used by the POST PAYMENT statement.
MISC3	Optional user-defined miscellaneous attribute used by the POST PAYMENT statement.

**Example:**

```

SERV_PLAN.STARTDATE = "01/01/2000";
SERV_PLAN.SERVICETYPECODE = "ELECTRIC";

BUDGET_PLAN.STARTDATE = "01/01/2000";
BUDGET_PLAN.BUDGETTYPECODE = "SIMPLE";
BUDGET_PLAN.SERVICEPLAN = "SERV_PLAN";

SERV_CHG_1.TRANSACTIONID = "350";
SERV_CHG_1.AMOUNT = 59.95;
SERV_CHG_1.CURRENCY = "USD";
SERV_CHG_1.BILLEDDATE = "07/15/2000";
SERV_CHG_1.DUEDATE = "07/30/2000";
SERV_CHG_1.CHARGETYPEID = "ELECTRIC_USAGE_CHARGE";
SERV_CHG_1.SERVICEPLAN = "SERV_PLAN";
SERV_CHG_1.BUDGETPLAN = "BUDGET_PLAN";

POST CHARGEORCREDIT SERV_CHG_1;

```

## Using User-Defined Attributes

If the Transaction Table in your database contains columns that are not included in the base schema (and therefore not among the data elements listed), you can post values to those columns by assigning values to corresponding STEM.COLUMN\_NAME identifiers in the rate schedule. In this case, the column name specified in the rate schedule must be the exact name of the column in the database. For example, if your Transaction Table contains a column called ZONE, you could post data to that column by including the following line in your rate schedule:

```
USAGE_SERV_CHG.ZONE = "ZONE_1"  
  
/* Post Service Charge Statement */  
POST STATEMENT USAGE_SERV_CHG;
```

This would post the value assigned to the USAGE\_SERV\_CHG.ZONE identifier ("ZONE\_1") to the ZONE column in the Transaction Table.

---

## Post Charge Or Credit Statement

### Purpose

The POST CHARGEORCREDIT Statement posts a charge or credit as a single transaction. The transaction may be either deferred or not deferred. An optional service plan or budget plan may be associated with the transaction. If a budget plan is provided, the plan's variance will be updated accordingly.

### Format

POST CHARGEORCREDIT statements have this format:

```
POST CHARGEORCREDIT <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements->Financials->Post Charge or Credit** from the Rules Language Editor menu bar.

The POST CHARGEORCREDIT Statement template appears.

2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post a service charge transaction based on energy usage.*

```
/* Set Service Charge Attributes */
USAGE_SERV_CHG.ACCOUNTID = ACCOUNT.ACCOUNTID;
USAGE_SERV_CHG.TRANSACTIONID = "310";
USAGE_SERV_CHG.REVENUEMONTH = BILLMONTH;
USAGE_SERV_CHG.NOTE = "Electric Energy Charge - Energy Service
Provider";

USAGE_SERV_CHG.AMOUNT = $ENERGY_CHARGE;
USAGE_SERV_CHG.CURRENCY = "USD";
USAGE_SERV_CHG.BILLEDDATE = "07/15/2000";
USAGE_SERV_CHG.DUEDATE = "08/15/2000";

USAGE_SERV_CHG.RECEIVABLETYPENAME = "ESCO ELECTRIC";
USAGE_SERV_CHG.CHARGETYPEID = "ESCO ENERGY";
USAGE_SERV_CHG.OPCODE = OPCODE;
USAGE_SERV_CHG.JURISCODE = JURISCODE;
USAGE_SERV_CHG.BILLCYCLEDATE = BILLCYCLEDATE.READDATE;

/* Post Usage Service Charge */
POST CHARGEORCREDIT USAGE_SERV_CHG;
```

## Notes

In the above example, several of the USAGE\_SERV\_CHG attributes are 'hard-coded' into the rate schedule. In actual practice, this data would probably come directly from records in the Oracle Utilities Data Repository, or through list queries. Also, the above example includes values for all the optional USAGE\_SERV\_CHG attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.



## Post Tax Statement

### Purpose

The POST TAX Statement posts a tax charge or credit transaction for a specified account. The transaction may be either deferred or not deferred. An optional service plan or budget plan may be associated with the transaction. If a budget plan is provided, the plan's variance will be updated accordingly. Additionally, the tax transaction may be associated with one or more previously posted transactions.

### Format

POST TAX statements have this format:

```
POST TAX <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements->Financials->Post Tax** from the Rules Language Editor menu bar. The POST TAX Statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post a tax transaction based on energy usage.*

```
/* Set Energy Tax Charge Attributes */
USAGE_TAX.ACCOUNTID = ACCOUNT.ACCOUNTID;
USAGE_TAX.REVENUEMONTH = BILLMONTH;
USAGE_TAX.NOTE = "Electric Energy Tax Charge - Energy Service
Provider";

USAGE_TAX.AMOUNT = $TAX_CHARGE;
USAGE_TAX.CURRENCY = "USD";
USAGE_TAX.BILLEDDATE = "07/15/2000";
USAGE_TAX.DUEDATE = "08/15/2000";

USAGE_TAX.RECEIVABLETYPENAME = "ESCO ELECTRIC";
USAGE_TAX.CHARGETYPEID = "ESCO ENERGY";
USAGE_TAX.OPCODE = OPCODE;
USAGE_TAX.JURISCODE = JURISCODE;
USAGE_TAX.BILLCYCLEDATE = BILLCYCLEDATE.READDATE;

/* Post Tax Transaction */
POST TAX USAGE_TAX;
```

## Notes

In the above example, several of the USAGE\_TAXG attributes are 'hard-coded' into the rate schedule. In actual practice, this data would probably come directly from records in the Oracle Utilities Data Repository, or through list queries. Also, the above example includes values for all the optional USAGE\_TAX attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.

## Post Installment Statement

### Purpose

The POST INSTALLMENT Statement posts a non-deferred charge transaction related to a previously created installment plan against a specified account.

### Format

POST INSTALLMENT statements have this format:

```
POST INSTALLMENT <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements->Financials->Post Installment** from the Rules Language Editor menu bar.  
The POST INSTALLMENT Statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post an installment transaction based on energy usage.*

```
/* Set installment Attributes */
USAGE_INST.ACCOUNTID = ACCOUNT.ACCOUNTID;
USAGE_INST.TRANSACTIONID = "310";
USAGE_INST.REVENUEMONTH = BILLMONTH;
USAGE_INST.NOTE = "Electric Energy Charge - Energy Service Provider";

USAGE_INST.AMOUNT = $ENERGY_CHARGE;
USAGE_INST.CURRENCY = "USD";
USAGE_INST.BILLEDDATE = "07/15/2000";
USAGE_INST.DUEDATE = "08/15/2000";

USAGE_INST.RECEIVABLETYPENAME = "ESCO ELECTRIC";
USAGE_INST.CHARGETYPEID = "ESCO ENERGY";
USAGE_INST.OPCOCODE = OPCOCODE;
USAGE_INST.JURISCODE = JURISCODE;
USAGE_INST.BILLCYCLEDATE = BILLCYCLEDATE.READDATE;

/* Post Installment */
POST INSTALLMENT USAGE_INST;
```

## Notes

In the above example, several of the USAGE\_INST attributes are 'hard-coded' into the rate schedule. In actual practice, this data would probably come directly from records in the Oracle Utilities Data Repository or through list queries. Also, the above example includes values for all the optional USAGE\_INST attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.

## Post Statement Statement

### Purpose

The POST STATEMENT Statement posts a single statement transaction against an account. The transaction indicates the current balance for the account. The account's current balance will not change.

### Format

POST STATEMENT statements have this format:

```
POST STATEMENT <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements→Financials→Post Statement** from the Rules Language Editor menu bar. The POST STATEMENT Statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post a statement transaction based on energy usage.*

```
/* Set Service Charge Attributes */
USAGE_SERV_CHG.ACCOUNTID = ACCOUNT.ACCOUNTID;
USAGE_SERV_CHG.TRANSACTIONID = "310";
USAGE_SERV_CHG.REVENUEMONTH = BILLMONTH;
USAGE_SERV_CHG.NOTE = "Electric Energy Charge - Energy Service
Provider";

USAGE_SERV_CHG.AMOUNT = $ENERGY_CHARGE;
USAGE_SERV.CHURRENCY = "USD";
USAGE_SERV_CHG.BILLEDDATE = "07/15/2000";
USAGE_SERV_CHG.DUEDATE = "08/15/2000";

USAGE_SERV_CHG.RECEIVABLETYPENAME = "ESCO ELECTRIC";
USAGE_SERV_CHG.CHARGETYPEID = "ESCO ENERGY";
USAGE_SERV_CHG.OPCODE = OPCODE;
USAGE_SERV_CHG.JURISCODE = JURISCODE;
USAGE_SERV_CHG.STATEMENTDATE = "08/01/2000";
USAGE_SERV_CHG.BILLCYCLEDATE = BILLCYCLEDATE.READDATE;

/* Post Usage Statement */
POST STATEMENT USAGE_SERV_CHG;
```

## Notes

In the above example, several of the USAGE\_SERV\_CHG attributes are 'hard-coded' into the rate schedule. In actual practice, this data would probably come directly from records in the Oracle Utilities Data Repository, or through list queries. Also, the above example includes values for all the optional USAGE\_SERV\_CHG attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.

## Post Bill Statement

### Purpose

The POST BILL Statement posts a bill transaction against an account. This will trigger the IMMEDIATE credit application process, unless the APPLICATIONMETHOD is set to “DEFERRED”. It may also initiate an autopayment for the account, if set up to do so. The account’s current balance will not change, unless DEFERBALANCE is set to “FALSE”.

### Format

POST BILL statements have this format:

```
POST BILL <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements→Financials→Post Bill** from the Rules Language Editor menu bar.  
The POST BILL Statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post a bill for the total charge to the customer.*

```
/* Set Bill Attributes */
TOTAL_BILL.ACCOUNTID = ACCOUNT.ACCOUNTID;
TOTAL_BILL.TRANSACTIONID = "3000";
TOTAL_BILL.REVENUEMONTH = BILLMONTH;
TOTAL_BILL.NOTE = "Total Bill, including customer and energy charges";

TOTAL_BILL.AMOUNT = $EFFECTIVE_REVENUE;
TOTAL_BILL.CURRENCY = "USD";
TOTAL_BILL.BILLEDDATE = "07/15/2000";
TOTAL_BILL.DUEDATE = "08/15/2000";

TOTAL_BILL.RECEIVABLETYPEID = "ESCO ELECTRIC";
TOTAL_BILL.CHARGETYPEID = "ESCO ENERGY";
TOTAL_BILL.OPCODE = OPCODE;
TOTAL_BILL.JURISCODE = JURISCODE;
TOTAL_BILL.BILLCYCLEDATE = BILLCYCLEDATE.READDATE;

/* Post Bill */
POST BILL TOTAL_BILL;
```

## Notes

In the above example, several of the TOTAL\_BILL attributes are 'hard-coded' into the rate schedule. In actual practice, this data would probably come directly from records in the Oracle Utilities Data Repository, or through list queries. Also, the above example includes values for all the optional TOTAL\_BILL attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.

The POST BILL statements returns the following tail identifiers to allow utilizing a created transaction in subsequent processing:

- **UIDTRANSACTION:** The UID of the transaction
- **TRANSACTIONTIME:** The time of the transaction
- **TRANSACTIONNO:** The transaction number



## Post Payment Statement

### Purpose

The POST PAYMENT Statement posts a payment transaction against an account. This will trigger the IMMEDIATE credit application process, unless the APPLICATIONMETHOD is set to “DEFERRED” or “INVOICEID”. The account’s current balance will change.

### Format

POST PAYMENT statements have this format:

```
POST PAYMENT <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements→Financials→Post Payment** from the Rules Language Editor menu bar. The POST PAYMENT Statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post a payment.*

```
/* Set Payment Attributes */
PAYMENT.ACCOUNTID = ACCOUNT.ACCOUNTID;
PAYMENT.DEFACCOUNTID = "99999";
PAYMENT.SOURCECODE = "LOCKBOX";

PAYMENT.AMOUNT = "$90.00";
PAYMENT.CURRENCY = "USD";

/* Post Payment */
POST PAYMENT PAYMENT;
```

## Notes

In the preceding example, several of the PAYMENT attributes are 'hard-coded' into the rate schedule. In actual practice, this data could also come directly from records in the Oracle Utilities Data Repository or through list queries.

---

## Post Adjustment Statement

### Purpose

The POST ADJUSTMENT Statement posts an adjustment transaction against an account. This will, by default, trigger the credit application process if the adjustment is a credit, unless the APPLICATIONMETHOD is overridden. The account's current balance will change by default unless the DEFERBALANCE is set to TRUE.

### Format

POST ADJUSTMENT statements have this format:

```
POST ADJUSTMENT <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements->Financials->Post Adjustment** from the Rules Language Editor menu bar.

The POST ADJUSTMENT Statement template appears.

2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post a credit adjustment.*

```
/* Set Payment Attributes */
CHG_ADJUST.ACCOUNTID = ACCOUNT.ACCOUNTID;
CHG_ADJUST.CHARGEORCREDIT = "CR"

CHG_ADJUST.AMOUNT = "$90.00";
CHG_ADJUST.CURRENCY = "USD";

/* Post Adjustment */
POST ADJUSTMENT CHG_ADJUST;
```

## Notes

In the preceding example, several of the ADJUSTMENT attributes are 'hard-coded' into the rate schedule. In actual practice, this data could also come directly from records in the Oracle Utilities Data Repository or through list queries.

## Post Refund Statement

### Purpose

The POST Refund Statement posts a refund transaction against an account. This will trigger the IMMEDIATE credit application process, unless the APPLICATIONMETHOD is set to “DEFERRED” or “INVOICEID”. The account’s current balance will change.

### Format

POST REFUND statements have this format:

```
POST REFUND <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements→Financials→Post Refund** from the Rules Language Editor menu bar. The POST REFUND Statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post a refund for the total charge to the customer.*

```
/* Set Bill Attributes */
BILL_REFUND.ACCOUNTID = ACCOUNT.ACCOUNTID;
BILL_REFUND.TRANSACTIONID = "3000";
BILL_REFUND.REVENUEMONTH = BILLMONTH;
BILL_REFUND.NOTE = "Refund for Total Bill, including customer and
energy charges";

BILL_REFUND.AMOUNT = $EFFECTIVE_REVENUE;
BILL_REFUND.CURRENCY = "USD";

BILL_REFUND.RECEIVABLETYPEID = "ESCO ELECTRIC";
BILL_REFUND.CHARGETYPEID = "ESCO ENERGY";
BILL_REFUND.OPCODE = OPCODE;
BILL_REFUND.JURISCODE = JURISCODE;
BILL_REFUND.BILLCYCLEDATE = BILLCYCLEDATE.READDATE;

BILL_REFUND.APPLICATIONMETHOD = "IMMEDIATE";
BILL_REFUND.DEFERBALANCE = "FALSE";

/* Post Refund */
POST REFUND BILL_REFUND;
```

## Notes

In the above example, several of the BILL\_REFUND attributes are 'hard-coded' into the rate schedule. In actual practice, this data would probably come directly from records in the Oracle Utilities Data Repository, or through list queries. Also, the above example includes values for all the optional BILL\_REFUND attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.

## Post Writeoff Statement

### Purpose

The POST WRITEOFF Statement is used to write off an account. All transactions for the account with an outstanding balance will be written off. This will trigger the IMMEDIATE credit application process. The account's current balance will change.

### Format

POST WRITEOFF statements have this format:

```
POST WRITEOFF <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements->Financials->Post Writeoff** from the Rules Language Editor menu bar. The POST WRITEOFF Statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Write off an account.*

```
/* Set Write Off Attributes */  
ACCT_WRITEOFF.ACCOUNTID = ACCOUNT.ACCOUNTID;
```

```
/* Post Write Off */  
POST WRITEOFF ACCT_WRITEOFF;
```



## Post Deposit Statement

### Purpose

The POST DEPOSIT Statement posts a deposit charge transaction against an account. The account's current balance will change by default unless DEFERBALANCE is set to TRUE.

### Format

POST DEPOSIT statements have this format:

```
POST DEPOSIT <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements→Financials→Post Deposit** from the Rules Language Editor menu bar. The POST DEPOSIT Statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post a deposit.*

```
/* Set Deposit Attributes */
ACCT_DEP.ACCOUNTID = ACCOUNT.ACCOUNTID;
ACCT_DEP.DEPINTRATE = FACTOR[DEPOSIT_INT_RATE].VALUE

ACCT_DEP.AMOUNT = "$90.00";
ACCT_DEP.CURRENCY = "USD";

/* Post Deposit */
POST DEPOSIT ACCT_DEP;
```

## Post Deposit Interest Statement

### Purpose

The POST DEPOSIT INTEREST Statement posts deposit interest as a single transaction.

### Format

POST DEPOSIT INTEREST statements have this format:

```
POST DEPOSIT INTEREST <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements->Financials->Post Deposit Interest** from the Rules Language Editor menu bar.  
The POST DEPOSIT INTEREST Statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post deposit interest.*

```
/* Set Deposit Interest Attributes */
DEP_INT.ACCOUNTID = ACCOUNT.ACCOUNTID;

DEP_INT.AMOUNT = $DEPOSIT_INTEREST;
DEP_INT.CURRENCY = "USD";

DEP_INT.OPCOCODE = OPCOCODE;
DEP_INT.JURISCODE = JURISCODE;

/* Post Deposit Interest */
POST DEPOSIT INTEREST USAGE_SERV_CHG;
```

## Notes

In the preceding example, several of the DEP\_INT attributes are 'hard-coded' into the rate schedule. In actual practice, this data would probably come directly from records in the Oracle Utilities Data Repository, or through list queries. Also, the above example includes values for all the optional DEP\_INT attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.

## Post Deposit Application Statement

### Purpose

The POST DEPOSIT APPLICATION Statement is used to apply a deposit as a single transaction.

### Format

POST DEPOSIT APPLICATION statements have this format:

```
POST DEPOSIT APPLICATION <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements→Financials→Post Deposit Application** from the Rules Language Editor menu bar.  
The POST DEPOSIT APPLICATION Statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post a deposit application.*

```
/* Set deposit application attributes */
DEP_APP.ACCOUNTID = ACCOUNT.ACCOUNTID;

DEP_APP.AMOUNT = $ENERGY_CHARGE;
DEP_APP.CURRENCY = "USD";

DEP_APP.OPPCODE = OPCODE;
DEP_APP.JURISCODE = JURISCODE;

/* Post Deposit Application */
POST DEPOSIT APPLICATION DEP_APP;
```

## Notes

In the above example, several of the DEP\_APP attributes are 'hard-coded' into the rate schedule. In actual practice, this data would probably come directly from records in the Oracle Utilities Data Repository, or through list queries. Also, the above example includes values for all the optional DEP\_APP attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.

---

## Cancel Transaction Statement

### Purpose

The CANCEL\_TRAN statement cancels a single transaction.

### Format

CANCEL\_TRAN statements have this format:

```
CANCEL_TRAN <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the following:
  - UID
  - ACCOUNTID
  - TRANSACTIONID
  - CANCELREVENUEMONTH (optional)
  - CANCELREASONCODE (optional)
  - CANCELNOTE (optional)

**Note:** Either the UIDTRANSACTION or the ACCOUNTID and TRANSACTIONID are required to identify the specific transaction to be cancelled. See **Using the Financial Management Statements** on page 7-2 for more information about attributes used with Oracle Utilities Receivables Component Rules Language statements.

### To Create

1. Select **Statements→Financials→Cancel Transaction** from the Rules Language Editor menu bar.  
The CANCEL\_TRAN statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Cancel a service charge transaction based on energy usage.*

```
/* Set Cancel Attributes */
CANCEL_SERV_CHG.UID = 24579;
CANCEL_SERV_CHG.CANCELNOTE = "Cancelled";
CANCEL_SERV_CHG.CANCELREASONCODE = "ERROR";
CANCEL_SERV_CHG.CANCELREVENUEMONTH = BILLMONTH;
```

```
/* Cancel Service Charge */
CANCEL_TRAN CANCEL_SERV_CHG;
```

or

*Cancel a service charge transaction based on energy usage.*

```
/* Set Cancel Attributes */
CANCEL_SERV_CHG.ACCOUNTID = ACCOUNT.ACCOUNTUID;
CANCEL_SERV_CHG.TRANSACTIONID = 110;
CANCEL_SERV_CHG.CANCELNOTE = "Cancelled";
CANCEL_SERV_CHG.CANCELREASONCODE = "ERROR";
CANCEL_SERV_CHG.CANCELREVENUEMONTH = BILLMONTH;
```

```
/* Cancel Service Charge */
CANCEL_TRAN CANCEL_SERV_CHG;
```

## Notes

In the above example, several of the CANCEL\_SERV\_CHG attributes are 'hard-coded' into the rate schedule. In actual practice, this data would probably come directly from records in the Oracle Utilities Data Repository, or through list queries. Also, the above example includes values for all the optional CANCEL\_SERV\_CHG attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.



## CALCULATE\_LATEPAYMENT Function

### Purpose

The CALCULATE\_LATEPAYMENT function is used to calculate a late payment fee based on an account's outstanding balance and current collections status.

### Format

Statements using the CALCULATE\_LATEPAYMENT function have this format:

```
<identifier> = CALCULATE_LATEPAYMENT (<ACCOUNTID> , <DATETYPE> ,
<COLLECTIONSTATUS> ) ;
```

#### Where:

- <ACCOUNTID> The Account ID of the account.
- <DATETYPE> The date type used for the calculation. Valid values include:
  - TRANSACTIONTIME
  - STATEMENTDATE
  - INVOICEDATE
  - DUEDATE
- <COLLECTIONSTATUS> The account's current collection status, from the Account Oracle Utilities Receivables Component table.

The function will return a structure that includes:

- The calculated late payment fee
- The maximum amount the late payment should be for the account.

### Example

*Calculate latepayment fees for each account in the BACK\_OFFICE list based on INVOICEDATE.*

```
FOR EACH ACCT IN LIST BACK_OFFICE
  ACCT_ID = ACCOUNTS.ACCOUNTID
  STATUS = ACCOUNTFME.COLLECTIONSTATUS
  ACCOUNT_LATE_FEE = CALCULATE_LATEPAYMENT(ACCT_ID, "INVOICEDATE",
STATUS)
  ...
END FOR;
```

## FMGETBILLINFO Function

### Purpose

The FMGETBILLINFO function is used to gather bill information for an account.

### Format

Statements using the FMGETBILLINFO function have this format:

```
<identifier> = FMGETBILLINFO [( <ID> , <DATE> )];
```

#### Where:

- <ID> <DATE> *Optional.* Account ID and date. If the account ID is not provided, the account processed in the rate form will be used. If the date is not provided, the current date will be used.

The function will return a structure (stem) that includes:

- The Account's Receivable Status
- The Account's Current Balance
- The Account's Past Due Balance.

### Example

*Get the account bill info for account ID BACK-OFFICE-1 for August 15, 2000.*

```
ACCOUNT_BILL_INFO = FMGETBILLINFO (BACK-OFFICE-1, '08/15/2000')
```

## PROCESSAUTOPAYMENT Function

### Purpose

The PROCESSAUTOPAYMENT function is used to process an automatic payment for an account. This function is typically used if normal automatic payments for an account have been suspended.

### Format

Statements using the PROCESSAUTOPAYMENT function have this format:

```
<identifier> = PROCESSAUTOPAYMENT (<STEM>);
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

The function return zero (0) if successful.

### Example

*Process an automatic payment after posting a bill.*

```
/* Set Bill Attributes */
TOTAL_BILL.ACCOUNTID = ACCOUNT.ACCOUNTID;
TOTAL_BILL.TRANSACTIONID = "3000";
TOTAL_BILL.REVENUEMONTH = BILLMONTH;
TOTAL_BILL.NOTE = "Total Bill, including customer and energy charges";

TOTAL_BILL.AMOUNT = $EFFECTIVE_REVENUE;
TOTAL_BILL.CURRENCY = "USD";
TOTAL_BILL.BILLEDDATE = "07/15/2000";
TOTAL_BILL.DUEDATE = "08/15/2000";

TOTAL_BILL.RECEIVABLETYPENAME = "ESCO ELECTRIC";
TOTAL_BILL.CHARGETYPEID = "ESCO ENERGY";
TOTAL_BILL.OPCODE = OPCOCODE;
TOTAL_BILL.JURISCODE = JURISCODE;
TOTAL_BILL.SUSPENDAUTOPAYMENT = "TRUE";
TOTAL_BILL.BILLCYCLEDATE = BILLCYCLEDATE.READDATE;

/* Post Bill */
POST BILL TOTAL_BILL;
...
...
/* Process autopayment */
PAP = PROCESSAUTOPAYMENT (TOTAL_BILL);
```

## Deprecated Statements

The following statements have been replaced or fallen into disuse. Use the POST CHARGEORCREDIT statement instead of the following statements where possible.

### Post Service Charge Statement

#### Purpose

The POST SERVICE CHARGE Statement posts a service charge against an account associated with a service plan. The account's current balance should increase by the amount of the charge, unless the DEFERBALANCE is set to "TRUE".

#### Format

POST SERVICE CHARGE statements have this format:

```
POST SERVICE CHARGE <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

#### To Create

1. Select **Statements->Financials->Post Service Charge** from the Rules Language Editor menu bar.

The POST SERVICE CHARGE Statement template appears.

2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post a service charge for energy usage.*

```

/* Set Service Charge Attributes */
USAGE_SERV_CHG.ACCOUNTID = ACCOUNT.ACCOUNTID;
USAGE_SERV_CHG.TRANSACTIONID = "310";
USAGE_SERV_CHG.REVENUEMONTH = BILLMONTH;
USAGE_SERV_CHG.NOTE = "Electric Energy Charge - Energy Service
Provider";

USAGE_SERV_CHG.AMOUNT = $ENERGY_CHARGE;
USAGE_SERV_CHG.BILLEDDATE = "07/15/2000";
USAGE_SERV_CHG.DUEDATE = "08/15/2000";

USAGE_SERV_CHG.RECEIVABLETYPENAME = "ESCO ELECTRIC";
USAGE_SERV_CHG.CHARGETYPEID = "ESCO ENERGY";
USAGE_SERV_CHG.OPCOCODE = OPCOCODE;
USAGE_SERV_CHG.JURISCODE = JURISCODE;
USAGE_SERV_CHG.BILLCYCLEDATE = BILLCYCLEDATE.READDATE;

/* Post Usage Service Charge */
POST SERVICE CHARGE USAGE_SERV_CHG;

```

## Notes

In the above example, several of the USAGE\_SERV\_CHG attributes are 'hard-coded' into the rate schedule. In actual practice, this data would probably come directly from records in the Oracle Utilities Data Repository, or through list queries. Also, the above example includes values for all the optional USAGE\_SERV\_CHG attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.

If the Transaction Table in your database contains columns that are not included in the base schema (and therefore not among the data elements listed under **Using the Financial Management Statements**), you can post values to those columns by assigning values to corresponding STEM.COLUMN\_NAME identifiers in the rate schedule. In this case, the column name specified in the rate schedule must be the exact name of the column in the database. For example, if your Transaction Table contains a column called ZONE, you could post data to that column by including the following line in your rate schedule:

```

USAGE_SERV_CHG.ZONE = "ZONE_1"

/* Post Usage Service Charge */
POST SERVICE CHARGE USAGE_SERV_CHG;

```

This would post the value assigned to the USAGE\_SERV\_CHG.ZONE identifier ("ZONE\_1") to the ZONE column in the Transaction Table.

## Post Deferred Service Charge Statement

### Purpose

The POST DEFERRED SERVICE CHARGE Statement posts a deferred service charge against an account associated with a service plan. The account's current balance will not change.

### Format

POST DEFERRED SERVICE CHARGE statements have this format:

```
POST DEFERRED SERVICE CHARGE <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements->Financials->Post Deferred Service Charge** from the Rules Language Editor menu bar.  
The POST DEFERRED SERVICE CHARGE Statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post a deferred service charge for energy usage.*

```

/* Set Deferred Service Charge Attributes */
USAGE_SERV_CHG.ACCOUNTID = ACCOUNT.ACCOUNTID;
USAGE_SERV_CHG.TRANSACTIONID = "310";
USAGE_SERV_CHG.REVENUEMONTH = BILLMONTH;
USAGE_SERV_CHG.NOTE = "Electric Energy Charge - Energy Service
Provider";

USAGE_SERV_CHG.AMOUNT = $ENERGY_CHARGE;
USAGE_SERV_CHG.BILLEDDATE = "07/15/2000";
USAGE_SERV_CHG.DUEDATE = "08/15/2000";

USAGE_SERV_CHG.RECEIVABLETYPENAME = "ESCO ELECTRIC";
USAGE_SERV_CHG.CHARGETYPEID = "ESCO ENERGY";
USAGE_SERV_CHG.OPCOCODE = OPCOCODE;
USAGE_SERV_CHG.JURISCODE = JURISCODE;
USAGE_SERV_CHG.BILLCYCLEDATE = BILLCYCLEDATE.READDATE;

/* Post Usage Service Charge */
POST DEFERRED SERVICE CHARGE USAGE_SERV_CHG;

```

## Notes

In the above example, several of the USAGE\_SERV\_CHG attributes are 'hard-coded' into the rate schedule. In actual practice, this data would probably come directly from records in the Oracle Utilities Data Repository, or through list queries. Also, the above example includes values for all the optional USAGE\_SERV\_CHG attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.

If the Transaction Table in your database contains columns that are not included in the base schema (and therefore not among the data elements listed under **Using the Financial Management Statements**), you can post values to those columns by assigning values to corresponding STEM.COLUMN\_NAME identifiers in the rate schedule. In this case, the column name specified in the rate schedule must be the exact name of the column in the database. For example, if your Transaction Table contains a column called ZONE, you could post data to that column by including the following line in your rate schedule:

```

USAGE_SERV_CHG.ZONE = "ZONE_1"

/* Post Usage Service Charge */
POST DEFERRED SERVICE CHARGE USAGE_SERV_CHG;

```

This would post the value assigned to the USAGE\_SERV\_CHG.ZONE identifier ("ZONE\_1") to the ZONE column in the Transaction Table.

## Post Budget Service Charge Statement

### Purpose

The POST BUDGET SERVICE CHARGE Statement posts a service charge against an account associated with a service plan and a budget plan. The account's current balance will not change; however, the budget plan variance will increase by the amount of the charge.

### Format

POST BUDGET SERVICE CHARGE statements have this format:

```
POST BUDGET SERVICE CHARGE <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements->Financials->Post Budget Service Charge** from the Rules Language Editor menu bar.  
The POST BUDGET SERVICE CHARGE Statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.



## Example

*Post a budget service charge for energy usage.*

```

/* Set Service Plan Attributes */
SERV_PLAN.STARTDATE = "01/01/1998";
SERV_PLAN.SERVICETYPECODE = "ELECTRIC";

/* Set Budget Plan Attributes */
BUDGET_PLAN.STARTDATE = "01/01/1998";
BUDGET_PLAN.BUDGETTYPECODE = "BUDGETELECTRIC";
BUDGET_PLAN.SERVICEPLAN = "SERV_PLAN";

/* Set Budget Service Charge Attributes */
USAGE_SERV_CHG.ACCOUNTID = ACCOUNT.ACCOUNTID;
USAGE_SERV_CHG.TRANSACTIONID = "1105";
USAGE_SERV_CHG.REVENUEMONTH = BILLMONTH;
USAGE_SERV_CHG.NOTE = "Electric Energy Charge - Energy Service
Provider";
USAGE_SERV_CHG.AMOUNT = $ENERGY_CHARGE;
USAGE_SERV_CHG.BILLEDDATE = "07/15/2000";
USAGE_SERV_CHG.DUEDATE = "08/15/2000";
USAGE_SERV_CHG.RECEIVABLETYPENAME = "ESCO ELECTRIC";
USAGE_SERV_CHG.CHARGETYPEID = "ESCO ENERGY";
USAGE_SERV_CHG.OPCOCODE = OPCOCODE;
USAGE_SERV_CHG.JURISCODE = JURISCODE;
USAGE_SERV_CHG.BILLCYCLEDATE = BILLCYCLEDATE.READDATE;

USAGE_SERV_CHG.SERVICEPLAN = "SERV_PLAN";
USAGE_SERV_CHG.BUDGETPLAN = "BUDGET_PLAN";

/* Post Usage Budget Service Charge */
POST BUDGET SERVICE CHARGE USAGE_SERV_CHG;

```

## Notes

In the above example, several of the USAGE\_SERV\_CHG attributes are 'hard-coded' into the rate schedule. In actual practice, this data would probably come directly from records in the Oracle Utilities Data Repository, or through list queries. Also, the above example includes values for all the optional USAGE\_SERV\_CHG attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.

If the Transaction Table in your database contains columns that are not included in the base schema (and therefore not among the data elements listed under **Using the Financial Management Statements**), you can post values to those columns by assigning values to corresponding STEM.COLUMN\_NAME identifiers in the rate schedule. In this case, the column name specified in the rate schedule must be the exact name of the column in the database. For example, if your Transaction Table contains a column called ZONE, you could post data to that column by including the following line in your rate schedule:

```

USAGE_SERV_CHG.ZONE = "ZONE_1"

/* Post Usage Service Charge */
POST BUDGET SERVICE CHARGE USAGE_SERV_CHG;

```

This would post the value assigned to the USAGE\_SERV\_CHG.ZONE identifier ("ZONE\_1") to the ZONE column in the Transaction Table.

## Post Budget Bill Charge Statement

### Purpose

The POST BUDGET BILL CHARGE Statement posts a budget bill charge against an account associated with a budget plan. The account's current balance should increase by the amount of the charge, and the budget plan variance should decrease by the amount of the charge.

### Format

POST BUDGET BILL CHARGE statements have this format:

```
POST BUDGET BILL CHARGE <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements->Financials->Post Budget Bill Charge** from the Rules Language Editor menu bar.  
The POST BUDGET BILL CHARGE Statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post a budget bill charge for energy usage.*

```

/* Set Service Plan Attributes */
SERV_PLAN.STARTDATE = "01/01/1998";
SERV_PLAN.SERVICETYPECODE = "ELECTRIC";

/* Set Budget Plan Attributes */
BUDGET_PLAN.STARTDATE = "01/01/1998";
BUDGET_PLAN.BUDGETTYPECODE = "BUDGETELECTRIC";
BUDGET_PLAN.SERVICEPLAN = "SERV_PLAN";

/* Set Budget Bill Attributes */
USAGE_BUDGET_CHG.ACCOUNTID = ACCOUNT.ACCOUNTID;
USAGE_BUDGET_CHG.TRANSACTIONID = "1100";
USAGE_BUDGET_CHG.REVENUEMONTH = BILLMONTH;
USAGE_BUDGET_CHG.NOTE = "Electric Energy Charge - Energy Service
Provider";
USAGE_BUDGET_CHG.AMOUNT = $ENERGY_CHARGE;
USAGE_BUDGET_CHG.BILLEDDATE = "07/15/2000";
USAGE_BUDGET_CHG.DUEDATE = "08/15/2000";
USAGE_BUDGET_CHG.RECEIVABLETYPENAME = "ESCO ELECTRIC";
USAGE_BUDGET_CHG.CHARGETYPEID = "ESCO ENERGY";
USAGE_BUDGET_CHG.OPCOCODE = OPCOCODE;
USAGE_BUDGET_CHG.JURISCODE = JURISCODE;
USAGE_BUDGET_CHG.BILLCYCLEDATE = BILLCYCLEDATE.READDATE;

USAGE_BUDGET_CHG.SERVICEPLAN = "SERV_PLAN";
USAGE_BUDGET_CHG.BUDGETPLAN = "BUDGET_PLAN";

/* Post Budget Bill Usage Charge */
POST BUDGET BILL CHARGE USAGE_BUDGET_CHG;

```

## Notes

In the above example, several of the USAGE\_BUDGET\_CHG attributes are ‘hard-coded’ into the rate schedule. In actual practice, this data would probably come directly from records in the Oracle Utilities Data Repository, or through list queries. Also, the above example includes values for all the optional USAGE\_BUDGET\_CHG attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.

If the Transaction Table in your database contains columns that are not included in the base schema (and therefore not among the data elements listed under **Using the Financial Management Statements**), you can post values to those columns by assigning values to corresponding STEM.COLUMN\_NAME identifiers in the rate schedule. In this case, the column name specified in the rate schedule must be the exact name of the column in the database. For example, if your Transaction Table contains a column called ZONE, you could post data to that column by including the following line in your rate schedule:

```
USAGE_BUDGET_CHG.ZONE = "ZONE_1"
```

```

/* Post Budget Bill Usage Charge */
POST BUDGET BILL CHARGE USAGE_BUDGET_CHG;

```

This would post the value assigned to the USAGE\_BUDGET\_CHG.ZONE identifier (“ZONE\_1”) to the ZONE column in the Transaction Table.

## Post Budget Bill Trueup Statement

### Purpose

The POST BUDGET BILL TRUEUP Statement posts a budget bill true-up charge or credit against an account associated with a budget plan. The account's current balance should increase (if a charge) or decrease (if a credit) by the amount of the transaction. The budget plan variance should decrease (if a charge) or increase (if a credit) by the amount of the transaction.

### Format

POST BUDGET BILL TRUEUP statements have this format:

```
POST BUDGET SERVICE CHARGE <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements->Financials->Post Budget Bill Trueup** from the Rules Language Editor menu bar.  
The POST BUDGET BILL TRUEUP Statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post a budget bill true-up charge based on energy usage.*

```

/* Set Service Plan Attributes */
SERV_PLAN.STARTDATE = "01/01/1998";
SERV_PLAN.SERVICETYPECODE = "ELECTRIC";

/* Set Budget Plan Attributes */
BUDGET_PLAN.STARTDATE = "01/01/1998";
BUDGET_PLAN.BUDGETTYPECODE = "BUDGETELECTRIC";
BUDGET_PLAN.SERVICEPLAN = "SERV_PLAN";

/* Set Budget Bill Trueup Attributes */
BUDGET_TRUEUP_USAGE_CHG.ACCOUNTID = ACCOUNT.ACCOUNTID;
BUDGET_TRUEUP_USAGE_CHG.TRANSACTIONID = "1130";
BUDGET_TRUEUP_USAGE_CHG.REVENUEMONTH = BILLMONTH;
BUDGET_TRUEUP_USAGE_CHG.NOTE = "Budget Bill Trueup - Electric Energy
Charge";
BUDGET_TRUEUP_USAGE_CHG.CHARGEORCREDIT = "CH"
BUDGET_TRUEUP_USAGE_CHG.AMOUNT = $ENERGY_CHARGE;
BUDGET_TRUEUP_USAGE_CHG.BILLEDDATE = "07/15/2000";
BUDGET_TRUEUP_USAGE_CHG.DUEDATE = "08/15/2000";
BUDGET_TRUEUP_USAGE_CHG.RECEIVABLETYPENAME = "ESCO ELECTRIC";
BUDGET_TRUEUP_USAGE_CHG.CHARGETYPEID = "ESCO ENERGY";
BUDGET_TRUEUP_USAGE_CHG.OPCOCODE = OPCOCODE;
BUDGET_TRUEUP_USAGE_CHG.JURISCODE = JURISCODE;
BUDGET_TRUEUP_USAGE_CHG.BILLCYCLEDATE = BILLCYCLEDATE.READDATE;
BUDGET_TRUEUP_USAGE_CHG.SERVICEPLAN = "SERV_PLAN";
BUDGET_TRUEUP_USAGE_CHG.BUDGETPLAN = "BUDGET_PLAN";

/* Post Budget Bill Trueup Usage Charge */
POST BUDGET BILL TRUEUP BUDGET_TRUEUP_USAGE_CHG;

```

## Notes

In the above example, several of the BUDGET\_TRUEUP\_USAGE\_CHG attributes are 'hard-coded' into the rate schedule. In actual practice, this data would probably come directly from records in the Oracle Utilities Data Repository, or through list queries. Also, the above example includes values for all the optional BUDGET\_TRUEUP\_USAGE\_CHG attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.

If the Transaction Table in your database contains columns that are not included in the base schema (and therefore not among the data elements listed under **Using the Financial Management Statements**), you can post values to those columns by assigning values to corresponding STEM.COLUMN\_NAME identifiers in the rate schedule. In this case, the column name specified in the rate schedule must be the exact name of the column in the database. For example, if your Transaction Table contains a column called ZONE, you could post data to that column by including the following line in your rate schedule:

```

BUDGET_TRUEUP_USAGE_CHG.ZONE = "ZONE_1"
/* Post Budget Bill Trueup Usage Charge */
POST BUDGET BILL TRUEUP BUDGET_TRUEUP_USAGE_CHG;

```

This would post the value assigned to the BUDGET\_TRUEUP\_USAGE\_CHG.ZONE identifier ("ZONE\_1") to the ZONE column in the Transaction Table.

## Post Installment Charge Statement

### Purpose

The POST INSTALLMENT CHARGE Statement posts an installment charge against an account. The account's current balance should increase by the amount of the charge, unless the DEFERBALANCE is set to "TRUE".

### Format

POST INSTALLMENT CHARGE statements have this format:

```
POST INSTALLMENT CHARGE <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Financial Management Statements**.

### To Create

1. Select **Statements->Financials->Post Installment Charge** from the Rules Language Editor menu bar.  
The POST INSTALLMENT CHARGE Statement template appears.
2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post an installment charge.*

```

/* Set Installment Charge Attributes */
BILL_INST_CHG.ACCOUNTID = ACCOUNT.ACCOUNTID;
BILL_INST_CHG.TRANSACTIONID = "1510";
BILL_INST_CHG.REVENUEMONTH = BILLMONTH;
BILL_INST_CHG.NOTE = "Bill Installment";

BILL_INST_CHG.AMOUNT = $INSTALL_CHG;
BILL_INST_CHG.BILLEDDATE = "07/15/2000";
BILL_INST_CHG.DUEDATE = "08/15/2000";

BILL_INST_CHG.RECEIVABLETYPENAME = "ESCO ELECTRIC";
BILL_INST_CHG.CHARGETYPEID = "ESCO ENERGY";
BILL_INST_CHG.OPCOCODE = OPCOCODE;
BILL_INST_CHG.JURISCODE = JURISCODE;
BILL_INST_CHG.BILLCYCLEDATE = BILLCYCLEDATE.READDATE;

/* Post Installment Charge */
POST INSTALLMENT CHARGE BILL_INST_CHG;

```

## Notes

In the above example, several of the BILL\_INST\_CHG attributes are ‘hard-coded’ into the rate schedule. In actual practice, this data would probably come directly from records in the Oracle Utilities Data Repository, or through list queries. Also, the above example includes values for all the optional BILL\_INST\_CHG attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.

If the Transaction Table in your database contains columns that are not included in the base schema (and therefore not among the data elements listed under **Using the Financial Management Statements**), you can post values to those columns by assigning values to corresponding STEM.COLUMN\_NAME identifiers in the rate schedule. In this case, the column name specified in the rate schedule must be the exact name of the column in the database. For example, if your Transaction Table contains a column called ZONE, you could post data to that column by including the following line in your rate schedule:

```

BILL_INST_CHG.ZONE = "ZONE_1"

/* Post Installment Charge */
POST INSTALLMENT CHARGE BILL_INST_CHG;

```

This would post the value assigned to the BILL\_INST\_CHG.ZONE identifier (“ZONE\_1”) to the ZONE column in the Transaction Table.





# Appendix E

---

## Workflow Management Rules Language Statements

This chapter provides detailed explanations of the workflow management statements available in the Oracle Utilities Rules Language. These statements are used to work with processes and events using Workflow Management including:

- **Using the Workflow Management Statements**

## Using the Workflow Management Statements

The Workflow Management statements are used with processes in run via the workflow management functionality of Oracle Utilities Billing Component. Each statement takes as a single argument an identifier. The identifier is a stem that should contain several tail attributes, as described below.

Attribute	Description
UID	The process instance UID. Required input for the Process Suspend, Process Resume, and Process Terminate statements, and is the output of the Process Start Statement.
NOTE	A process instance note. Required input for the Process Terminate Statement; optional for all others.
UIDACCOUNT	Optional account UID, used as input for the Process Start or Process Event statements. If an account is to be associated with the process, either this or the ACCOUNTID (below) must be provided.
ACCOUNTID	Optional account ID, used as for input for the Process Start or Process Event statements. If an account is to be associated with the process, either this or the UIDACCOUNT (above) must be provided.
PROCESSNAME	Name of the process model to be started by the Process Start Statement. Required input for the Process Start Statement.
OPCOCODE	Optional operating company code of the process model to be started by the Process Start Statement.
JURISCODE	Optional jurisdiction code of the process model to be started by the Process Start Statement.
EVENTCODE	Required event type code for the Process Event Statement.
CONTEXT	Optional context for the Process Start or Process Event statements. Should be set to an identifier that is either an XML DOM node or XML DOM document.

### Example:

```
COLLECT_PROC.PROCESSNAME = "Collections";
COLLECT_PROC.OPCOCODE = "AGL";
COLLECT_PROC.ACCOUNTID = ACCOUNT.ID;
```

```
PROCESS START COLLECT_PROC;
```

## Process Start Statement

### Purpose

The PROCESS START Statement is used to start a new process instance.

### Format

PROCESS START statements have this format:

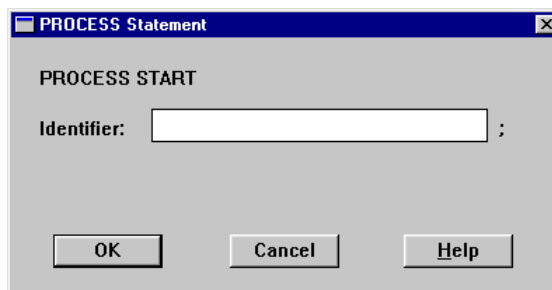
```
PROCESS START <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Workflow Management Statements**.

### To Create

1. Select **Statements->Workflow->Process Start** from the Rules Language Editor menu bar. The PROCESS START Statement template appears.



2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Start a process instance of the COLLECT\_PROC process.*

```
/* Set the process context */
PROC_CONTEXT = DOMDOCLOADFILE ("COLLECT_CONTEXT.XML")

/* Set Process Instance Attributes */
COLLECT_PROC.ACCOUNTID = ACCOUNT.ID;
COLLECT_PROC.PROCESSNAME = "Collections";
COLLECT_PROC.OPCOCODE = "AGL";
COLLECT_PROC.JURISCODE = "GA";
COLLECT_PROC.CONTEXT = PROC_CONTEXT;

/* Start the process */
PROCESS START COLLECT_PROC;
```

## Notes

In the preceding example, several of the COLLECT\_PROC attributes are 'hard-coded' into the rate schedule. In actual practice, this data could also come directly from records in the Oracle Utilities Data Repository, or through list queries. Also, the above example includes values for several of the optional COLLECT\_PROC attributes. These are included to illustrate how those attributes might be supplied in a rate schedule.

## Process Suspend Statement

### Purpose

The PROCESS SUSPEND Statement is used to suspend an existing running process instance.

### Format

PROCESS SUSPEND statements have this format:

```
PROCESS SUSPEND <stem_identifier>;
```

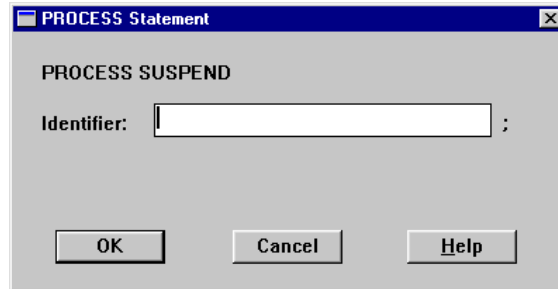
#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Workflow Management Statements**.

### To Create

1. Select **Statements->Workflow->Process Suspend** from the Rules Language Editor menu bar.

The PROCESS SUSPEND Statement template appears.



The screenshot shows a dialog box titled "PROCESS Statement". Inside the dialog, the text "PROCESS SUSPEND" is displayed. Below it, the label "Identifier:" is followed by a text input field and a semicolon. At the bottom of the dialog, there are three buttons: "OK", "Cancel", and "Help".

2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Suspend a process instance of the COLLECT\_PROC process.*

```
/* Set Process Instance Attributes */
COLLECT_PROC.UID = "123";
COLLECT_PROC.NOTE = "Verify customer status";

/* Suspend the process */
PROCESS SUSPEND COLLECT_PROC;
```

## Notes

In the above example, several of the COLLECT\_PROC attributes are 'hard-coded' into the rate schedule. In actual practice, this data could also come directly from records in the Oracle Utilities Data Repository, or through list queries.

## Process Resume Statement

### Purpose

The PROCESS RESUME Statement is used to resume an existing suspended process instance.

### Format

PROCESS RESUME statements have this format:

```
PROCESS RESUME <stem_identifier>;
```

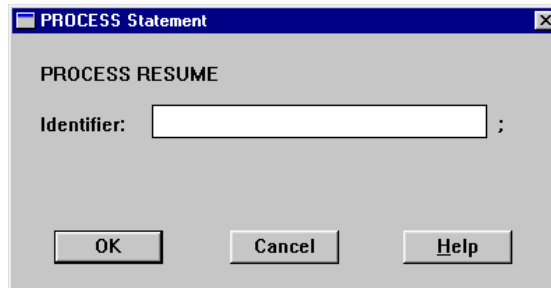
#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Workflow Management Statements**.

### To Create

1. Select **Statements->Workflow->Process Resume** from the Rules Language Editor menu bar.

The PROCESS RESUME Statement template appears.



The screenshot shows a dialog box titled "PROCESS Statement". Inside the dialog, the text "PROCESS RESUME" is displayed. Below it, the label "Identifier:" is followed by a text input field and a semicolon. At the bottom of the dialog, there are three buttons: "OK", "Cancel", and "Help".

2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Resume a suspended process instance of the COLLECT\_PROC process.*

```
/* Set Process Instance Attributes */
COLLECT_PROC.UID = "123";
COLLECT_PROC.NOTE = "Verify customer status";

/* Resume the process */
PROCESS RESUME COLLECT_PROC;
```

## Notes

In the above example, several of the COLLECT\_PROC attributes are 'hard-coded' into the rate schedule. In actual practice, this data could also come directly from records in the Oracle Utilities Data Repository, or through list queries.



## Process Terminate Statement

### Purpose

The PROCESS TERMINATE Statement is used to terminate an existing process instance.

### Format

PROCESS TERMINATE statements have this format:

```
PROCESS TERMINATE <stem_identifier>;
```

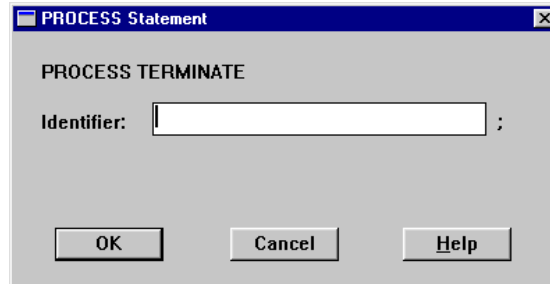
#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Workflow Management Statements**.

### To Create

1. Select **Statements->Workflow->Process Terminate** from the Rules Language Editor menu bar.

The PROCESS TERMINATE Statement template appears.



The screenshot shows a dialog box titled "PROCESS Statement". Inside the dialog, the text "PROCESS TERMINATE" is displayed. Below this, there is a label "Identifier:" followed by a text input field and a semicolon. At the bottom of the dialog, there are three buttons: "OK", "Cancel", and "Help".

2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Terminate a process instance of the COLLECT\_PROC process.*

```
/* Set Process Instance Attributes */
COLLECT_PROC.UID = "123";
COLLECT_PROC.NOTE = "Verify customer status";

/* Terminate the process */
PROCESS TERMINATE COLLECT_PROC;
```

## Notes

In the above example, several of the COLLECT\_PROC attributes are 'hard-coded' into the rate schedule. In actual practice, this data could also come directly from records in the Oracle Utilities Data Repository, or through list queries.

## Process Event Statement

### Purpose

The PROCESS EVENT Statement posts an activity event.

### Format

PROCESS EVENT statements have this format:

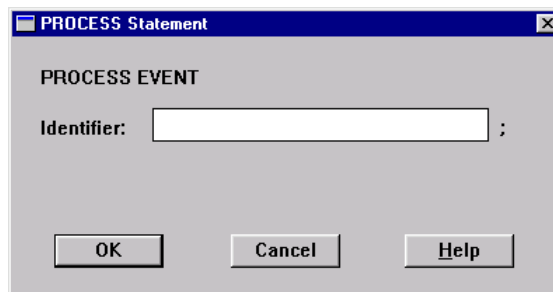
```
PROCESS EVENT <stem_identifier>;
```

#### Where:

- <stem\_identifier> is a stem that contains the appropriate attributes, as described under **Using the Workflow Management Statements**.

### To Create

1. Select **Statements->Workflow->Process Event** from the Rules Language Editor menu bar.  
The PROCESS EVENT Statement template appears.



2. Enter the appropriate stem identifier, or click the *right* mouse button and choose the stem identifier from the **Other Identifiers** list in the **Rules Language Elements Editor**.
3. Click **OK**. The statement appears in the rate form.

## Example

*Post an activity event in the COLLECT\_PROC process.*

```
/* Set Event Attributes */  
COLLECT_PROC.UID = "123";  
COLLECT_PROC.NOTE = "Verify customer status";  
  
/* Post the Event */  
PROCESS EVENT COLLECT_PROC;
```

## Notes

In the above example, several of the COLLECT\_PROC attributes are 'hard-coded' into the rate schedule. In actual practice, this data could also come directly from records in the Oracle Utilities Data Repository, or through list queries.

# Appendix F

---

## XML Rules Language Statements and Functions

This chapter describes XML statements and functions provided by the Oracle Utilities Rules Language, including:

- **XML Overview**
- **XML Statements**
- **XML/Document Object Management Functions**
- **Using the XML Statements and Functions**

## XML Overview

The Oracle Utilities Rules Language provides two mechanisms for processing XML: a declarative approach using the XML\_ELEMENT and XML\_OP statements, and a functional approach using the XML/Document Object Management functions.

The declarative approach allows the user to specify a known XML format and have the underlying Rules Language processor assign values appropriately. This approach essentially "flattens" the nested XML structure so that every element or sub-element is uniquely represented by one Rules Language identifier.

The functional approach gives the user more flexibility in handling an unknown format, but requires detailed knowledge of Document Object Management (DOM).

## XML Data Types

The two new data types introduced with these functions are XML Document and XML Node. The same XML document or node may be assigned to several identifiers; care should be exercised when using these functions, particularly the delete functionality.

The main uses of the XML document format are to load and save data, and to retrieve the root element in the document. The root element is an XML node. You can retrieve the type and value of a node and its siblings and, for nodes that are elements, its attributes and children. If a child node is not actually in the XML document, the node and all its attributes will be cleared.

There are no operations allowed on an XML document. XML documents can only be used as a parameter to one of the XML functions described in this appendix (see **XML/Document Object Management Functions** on page F-12). The only operations allowed on a XML node are comparison (= or <>) to zero. Otherwise, nodes must be used in one of the statements or functions described in this appendix.

---

## Using Stem.Tail XML Identifiers

If an XML node is assigned to an identifier that is a stem, its Stem.Tail identifiers with tails NODENAME, NODETYPE and NODEVALUE are also assigned their corresponding values. The Stem.Tail identifier with tail NODEPRESENT is assigned the integer 1. All other Stem.Tail identifiers are assigned the value of the node's child whose name is the tail.

If a Stem.Tail identifier whose tail is NODEVALUE is assigned a value, and the stem is an XML element with a node, the string representation of the value will be assigned as the node's value.

If an identifier that is an XML element with a node is assigned a value, the string representation of the value will be assigned as the node's value. However, if you want to use the node's value in an expression, you must use Stem.NODEVALUE.

If a Stem.Tail identifier whose tail is not NODEVALUE is assigned a value, and the stem is an XML element with a node, the tail is assumed to be the name of an attribute of the node, and the string representation of the value will be assigned as the value of this attribute.

To remove an attribute from a node, assign it an empty value:

```
// Remove the attribute Tail from the node STEM.  
CLEAR X;  
STEM.Tail = X;
```

## XML Statements

This section provides detailed explanations of the XML statements available in the Oracle Utilities Rules Language. It also describes the formats and conventions used with statements in this manual, and the format in which the statement descriptions are presented.

### Identifier Statement

#### Purpose

The IDENTIFIER Statement is used to define identifiers before they are used.

The order in which identifiers appear in a rate form determines several things, such as the order in which they appear in reports. In general, the earlier an identifier appears in a rate schedule, the earlier it appears in the report. This statement lets you determine ordering without executing any statements (the IDENTIFIER Statement has no run-time component; it only defines identifiers).

This statement can also be used to define a parent identifier before the identifier is used in the **XML\_ELEMENT Statement** on page F-6.

#### Format

IDENTIFIER statements have this format:

```
IDENTIFIER <identifier>, <identifier> ...;
```

#### Where:

- <identifier> is the identifier you wish to define.

#### To Create

The IDENTIFIER statement can only be created from the Rules Language Text Editor. See **The Rules Language Text Editor** on page 2-11 in the *Oracle Utilities Rules Language User's Guide* for more information.

#### Example

*Define the LS\_INPUT identifier.*

```
/* Predefine identifier */  
IDENTIFIER LS_INPUT;
```

#### Notes

When using the IDENTIFIER statement to define XML element identifiers, the IDENTIFIER statement should only be used to define the root element.



---

## OPTIONS Statement

### Purpose

The OPTIONS Statement is used to specify that the case (UPPER or lower) of identifiers should remain as defined. If not present, all identifiers are converted to uppercase. If it is present the name of an identifier remains exactly as typed. With this option, if two identifiers differ only in the case of some of their letters, they are different identifiers.

This statement is useful when defining XML attributes and elements that may need to be either lower-cased or mixed-case.

### Format

OPTIONS statements have this format:

```
IDENTIFIER MIXED_CASE_IDENTIFIERS_UNIQUE;
```

#### Where:

- MIXED\_CASE\_IDENTIFIERS\_UNIQUE indicates that the case of identifiers remain unaltered.

### To Create

The OPTIONS statement can only be created from the Rules Language Text Editor. See **The Rules Language Text Editor** on page 2-11 in the *Oracle Utilities Rules Language User's Guide* for more information.

### Example

*Allow mixed case XML attributes*

```
/* Allow mixed-case XML attributes */  
OPTIONS MIXED_CASE_IDENTIFIERS_UNIQUE;
```

### Notes

Use of the OPTIONS statement affects identifiers that appear after this statement in the rule form. Identifiers that appear before it are uppercased.

All Oracle Utilities defined identifiers such as BILL\_PERIOD, \$EFFECTIVE\_REVENUE, determinant identifiers, and interval data attributes must all be entered in upper case if the OPTIONS statement is used.

## XML\_ELEMENT Statement

### Purpose

The XML\_ELEMENT Statement lets you map an XML format into Rules Language identifiers. The XML format consists of elements and sub-elements, and this statement describes the relationship between a sub-element and its parent. If the parent element is assigned, all its attributes and children are automatically assigned their respective values, recursively. The defined identifier can also be used in the **FOR EACH x IN XML\_ELEMENT\_OF 0 Statement** on page F-8 to iterate over multiple sub-elements with the same name.

### Format

XML\_ELEMENT statements have this format:

```
XML_ELEMENT <identifier> NODENAME <symbol|literal> PARENT
<parent_identifier>;
```

#### Where:

- <identifier> is an identifier used to represent this child element of the parent. An identifier may appear at most once here. Attributes of the element can be represented using the `identifier.attribute` syntax.
- NODENAME is an optional keyword that allows you to define the node name of the element.
- <symbol|literal> is a symbol or literal that exactly matches an element name (case sensitive). There may be several identifiers with the same node name, but different parents.
- PARENT is an optional keyword that allows you to define the parent of the element.
- <parent\_identifier> *Optional*; a previously defined identifier. When it is assigned, this identifier is also set if its element is a child of the parent element. If there is no parent assigned, the identifier is assumed to be the root element of the document.

### To Create

The XML\_ELEMENT Statement can only be created from the Rules Language Text Editor. See **The Rules Language Text Editor** on page 2-11 of the *Oracle Utilities Rules Language User's Guide* for more information.

## Example

*Set the structure of the Oracle Utilities Import format.*

```

/* Set the XML structure of the Oracle Utilities Import XML Format */
/* Set the root element */
IDENTIFIER LS_IMPORT;
/* Declare the child tree */
XML_ELEMENT LS_IMPORT NODENAME "LS_IMPORT";
XML_ELEMENT CUST_DATA NODENAME "CUSTOMER_DATA" PARENT LS_IMPORT;
XML_ELEMENT REC_GROUP NODENAME "RECORD_GROUP_TRANSACTION" PARENT
CUST_DATA;
/* Declare a record */
XML_ELEMENT LS_RECORD NODENAME "LODESTAR_RECORD" PARENT REC_GROUP;
XML_ELEMENT TABLE_ID NODENAME "TABLE" PARENT LS_RECORD;
XML_ELEMENT TABLE_NAME NODENAME "NAME" PARENT TABLE_ID;
/* Declare a column */
XML_ELEMENT LS_COLUMN NODENAME "COLUMN" PARENT TABLE_ID;
XML_ELEMENT COLUMN_NAME NODENAME "COLUMN_NAME" PARENT LS_COLUMN;
XML_ELEMENT COLUMN_VALUE NODENAME "COLUMN_VALUE" PARENT LS_COLUMN;

```

## Notes

If a parent identifier is cleared using the **Clear Statement**, all its children and attribute identifiers are also cleared, recursively. If an XML\_ELEMENT identifier is assigned an XML node, all of its unassigned parent elements will be created as needed, so that its entire parent structure is assigned. The attributes of an element can be assigned any time after the element has been created.

## FOR EACH x IN XML\_ELEMENT\_OF 0 Statement

### Purpose

The FOR EACH x IN XML\_ELEMENT\_OF 0 Statement repeats a set of nested statements for each element defined in an XML structure. This statement iterates the nested statements over all matching elements, one by one. Matching elements have the same element name and the same parent element, as defined in the XML\_ELEMENT Statement.

### Format

FOR EACH x IN XML\_ELEMENT\_OF 0 statements have this format:

```
FOR EACH <xml_element_identifier> IN XML_ELEMENT_OF 0
  <nested_statements>
END FOR;
```

#### Where:

- <xml\_element\_identifier> is an identifier that appears in the IDENTIFIER clause of an XML\_ELEMENT Statement.

### To Create

The FOR EACH x IN XML\_ELEMENT\_OF 0 Statement can only be created from the Rules Language Text Editor. See **The Rules Language Text Editor** on page 2-11 of the *Oracle Utilities Rules Language User's Guide* for more information.

### Example

*Perform a set of operations on each LODESTAR\_RECORD element.*

```
/* Set the XML structure of the Oracle Utilities Import XML Format */
FOR EACH LS_RECORD IN XML_ELEMENT_OF 0;
  <operations>
END FOR
```

### Notes

The 0 is required after XML\_ELEMENT\_OF.

## XML\_OP Statement

### Purpose

The XML\_OP Statement performs an operation on one or more XML elements (as defined using the XML\_ELEMENT Statement). Supported operations include CREATE, INSERT, COPY, and DELETE.

### Format

XML\_OP statements have this format:

```
XML_OP <operation> <identifier> [,<identifier>...];
```

#### Where:

- <operation> is a literal or symbol that is one of the following:
  - **“CREATE” or CREATE:** Creates this node as a child of its parent. Each identifier's node is created, from left to right. If no parent node was specified in the XML\_ELEMENT statement, this node is assumed to be the root element of an XML document. The document is created, with this identifier as its root. If the parent node exists, the sub-element is created and attached to the parent. If a node with this name already exists or the identifier is already assigned a node, a new node is still created. The new node will be a sibling of the previous node if the parent is unchanged; otherwise, it will be a new child of the parent node.
  - **“CREATE\_ALL” or CREATE\_ALL:** Creates this node as a child of its parent, and then creates all its children, recursively. Each identifier's node is created, from left to right. If no parent node was specified in the XML\_ELEMENT statement, this node is assumed to be the root element of an XML document. The document is created, with this identifier as its root. If the parent node exists, the sub-element is created and attached to the parent. If a node with this name already exists or the identifier is already assigned a node, a new node is still created. The new node will be a sibling of the previous node if the parent is unchanged; otherwise, it will be a new child of the parent node.
  - **“INSERT” or INSERT:** Creates this node as a child of its parent, immediately after its previous instance. The node must have a parent. If it has not been created, it is created and appended to the end of the parent's nodes. Each identifier's node is created, from left to right.
  - **“INSERT\_ALL” or INSERT\_ALL:** Creates this node as a child of its parent, immediately after its previous instance, and then creates its children, recursively. The node must have a parent. If it has not been created, it is created and appended to the end of the parent's nodes. Each identifier's node is created, from left to right.
  - **“INSERT\_UNUSED” or INSERT\_UNUSED:** Creates this node as a child of its parent, immediately after its previous instance, if the identifier's node does not have a value or any attributes. The node must have a parent. If it has not been created, it is created and appended to the end of the parent's nodes. If it was created and has a value or attribute, a new node is created immediately after it. If it does not have a value or attribute, the identifier is unchanged. Each identifier's node is created, from left to right.
  - **“INSERT\_UNUSED\_ALL” or INSERT\_UNUSED\_ALL:** Same as INSERT\_UNUSED, except that if a new node is created, all its children are also created, recursively.
  - **“DELETE” or DELETE:** Removes this node as a child of its parent, then deletes it and all its child nodes. Each identifier and all its children are cleared. Identifiers are deleted from left to right. If the node does not have a parent, it is assumed to be the root node. Its document and all related nodes are deleted. **This operation should not be**

used if the node or one of its children has been assigned to more than one identifier.

- <identifier> one or more identifiers that are XML elements (as defined by the **XML\_ELEMENT Statement** on page F-6).

### To Create

The XML\_OP Statement can only be created from the Rules Language Text Editor. See **The Rules Language Text Editor** on page 2-11 of the *Oracle Utilities Rules Language User's Guide* for more information.

## Example

*Create an XML node called LS\_IMPORT and insert a child node called LS\_RECORD as a child of LS\_IMPORT.*

```
/* Set the root element */
IDENTIFIER LS_IMPORT;
/* Declare the child tree */
XML_ELEMENT LS_IMPORT NODENAME "LS_IMPORT";
XML_ELEMENT CUST_DATA NODENAME "CUSTOMER_DATA" PARENT LS_IMPORT;
XML_ELEMENT REC_GROUP NODENAME "RECORD_GROUP_TRANSACTION" PARENT
CUST_DATA;
/* Declare a record */
XML_ELEMENT LS_RECORD NODENAME "LODESTAR_RECORD" PARENT REC_GROUP;

/* Create the LS_IMPORT XML node */
XML_OP CREATE LS_IMPORT;
/* Insert the child LS_RECORD node */
XML_OP INSERT LS_RECORD;
```

## XML/Document Object Management Functions

The functions in this section manipulate an XML string, using Document Object Management (DOM) functions.

Like all functions, you must assign the results of these functions to an identifier using an Assignment Statement. The format is:

```
<identifier> = FUNCTION(<parameters>);
```

**Where:**

- <identifier> is a temporary, determinant, or interval data identifier. The function description below indicates what each returns: a scalar numeric (should be assigned to a temporary identifier), historical values (should be assigned to a determinant identifier), or an interval data reference (should be assigned to an interval data identifier).
- FUNCTION is one of the functions described below.
- <parameters> are one or more expressions, identifiers, or constants, as described in each function listed below.



## DOMDOCCREATE Function

*Creates an XML document with a root element node.*

This function creates an XML document with a specified root element node. Currently, any errors are fatal. Returns an XML document.

**Format:**

```
<identifier> = DOMDOCCREATE (<identifier|string expression>);
```

**Where:**

- <identifier|string expression> is either an identifier or a string expression that evaluates to a string that will be the root element name of the document.

**Example:**

*Create an XML document with a root element name of LS\_IMPORT.*

```
LS_IMP_DOC = DOMDOCCREATE (LS_IMPORT);
```

## DOMDOCLOADFILE Function

*Loads and parses an XML file.*

This function loads and parses an XML file, and returns the XML document contained in the file. Currently, any errors are fatal. Returns an XML document.

**Format:**

```
<identifier> = DOMDOCLOADFILE(<identifier|string expression>);
```

**Where:**

- <identifier|string expression> is either an identifier or a string expression that evaluates to a string that is name of a file containing XML. The default location of the file is the C:\LODESTAR\User directory, but a full path can be specified.

**Example:**

*Load an XML file named LS\_IMPORT.XML.*

```
LS_IMP_FILE = DOMDOCLOADFILE("LS_IMPORT.XML");
```

## DOMDOCLOADXML Function

*Loads and parses an XML document.*

This function loads and parses an XML document. Currently, any errors are fatal. Returns an XML document.

**Format:**

```
<identifier> = DOMDOCLOADXML(<identifier|string expression>);
```

**Where:**

- <identifier|string expression> is either an identifier or a string expression that evaluates to a string that is name of an XML document.

**Example:**

*Load an XML document named LS\_IMPORT.*

```
LS_IMP_DOC = DOMDOCLOADXML("LS_IMPORT");
```

## DOMDOCSAVEFILE Function

*Saves an XML file based on a specified XML document.*

This function creates an XML file based on a specified XML document. The document is written out as XML to the specified file, replacing its contents. Returns the integer 0.

**Format:**

```
<identifier> = DOMDOCSAVEFILE (<xml_document_identifier>,  
<identifier|string expression>);
```

**Where:**

- <xml\_document\_identifier> is an XML document identifier.
- <identifier|string expression> is either an identifier or a string expression that evaluates to a string that is the name of a file that will contain XML. The default location of the file is the C:\LODESTAR\User directory, but a full path can be specified.

**Example:**

*Save an XML document called LS\_IMPORT to a file called LS\_IMPORT.XML.*

```
LS_IMP_SAVE = DOMDOCSAVEFILE (LS_IMPORT, "LS_IMPORT.XML");
```

## DOMDOCGETROOT Function

*Retrieves the root node of an XML document.*

This function retrieves the root node of a specified XML document. Returns an XML node.

**Format:**

```
<identifier> = DOMDOCGETROOT (<xml_document_identifier>);
```

**Where:**

- <xml\_document\_identifier> is an identifier that is an XML document.

**Example:**

*Get the root node of the LS\_DATA XML document.*

```
LS_IMP_ROOT = DOMDOCGETROOT (LS_DATA);
```

## DOMDOCADDPI Function

*Adds a processing instruction to an XML document.*

This function adds a processing instruction to an XML document. Returns an XML node.

**Note:** If using this function inside a FOR EACH in LIST statement, include the USE\_DOMDOCADDAPI\_IN\_LOOP = 1 keyword and value in the LODESTAR.CFG file.

**Format:**

```
<identifier> = DOMDOCADDPI(<identifier|string expression>);
```

**Where:**

- <xml\_node\_identifier> is the root element of the XML document. The 'PI' is inserted before it.
- <node\_name> is a valid XML node name.
- <node\_value> is a literal or string value in the form "attribute=""value"" attribute=""value"" ..." (the double double-quotes will become single double-quotes).

**Example:**

*Add a reference to an XSL style sheet*

```
SS_OUT = DOMDOCADDPI(CUST_BILL, "xml-stylesheet", "type='text/xsl' href='PSNH_BillReport.xsl'");
```

## DOMNODEGETNAME Function

*Retrieves the name of an XML node.*

This function retrieves the name of an XML node. Returns a string.

**Format:**

```
<identifier> = DOMNODEGETNAME (<xml_node_identifier>);
```

**Where:**

- <xml\_node\_identifier> is an identifier that is an XML node.

**Example:**

*Get the name of the LS\_RECORD node.*

```
LS_RECORD_NAME = DOMNODEGETNAME (LS_RECORD);
```

## DOMNODEGETTYPE Function

*Retrieves the type of an XML node.*

This function retrieves the type of an XML node. Types may be "attribute", "element", "comment", "text", .... Returns a string.

**Format:**

```
<identifier> = DOMNODEGETTYPE (<xml_node_identifier>);
```

**Where:**

- <xml\_node\_identifier> is an identifier that is an XML node.

**Example:**

*Get the node type for the LS\_RECORD node.*

```
LS_RECORD_NODE_TYPE = DOMNODEGETTYPE (LS_RECORD);
```



## DOMNODEGETVALUE Function

*Retrieves the value of an XML node.*

This function retrieves the value of an XML node. Returns a string.

**Format:**

```
<identifier> = DOMNODEGETVALUE (<xml_node_identifier>);
```

**Where:**

- <xml\_node\_identifier> is an identifier that is an XML node.

**Example:**

*Get the value of the LS\_RECORD node.*

```
LS_RECORD_VAL = DOMNODEGETVALUE (LS_RECORD) ;
```

## DOMNODEGETCHILDCT Function

*Retrieves the number of child nodes of an XML node.*

This function retrieves the number of child nodes of an XML node (may be 0). Returns an integer.

**Format:**

```
<identifier> = DOMNODEGETCHILDCT(<xml_node_identifier>);
```

**Where:**

- <xml\_node\_identifier> is an identifier that is an XML node.

**Example:**

*Get the number of child nodes in the LS\_RECORD node.*

```
LS_RECORD_NUM_CHILDREN = DOMNODEGETCHILDCT(LS_RECORD);
```

## DOMNODEGETFIRSTCHILD Function

*Retrieves the first child of an XML node, if any.*

This function retrieves the first child of an XML node, if any. If there are no child nodes, returns 0. Returns an XML node.

**Format:**

```
<identifier> = DOMNODEGETFIRSTCHILD(<xml_node_identifier>);
```

**Where:**

- <xml\_node\_identifier> is an identifier that is an XML node.

**Example:**

*Get the first child node of the LS\_RECORD node.*

```
LS_RECORD_FIRSTCHILD = DOMNODEGETFIRSTCHILD(LS_RECORD);
```

## DOMNODEGETSIBLING Function

*Retrieves the next (right side) child of an XML node, if any.*

This function retrieves the next child of an XML node, if any. If there is not another child, returns 0. Returns an XML node.

**Format:**

```
<identifier> = DOMNODEGETSIBLING(<xml_node_identifier>);
```

**Where:**

- <xml\_node\_identifier> is an identifier that is an XML node.

**Example:**

*Get the next child of the LS\_RECORD node.*

```
LS_RECORD_NEXTCHILD = DOMNODEGETSIBLING(LS_RECORD);
```

## DOMNODECREATECHILDELEMENT Function

*Creates a child node in an XML node.*

This function creates a child node in a specified XML node. The new element is appended as the last child node of the specified node. Currently, any errors are fatal. Returns an XML node that is the new element.

### Format:

```
<identifier> = DOMNODECREATECHILDELEMENT (<xml_node_identifier>,  
<identifier|string expression>);
```

### Where:

- <xml\_node\_identifier> is an identifier that is an XML node.
- <identifier|string expression> is either an identifier or a string expression that evaluates to a string that is the name of the new element.

### Example:

*Add a new child node called ARRANGEMENT to the LS\_ACCOUNT node.*

```
LS_RECORD_NEW_NODE = DOMNODECREATECHILDELEMENT (LS_ACCOUNT,  
"ARRANGMENT");
```

## DOMNODESETATTRIBUTE Function

*Sets the value of an attribute of an XML node.*

This function sets the value of an attribute of a specified XML node. The attribute value is added to the element if the attribute does not exist; otherwise, it replaces the attribute's value. Currently, any errors are fatal. Returns the integer 0.

### Format:

```
<identifier> = DOMNODESETATTRIBUTE(<xml_node_identifier>,  
<identifier|string expression>, <identifier|string expression>);
```

### Where:

- <xml\_node\_identifier> is an XML node that is an element.
- <identifier|string expression> is either an identifier or a string expression that evaluates to a string that is the name of the attribute.
- <identifier|string expression> is either an identifier or a string expression that evaluates to a string that is the value of the attribute.

### Example:

*Set the value of the "Arrangement" attribute of the LS\_ACCOUNT node to TRUE.*

```
LS_RECORD_NEW_NODE = DOMNODESETATTRIBUTE(LS_ACCOUNT, "ARRANGEMENT",  
"TRUE");
```

## DOMNODEGETCHILDELEMENTCT Function

*Retrieves the number of child nodes of an XML node that are elements.*

This function retrieves the number of child nodes of an XML node that are elements; this may be 0. Returns an integer.

**Format:**

```
<identifier> = DOMNODEGETCHILDELEMENTCT(<xml_node_identifier>);
```

**Where:**

- <xml\_node\_identifier> is an identifier that is an XML node.

**Example:**

*Get the number of child element nodes in the LS\_RECORD node.*

```
LS_RECORD_NUM_ELEM_CHILDREN = DOMNODEGETCHILDELEMENTCT(LS_RECORD);
```

## DOMNODEGETFIRSTCHILDELEMENT Function

*Retrieves the first child of an XML node that is an element, if any.*

This function retrieves the first child of an XML node that is an element, if any. If there are no child nodes, returns 0. Returns an XML node.

**Format:**

```
<identifier> = DOMNODEGETFIRSTCHILDELEMENT (<xml_node_identifier>);
```

**Where:**

- <xml\_node\_identifier> is an identifier that is an XML node.

**Example:**

*Get the first child element node of the LS\_RECORD node.*

```
LS_RECORD_FIRSTCHILD_ELEM = DOMNODEGETFIRSTCHILDELEMENT (LS_RECORD);
```



## DOMNODEGETSIBLINGELEMENT Function

*Retrieves the next (right side) child of an XML node that is an element, if any.*

This function retrieves the next child of an XML node that is an element, if any. If there is not another child, returns 0. Returns an XML node.

**Format:**

```
<identifier> = DOMNODEGETSIBLINGELEMENT (<xml_node_identifier>);
```

**Where:**

- <xml\_node\_identifier> is an identifier that is an XML node.

**Example:**

*Get the next child element node of the LS\_RECORD node.*

```
LS_RECORD_NEXTCHILD = DOMNODEGETSIBLINGELEMENT (LS_RECORD);
```

## DOMNODEGETATTRIBUTECT Function

*Retrieves the number of attribute nodes of an XML node, if any.*

This function retrieves the number of attribute nodes of an XML node; this may be 0. If the node is not an attribute, returns 0. Returns an integer.

**Format:**

```
<identifier> = DOMNODEGETATTRIBUTECT (<xml_node_identifier>);
```

**Where:**

- <xml\_node\_identifier> is an identifier that is an XML node.

**Example:**

*Get the number of attribute nodes in the LS\_RECORD node.*

```
LS_RECORD_ATT_NODES = DOMNODEGETATTRIBUTECT (LS_RECORD) ;
```

## DOMNODEGETATTRIBUTEI Function

*Retrieves the index'th attribute an XML node, if any.*

This function retrieves the index'th attribute of a specified XML node, if any. If there is no such attribute, returns a NULL node. Returns an XML node.

**Format:**

```
<identifier> = DOMNODEGETATTRIBUTEI (<xml_node_identifier>, <index>);
```

**Where:**

- <xml\_node\_identifier> is an identifier that is an XML node.
- <index> an integer between 1 and the number of attributes in the node, inclusive.

**Example:**

*Get the 4th attribute node in the LS\_RECORD node.*

```
LS_RECORD_ATT1_NODE_4 = DOMNODEGETATTRIBUTEI (LS_RECORD, 4);
```

## DOMNODEGETATTRIBUTEBYNAME Function

*Retrieves the attribute of an XML node with a specified name, if any.*

This function retrieves the attribute of a specified XML node with this name, if any. If there is no such attribute, returns a NULL node. Returns an XML node.

**Format:**

```
<identifier> = DOMNODEGETATTRIBUTEBYNAME (<xml_node_identifier>, <name>);
```

**Where:**

- <xml\_node\_identifier> is an identifier that is an XML node.
- <name> is a string that is the name of an attribute in the XML node.

**Example:**

*Get the attribute in the LS\_ACCOUNT node with the name ARRANGEMENT.*

```
LS_RECORD_ATT_ARRANGE = DOMNODEGETATTRIBUTEBYNAME (LS_RECORD, "ARRANGEMENT");
```

## DOMNODEGETBYNAME Function

*Retrieves the first node under a specified XML node with a specified name, if any.*

This function retrieves the first node under the specified node with the specified name. If there is no such node, returns 0. Returns an XML node.

**Format:**

```
<identifier> = DOMNODEGETBYNAME (<xml_node_identifier>, <name>);
```

**Where:**

- <xml\_node\_identifier> is an identifier that is an XML node.
- <name> a string that is the name of an XML node.

**Example:**

*Get the first node in the LS\_ACCOUNT node with the name ARRANGEMENT.*

```
LS_RECORD_FIRST_ARRANGE = DOMNODEGETBYNAME (LS_RECORD, "ARRANGEMENT");
```

## Using the XML Statements and Functions

The XML statements and functions described in this appendix allow you to obtain data values from XML documents and files and assign those values to identifiers. These identifiers can be used in Rules Language processing, and the results can be saved back to the XML structure for use as output data.

You can also create XML documents and files and populate the nodes within documents and files with appropriate data and values.

This section provides step-by-step descriptions for these operations.

### Reading from XML Documents and Files

The steps for reading data from existing XML documents and files are:

1. Define the XML structure.

Defining the XML structure of the XML document or file defines the relationship between the elements and nodes in the XML document or file. To do this, use the **Identifier Statement** on page F-4 and the **XML\_ELEMENT Statement** on page F-6 respectively.

2. Load the XML file or document.

Loading the XML document or file creates an XML document, and allows the Rules Language to access the root element.

3. Get the root element of the XML document.

Getting the root element of the XML document enables the Rules Language to access the XML elements and nodes. To do this, use the **DOMDOCGETROOT Function** on page F-17. The identifier assigned to the result of the DOMDOCGETROOT function **must** be the root element of the XML document.

4. Derive values from the XML, as specified in the XML structure.

Deriving the data values from the XML document is done using either Stem.Tail identifiers or the DOM functions.

## Example

The following example shows how data can be extracted from an XML structure. In this example, the XML structure is the context of an activity performed using the Oracle Utilities Billing Component - Workflow Management product. With the following activity context:

```
<CONTEXT>
  <ACCOUNTID>123</ACCOUNTID>
  <PASTDUEAMT>90.00</PASTDUEAMT>
  <OTHER />
</CONTEXT>
```

the following Rules Language statements could be used to extract data from the context.

```
/* Define the Context Structure */
IDENTIFIER CONTEXT;
XML_ELEMENT CONTEXT_ID NODENAME "CONTEXT"
XML_ELEMENT ACCOUNT_ID NODENAME "ACCOUNTID" PARENT CONTEXT
XML_ELEMENT PASTDUE_AMT NODENAME "PASTDUEAMT" PARENT CONTEXT
XML_ELEMENT OTHER_ID NODENAME "OTHER" PARENT CONTEXT
/* Load the XML document */
CONTEXT_DOC = DOMDOCLOADXML (RATE_SCHEDULE_INPUT_XML);
/* Obtain the Root Element */
CONTEXT = DOMDOCGETROOT (CONTEXT_DOC)
/* Get the Account ID */
ACCT_ID = ACCOUNT_ID.NODEVALUE;
/* Get the Past Due Amount */
PAST_DUE = PASTDUE_AMT.NODEVALUE;
```

## Creating XML Documents and Files

The steps to create an XML document or file are:

1. Define the XML structure.

Defining the XML structure of the XML document or file defines the relationship between the elements and nodes in the XML document or file. To do this, use the **Identifier Statement** on page F-4 and the **XML\_ELEMENT Statement** on page F-6 respectively.

2. Create the XML document.

Creating an XML document is performed using the CREATE operation of the **XML\_OP Statement** on page F-9. The XML element identifier created via the XML\_OP statement **must** be the root element of the XML document.

3. Create the nodes in the XML document, as specified in the XML structure.

Creating the nodes within the XML document is performed using either the CREATE or INSERT operations of the **XML\_OP Statement** on page F-9. The XML element identifier created via the XML\_OP Statement **must** be the node names of the XML document.

4. Set node values using Stem.Tail identifiers.

Setting the node values in the XML document can be done using either Stem.Tail identifiers or the DOM functions.

5. *Optional.* Save the XML document to a file.

Use the **DOMDOCSAVEFILE Function** on page F-16 to save the XML document to a file.

## Example

The following example shows how an XML structure can be created using the Rules Language. For this example, the XML structure is the context of an activity performed using the Oracle Utilities Billing Component - Workflow Management product.

```
/* Define the Context Structure */
```

```
IDENTIFIER CONTEXT;
XML_ELEMENT CONTEXT_ID NODENAME "CONTEXT"
XML_ELEMENT ACCOUNT_ID NODENAME "ACCOUNTID" PARENT CONTEXT;
XML_ELEMENT PASTDUE_AMT NODENAME "PASTDUEAMT" PARENT CONTEXT;
XML_ELEMENT OTHER_ID NODENAME "OTHER" PARENT CONTEXT;
/* Create the document */
XML_OP CREATE CONTEXT_ID;
/* Create the document nodes and assign values */
XML_OP INSERT ACCOUNT_ID;
ACCOUNT_ID.NODEVALUE = "123";
XML_OP INSERT PASTDUE_AMT;
PASTDUE_AMT.NODEVALUE = "90.00";
XML_OP INSERT OTHER_ID;
OTHER_ID.ARRANGEMENT = "TRUE";
```

The resulting XML structure would be:

```
<CONTEXT>
  <ACCOUNTID>123</ACCOUNTID>
  <PASTDUEAMT>90.00</PASTDUEAMT>
  <OTHER ARRANGEMENT='TRUE' />
</CONTEXT>
```

The following statement could be used to save the XML to a file.

```
CONTEXT_FILE = DOMDOCSAVEFILE (CONTEXT, "CONTEXT.XML");
```



---

---

# Index

## A

- account
  - service types 4-6, 6-2
- account adjustments 2-3
- Account Balance report 10-8
- Account Balance report example 10-9
- Account Balancing 5-26
  - command line program 5-26
- Account Balancing function
  - using 5-27
- account bill information
  - obtaining 6-12
- Account FME Table
  - information stored 3-2, 4-6
- Account Table 3-2, 4-6, 5-4, 5-5, 10-6, 10-10
  - information stored 3-2, 4-6
- ACCOUNT\_BALANCE\_ERROR 10-8
- account-level balancing 5-3, 5-24
- Address Table 10-6
  - information stored 3-3, 4-2
- Adjustments functions 2-3
- Application Method
  - credit application 5-10
- Application Method (credit) 5-4
- application priority of associated receivable type 3-8, 4-8
- application priority of the receivable type 3-7, 4-4
- Application Priority Table 5-2, 5-9, 5-13, 5-15
  - configuration 5-17
  - information stored 3-8, 4-8
- AR Aging report 10-10
- AR Aging report example 10-11
- attributes
  - budget plan D-3
  - service plan D-3
  - tail D-2
  - transaction identifier D-2, E-2
  - user-defined D-6
- Auto Payment Plan Table 7-2, 7-12
  - information stored 3-2, 4-6
- auto payment processing
  - triggering 6-10
- automatic payments 3-2, 4-6
- AUTOPAYMENTS\_PROCESSED message 7-13

## B

- BALACCT.EXE
  - command line program 5-27
- Balance Accounts
  - command line program 5-27
- Balance Accounts command line program
  - BALACCT.EXE 5-27
  - syntax 5-27
- Balance Journal
  - command line program 5-26
- Balance Journal command line program
  - syntax 5-26
- balancing
  - account-level 5-3, 5-24
  - Journal Transaction Table records 5-25
  - journal-level 5-3, 5-24
  - transaction-level 5-3, 5-24
- balancing controls functionality 5-24
  - levels 5-3
- Batch AutoPayments command line program 7-14
  - syntax 7-14
- batch files
  - Report Generator 10-18
- Batch Payment data structure 7-13
- Batch Payment Table 10-4
  - information stored 3-7
- batch payments 5-6, 7-5
- BATCHPAYMENT\_ERROR message 7-8
- BATCHPAYMENT\_PROCESSED message 7-9
- BAUTOPAY.EXE 7-14
- BAUTOPAY.EXE parameters 7-14
- bill calculation
  - data generated 6-3
- Bill Correction module
  - CANCEL mode 6-13
  - CANCEL/REBILL mode 6-13
- Bill Cycle Table 5-5
- Bill History Table 5-5
- Billing function processing 6-3
- Billing functions 6-1, 6-5
- Billing module 6-12, 6-13
  - description 2-3
  - functions 6-3
- Billing module functionality
  - overview 6-1
- Billing module functions

- Get Bill Info 6-12
  - triggering 6-3
- Budget Plan 6-4
- budget plan attribute D-3
- Budget Plan Table 6-2, 10-14
  - information stored 3-3, 4-6, 6-2
- Budget tables 6-2
  - Budget Plan 3-3
- Budget Type Table 6-2
  - information stored 3-3, 4-2, 6-2
- budget types 4-2, 6-2

## C

- CALCULATE\_LATEPAYMENT Function D-33
- Cancel AutoPayment function 7-13
- cancel charge transaction 5-16
  - defined 5-16
  - processing 5-16
- cancel credit transaction
  - defined 5-17
  - processing 5-17
- Cancel functions 6-13
  - triggering 8-15
- Cancel Reason Table
  - information stored 3-7, 4-2
- Cancel Transaction function 5-11
  - triggering 5-12, 6-5, 6-6, 6-7, 6-8, 6-9, 6-10, 6-11, 7-12, 8-3, 8-4, 8-5, 8-14
- Cancel Transaction Statement D-31
- Cancel Transfer Amount function 5-12
  - triggering 5-12
- canceling transaction process 6-13
- canceling transactions 6-13
- cancelled credit transaction 5-17
- charge transaction
  - cancelling 5-16
  - paid 5-17
- Charge Type Table 5-5
  - information stored 3-7, 4-2
- charge types 3-7, 4-2
- COLLECTIONS\_PAYMENT message 7-8
- command line program
  - Account Balancing 5-26
  - Balance Accounts 5-27
  - Balance Journal 5-26
    - for journal balancing 5-25
  - Sub-Ledger Roll-Up 5-21
- command line programs
  - Batch AutoPayments 7-14
- Configuration File
  - EnhancedLogging parameter 10-19
  - LogFile parameter 10-19
  - RptDir parameter 10-19
- Configuration File (LSReports.ini) 10-18
- control activity message 5-10, 5-20
  - JOURNAL\_BALANCED 5-25
- Control Message Data Table 10-8, 10-12
- Control Message record 10-8, 10-12
- Control Message Table 10-8, 10-12
- Control Messages queue C-5
- CONTROLMSG C-5

- controls
  - balancing 5-24
  - transaction-level 5-11
  - transaction-level balancing 5-10
- cost center ID 5-4
- Cost Center Table 5-4, 10-14
  - information stored 3-9, 4-3
- Cost Center Translation Table 5-2, 5-18
  - information stored 3-9, 4-8
- creating journal entries 5-4
- Credit Application function
  - triggering 5-13
- credit application functionality 5-2, 5-13
- credit application method 5-4, 5-13
- credit application methods
  - deferred 5-13
  - immediate 5-13
  - invoice ID 5-13
  - specified 5-13
- Credit Application No 5-14, 5-16, 5-17
- credit application processing
  - for BILL transaction 5-13, 5-15
  - for cancel charge transaction 5-16
  - for credit transaction 5-13, 5-15
- Credit Application record
  - corresponding to a journal entry 5-18
  - Directed flag 5-15
- credit application record 5-9
- Credit Application records
  - journaling 5-19
- credit application records 5-13
- Credit Application Table 5-3, 5-9, 5-13, 5-14, 5-15, 5-16, 5-17, 5-25, 10-12
  - Directed flag 5-15
  - information stored 3-8
  - journal records 5-17
- Credit Application tables 3-5
  - Application Priority Table 3-8, 4-8
  - Credit Application Table 3-8
- credit journal account ID 5-4
- credit transaction
  - cancelled 5-17
- custom reports
  - creating 10-21
  - integrating into FME 10-21

## D

- data
  - reading from XML F-34
- data generated by bill calculation 6-3
- Data Manager List/Query functionality 6-12
- data stored in the Transaction Table 3-6
- data structures
  - Batch Payment 7-13
  - Payment File 7-13
- data that comprises a batch payment 5-6
- data that comprises a payment file 5-7
- data that comprises a transaction in the Financial Engine 5-4
- Database schema diagram A-1, B-1
- database tables
  - Billing 6-2

- database tables used by the Remittance module 7-1
- debit journal account ID 5-4
- default message queue type C-3
- default message queues C-5, C-6
- default message types C-4
- DEFERRED A/R 10-14
- Deferred credit application 5-13
- deprecated statements D-36
- designing reports 10-21
- Directed flag 5-15
- Document Object Management (DOM) F-2
  - functions F-12
- DOM (Document Object Management) F-2
  - functions F-12
- DOMDOCADDPI function F-18
- DOMDOCCREATE function F-13
- DOMDOCGETROOT function F-17
- DOMDOCLOADFILE function F-14
- DOMDOCLOADXML function F-15
- DOMDOCSAVEFILE function F-16
- DOMNODECREATECHILDELEMENT function F-25
- DOMNODEGETATTRIBUTEBYNAME function F-32
- DOMNODEGETATTRIBUTECT function F-30
- DOMNODEGETATTRIBUTEI function F-31
- DOMNODEGETCHILDCT function F-22
- DOMNODEGETCHILDELEMENTCT function F-27
- DOMNODEGETFIRSTCHILD function F-23
- DOMNODEGETFIRSTCHILDELEMENT function F-28
- DOMNODEGETSIBLING function F-24
- DOMNODEGETSIBLINGELEMENT function F-29
- DOMNODEGETVALUE function F-21
- DOMNODESETATTRIBUTE function F-26

**E**

- EnhancedLogging parameter 10-19

**F**

- Financial Engine 2-2, 3-1
  - credit application functionality 5-17
  - definition 2-3
  - functions 5-1, 5-2
  - transactions functionality 5-9
- Financial Management Extension
  - Back Office Application 2-4
  - interfaces 2-2
  - modules 2-3
- Financial Management Extension to BillingExpert
  - Overview 2-1
- financial transaction data
  - obtaining 6-12
- FME Back Office C-8
- FME reporting functionality 10-1
- FME Reports 10-3
- FME reports 10-3
  - running 10-18
  - viewing 10-20
- FME statements
  - in Oracle Utilities Rules Language D-1
- FMGETBILLINFO function 6-12, D-34
- FOR EACH x IN XML\_ELEMENT\_OF 0 statement F-8
- Forecast Management Overview 1-1

- functions
  - Billing 6-3, 6-5
  - CALCULATE\_LATEPAYMENT D-33
  - Cancel AutoPayment 7-13
  - Cancel Transaction 5-11, 5-12, 7-12, 8-3, 8-4, 8-5, 8-14
  - Cancel Transfer Amount 5-12
  - Credit Application 5-13
  - DOMDOCADDPI F-18
  - DOMDOCCREATE F-13
  - DOMDOCGETROOT F-17
  - DOMDOCLOADFILE F-14
  - DOMDOCLOADXML F-15
  - DOMDOCSAVEFILE F-16
  - DOMNODECREATECHILDELEMENT F-25
  - DOMNODEGETATTRIBUTEBYNAME F-32
  - DOMNODEGETATTRIBUTECT F-30
  - DOMNODEGETATTRIBUTEI F-31
  - DOMNODEGETCHILDCT F-22
  - DOMNODEGETCHILDELEMENTCT F-27
  - DOMNODEGETFIRSTCHILD F-23
  - DOMNODEGETFIRSTCHILDELEMENT F-28
  - DOMNODEGETSIBLING F-24
  - DOMNODEGETSIBLINGELEMENT F-29
  - DOMNODEGETVALUE F-21
  - DOMNODESETATTRIBUTE F-26
  - FMGETBILLINFO 6-12, D-34
  - FPROCESSAUTOPAYMENT D-35
  - Get Bill Info 6-12
  - Hold, Release, Close, and Reopen Message C-7
  - Issue Refund 10-6
  - Journal Balancing 5-26
  - List Message C-7
  - Maintenance module 8-2
  - Messaging C-7, C-8
  - Peek Message C-7
  - Post C-8
  - Post Bill 6-10
  - Post Budget Bill Trueup 6-3
  - Post Message C-7, C-8
  - Post Payment 7-8
  - Post Penalty 7-12
  - Post Statement 6-3
  - Post Transaction 5-9, 5-10, 6-5, 6-6, 6-7, 6-8, 6-9, 6-10, 6-11, 7-7, 7-8, 8-4, 8-5
  - Process Auto Payment 6-11
  - Process Batch Payment 7-9
  - Remittance 7-1, 7-7
  - Remittance module 7-4
  - Remove Message C-7
  - Sub-Ledger Roll-Up 5-21
  - transaction balancing 5-24
  - Transfer Transaction Amount 5-11, 8-3
  - triggered by the Rate Language 6-3
  - triggered through an interface 6-3
  - triggering 2-2
  - XML DOM F-12
- functions designed to support work queues C-7

**G**

- G/L Activity report 10-12
- G/L Activity report example 10-13

Get Bill Info Billing module function 6-12  
Get Bill Info function  
    triggering 6-12

## H

held messages C-7  
Hold, Release, Close, and Reopen Message functions C-7

## I

identifier statement F-4  
identifiers  
    stem.column\_name D-6  
    stem.tail F-3  
immediate credit application 5-13  
Input Messages queue C-5  
INPUTMSG C-5  
interfaces  
    Financial Management Extension 2-2  
invoice ID credit application 5-13  
Issue Refund function 10-6  
Issue Refund report 10-6  
Issue Refund report example 10-7

## J

Journal Account record 10-14  
Journal Account Table 10-14  
    information stored 3-9, 4-3  
journal balancing 5-25  
    arguments 5-26  
    parameters 5-25  
journal balancing command line program 5-25  
Journal Balancing function 5-26  
journal records 5-9, 5-14, 5-16, 5-17  
Journal Transaction records 5-25  
Journal Transaction Table 5-2, 5-3, 5-9, 10-12  
    information stored 3-10  
    journal entries 5-18  
Journal Transaction Table records 5-20  
    balancing 5-25  
Journal Transaction tables 3-9  
    Cost Center Table 3-9  
    Journal Account 3-9, 4-3  
    Journal Transaction 3-10  
    Journal Translation 3-9, 4-8  
Journal Translation record 5-18, 5-19  
Journal Translation rules 5-19  
Journal Translation Table 5-2, 5-9, 5-18  
    configuration 5-14, 5-15, 5-16, 5-17  
    information stored 3-9, 4-8  
Journal Translation tables  
    Cost Center Translation 3-9, 4-8  
JOURNAL\_BALANCE\_ERROR message 5-25  
JOURNAL\_BALANCED 10-12  
JOURNAL\_BALANCED control activity message 5-25  
journaling Credit Application records 5-19  
journaling functionality 5-2, 5-18  
journaling process  
    steps 5-18  
Journaling Transaction records 5-19  
journal-level balancing 5-3, 5-24

Jurisdiction Table 5-5

## L

List Message function C-7  
LISTUPDATE Rate Language function 6-12  
Location tables  
    Address 3-3  
    Premise 3-3  
    Region 3-3  
LogFile parameter 10-19  
lookup tables  
    Write-Off Reason 3-3, 4-5  
LSDB message queue type C-3  
LSDB queue type C-6  
LSREPORTS.EXE 10-18  
LSReports.ini file  
    example 10-18  
LSTransaction Table  
    information stored 3-6

## M

Maintenance functions 8-1  
    triggered through the FME Back Office 8-2  
Maintenance module  
    description 2-3  
Maintenance module functionality 8-1  
Maintenance module functions 8-2  
    triggering 8-3  
message  
    control activity 5-10  
Message Data Table C-6  
    information stored C-6  
Message Queue Table  
    information stored C-3  
message queues  
    corresponding tables C-6  
    default C-5  
    supported C-5  
Message Table C-6  
    information stored C-6  
Message Type Queue Table  
    information stored C-3  
Message Type Table  
    information stored C-3  
message types  
    default C-4  
    used by the Messaging System C-4  
message, control activity 5-25  
MESSAGEQUEUE Table C-3  
messages  
    AUTOPAYMENTS\_PROCESSED 7-13  
    BATCHPAYMENT\_ERROR 7-8  
    BATCHPAYMENT\_PROCESSED 7-9  
    COLLECTIONS\_PAYMENT 7-8  
    control activity 5-20  
    JOURNAL\_BALANCE\_ERROR 5-25  
    PAYMENT\_EXCEPTION\_ERROR 7-12  
    PAYMENTFILE\_ERROR 7-9  
    PAYMENTFILE\_PROCESSED 7-10  
    PAYMENTFILEIN 7-13  
    PAYMENTFILEOUT 7-13

- UNPOSTABLE\_PAYMENT 7-8
- MESSAGE TYPE Table C-3
- MESSAGE TYPE QUEUE Table C-3
- Messaging functions C-2, C-7, C-8
  - access by external systems C-8
- Messaging Post function C-8
- Messaging System
  - example C-2
  - message types C-4
  - tables C-3
- Messaging system
  - FME C-1
- Messaging System functions C-7
- Meter History Table
  - information stored 3-4, 4-7
- Meter Read Table
  - information stored 3-4, 4-7
- Meter Table
  - information stored 3-4, 4-3
- Meter tables
  - Meter History Table 3-4, 4-7
  - Meter Read Table 3-4
  - Meter Table 3-4
- modules
  - Financial Management Extension 2-3

## O

- Operating Company Table 5-5
- OPTIONS statement F-5
- Oracle Utilities Data Repository 3-1
- other account Bill Information
  - obtaining 6-12
- Output Messages queue C-5
- OUTPUTMSG C-5
- outstanding charge transaction 5-13, 5-16
- outstanding charge transactions 5-13, 5-15
- Overview 2-1
- overview
  - Billing module functionality 6-1
  - database tables 3-1
  - Messaging system C-1

## P

- paid charge transaction 5-17
- parameters
  - journal balancing 5-25
- payment file 7-6
- Payment File data structure 7-13
- Payment Method Table
  - information stored 4-4, 7-2
- Payment Posting report 10-4
- Payment Posting report example 10-5
- Payment Source Table
  - information stored 4-4, 7-2
- Payment Table
  - information stored 7-2
- PAYMENT\_EXCEPTION\_ERROR message 7-12
- PAYMENTFILE\_ERROR message 7-9
- PAYMENTFILE\_PROCESSED message 7-10
- PAYMENTFILEIN message 7-13
- PAYMENTFILEOUT message 7-13

- Peek Message function C-7
- Post Adjustment statement D-19
- Post Bill 6-10
- Post Bill function 6-10
- POST BILL Statement 6-11, 8-5
- Post Bill Statement D-15
- Post Budget Bill Charge statement D-42
  - BUDGETPLAN attribute D-3
- Post Budget Bill Trueup function
  - Charge or Credit 6-3
- Post Budget Bill Trueup statement D-44
  - BUDGETPLAN attribute D-3
- Post Budget Service Charge statement D-40
  - BUDGETPLAN attribute D-3
  - SERVICEPLAN attribute D-3
- Post Charge Or Credit statement D-7
- Post Deferred Service Charge statement D-38
- Post Deposit Application Statement D-29
- Post Deposit statement D-25
- Post function
  - triggering C-8
- Post Installment Charge statement D-46
- Post Message function C-7
  - triggering C-8
- Post Payment function 7-8
- Post Payment statement D-17
- Post Penalty function
  - triggering 7-12
- Post Refund statement D-21
- Post Service Charge statement D-36
  - SERVICEPLAN attribute D-3
- Post Statement function 6-10
  - Statement Date 6-3
- POST STATEMENT Statement 6-10
- Post Statement statement D-13
  - STATEMENTDATE attribute D-3
- Post Tax statement D-9
- Post Transaction function 5-9, 7-8
  - triggering 5-10, 6-5, 6-7, 6-8, 6-9, 6-10, 7-7, 7-8, 8-4, 8-5, 8-14
- Post Transaction process
  - high-level illustration 5-9
- Post Writeoff Statement D-23
- Premise Table
  - information stored 3-3, 4-4
- Process AutoPayment function
  - triggering 6-11
- Process Batch Payment function
  - triggering 7-9
- Process Event statement E-11
- Process Resume statement E-7
- Process Start statement E-3
- Process Suspend statement E-5
- Process Terminate statement E-9
- PROCESSAUTOPAYMENT function D-35

## Q

- query
  - accessed from a billing rate schedule 6-12
  - Account Balancing 5-27
  - configuration 5-27

## R

- Rate Code Table 5-5
- Rate Language 6-3, 6-5, 6-6, 6-7, 6-8, 6-9, 6-10, 6-11, 6-12, 7-8, 8-4, 8-5, 8-14
- Rate Language functions
  - FMGETBILLINFO 6-12
  - LISTUPDATE 6-12
- Receivable Type Table 3-7, 4-4, 5-5
  - information stored 3-7, 4-4
- receivable types 3-7, 4-4
- Receivables Component
  - Rules Language statements D-2
- records
  - Control Message 10-8, 10-12
  - Journal Account 10-14
  - Work Queue Message 10-8
- Refunds functions 2-3
- Remittance database tables 7-2
- Remittance functions 7-1, 7-4
- Remittance module 6-11
  - database tables 7-1
- Remittance module functions
  - triggering 7-7
- Remittance modules
  - description 2-3
- Remove Message function C-7
- report format
  - FME reports 10-3
- Report Generator
  - Configuration File 10-18
- Report Generator batch file 10-18
- Report List Table
  - information stored 10-2
- report samples
  - Account Balance report 10-9
  - AR Aging report 10-11
  - G/L Activity 10-13
  - Issue Refund report 10-7
  - Payment Posting report 10-5
- Report Table 10-19
  - information stored 10-2
- Reporting functionality of FME 10-1
- reports 2-4
  - Account Balance 10-8
  - AR Aging 10-10
  - designing 10-21
  - FME 10-3
  - G/L Activity 10-12
  - Issue Refund 10-6
  - Payment Posting 10-4
  - System Balance 10-14
  - viewing 10-20
- Revenue Code Table 10-10
- RptDir parameter 10-19
- rules
  - Journal Translation 5-19
- Rules Language
  - Financial Management statements D-2
  - FME statements available in D-1
  - Workflow Management statements E-2
  - workflow management statements available in E-1
  - XML processing F-2

XML statements F-4

## S

- sample reports
  - System Balance 10-15, 10-17
- Service Plan 6-4
- service plan attribute D-3
- Service Plan Table
  - information stored 3-3, 4-6, 6-2
- Service tables
  - Service Plan 3-3
  - Service Type 3-3
- Service Type Table
  - information stored 3-3, 4-4, 6-2
- special message functionality required for work queues C-7
- specified credit application 5-13
- Statement Date 6-3
- statementing interface 6-12
- statements
  - Cancel Transaction D-31
  - FOR EACH x IN XML\_ELEMENT\_OF 0 F-8
  - identifier F-4
  - OPTIONS F-5
  - Post Adjustment D-19
  - Post Bill D-15
  - Post Budget Bill Charge D-3, D-42
  - Post Budget Bill Trueup D-3, D-44
  - Post Budget Service Charge D-3, D-40
  - Post Charge Or Credit D-7
  - Post Deferred Service Charge D-38
  - Post Deposit D-25
  - Post Deposit Application D-29
  - Post Installment Charge D-46
  - Post Payment D-17
  - Post Refund D-21
  - Post Service Charge D-36
  - Post Statement D-13
  - Post Tax D-9
  - Post Writeoff D-23
  - Process Event E-11
  - Process Resume E-7
  - Process Start E-3
  - Process Suspend E-5
  - Process Terminate E-9
  - Receivables Component D-2
  - XML\_ELEMENT F-6
  - XML\_OP F-9
- stem.column\_name identifiers D-6
- stem.tail identifiers F-3
- Subledger Account Table 10-14
  - information stored 3-10
- Subledger Account tables 3-9
- sub-ledger roll-up
  - steps 5-20
- Sub-Ledger Roll-Up command line program
  - syntax 5-21
- Sub-Ledger Roll-Up function 5-21
- sub-ledger roll-up functionality 5-2, 5-20
- supported message queue types C-3
- syntax
  - Balance Accounts command line program 5-27

Balance Journal command line program 5-26  
Batch AutoPayments command line program 7-14  
System Balance report 10-14  
System Balance report example 10-15, 10-17

## T

### tables

Account 3-2, 5-4, 5-5, 10-6, 10-10  
Account FME 3-2  
Address 3-3, 10-6  
Application Priority 3-8, 4-8, 5-2, 5-9, 5-13, 5-15, 5-17  
Auto Payment Plan 3-2, 4-6, 7-2, 7-12  
Batch Payment 3-7, 7-2, 10-4  
Bill Cycle 5-5  
Bill History 5-5  
Budget 6-2  
Budget Plan 3-3, 4-2, 4-6, 6-2, 10-14  
Budget Type 3-3, 4-2, 6-2  
Cancel Reason 3-7, 4-2  
Charge Type 3-7, 4-2, 5-5  
Collections 9-2  
Control Message 10-8, 10-12  
Control Message Data 10-8, 10-12  
Cost Center 3-9, 4-3, 5-4, 10-14  
Cost Center Translation 3-9, 4-8, 5-2  
Cost Center Translation Table 5-18  
Credit Application 3-5, 3-8, 5-3, 5-9, 5-13, 5-14, 5-15, 5-16, 5-17, 5-25, 10-12  
for FME Reporting functionality 10-2  
Journal Account 3-9, 4-3, 10-14  
Journal Transaction 3-10, 5-2, 5-3, 5-9, 5-18, 5-20, 5-25, 10-12  
Journal Translation 3-9, 4-8, 5-2, 5-9, 5-14, 5-15, 5-16, 5-17, 5-18  
Jurisdiction 5-5  
Message C-6  
Message Data C-6  
Message Queue C-3  
Message Type C-3  
Message Type Queue C-3  
Messaging C-3  
Meter History Table 3-4, 4-7  
Meter Read Table 3-4, 4-7  
Meter Table 3-4, 4-3  
Operating Company 5-5  
Payment 7-2  
Payment Source 4-4, 7-2  
Premise 3-3, 4-4  
Rate Code 5-5  
Receivable Type 3-7, 4-4, 5-5  
Remittance 7-2  
Report 10-19  
Report List 10-2  
Report Table 10-2  
Revenue Code 10-10  
Service Plan 3-3, 4-6, 6-2  
Service Type 3-3, 4-4, 6-2  
Subledger Account 3-10, 10-14  
Subledger Account tables 3-9  
Transaction 3-5, 3-6, 5-3, 5-9, 5-10, 5-24, 5-25, 10-6, 10-10, 10-12, 10-14, 10-16  
Transaction ID 3-5, 4-4, 5-4  
Transaction Table 5-4  
Transaction Type 3-5, 4-4, 5-4  
Work Queue Message 10-8  
Work Queue Message Data 10-8  
tail attributes D-2  
transaction balancing function 5-24  
transaction canceling process 5-11  
Transaction data tables 3-5  
Transaction ID Table 5-4  
uses 3-5, 4-4  
transaction identifier  
attributes D-2  
transaction identifier attribute E-2  
transaction posting process 5-10  
Transaction record  
corresponding to a journal entry 5-18  
Transaction Table 5-3, 5-9, 5-10, 5-24, 5-25, 10-6, 10-10, 10-12, 10-14, 10-16, D-6  
transaction processing 5-4  
ZONE column D-43  
Transaction tables 3-5  
Cancel Reason Table 3-7, 4-2  
Charge Type Table 3-7, 4-2  
LSTransaction Table 3-6  
Transaction ID Table 3-5, 4-4  
Transaction Type Table 3-5, 4-4  
Transaction Type Table 5-4  
financial transactions recognized 3-5, 4-4  
transaction types  
Payment 5-19  
transaction-level balancing 5-3, 5-10, 5-24  
transaction-level controls 5-11  
transactions  
canceling 6-13  
transactions functionality  
Financial Engine 5-2  
Transactions menu  
enabling 11-2  
transfer process 5-11  
Transfer Transaction Amount function 5-11  
triggering 5-11, 8-3  
Transfers functions 2-3  
Transfers module 2-3  
Translation tables 5-18  
Cost Center Translation 5-2  
Journal Translation 5-2  
triggering  
Remittance functions 7-7  
triggering a credit application 5-4  
triggering credit transaction 5-14, 5-16  
triggering functions 2-2  
triggering Maintenance module functions 8-3  
triggering transaction  
credit 5-13, 5-15  
non-deferred charge 5-14, 5-16  
non-deferred credit 5-14, 5-16  
types of message queues supported C-3  
types of payment data 7-4

## U

unapplied credit transaction 5-13, 5-16

unapplied credit transactions 5-13, 5-15  
UNPOSTABLE\_PAYMENT message 7-8  
user-defined attributes, accessing in Rules Language D-6

## **V**

viewing FME reports 10-20

## **W**

Work Queue Message Data Table 10-8  
Work Queue Message record 10-8  
Work Queue Message Table 10-8  
Work Queue Messages queue C-6  
Workflow Management  
    Rules Language statements E-2  
WORKQUEUEMSG C-6  
Write-Off Reason Table 3-3, 4-5  
Write-Offs module 2-3

## **X**

XML  
    documents and files  
        creating F-35  
        reading from F-34  
    DOM functions F-12  
    Rules Language processing of F-2  
    using statements and functions F-34  
XML\_ELEMENT statement F-2, F-6  
XML\_OP statement F-2, F-9