Oracle Utilities Energy Information Platform

Rules Language User's Guide

Release 1.6.1.23 for Windows

E18202-24

December 2018

(Revised July 2019)

ORACLE

Oracle Utilities Rules Language/Rules Language User's Guide, Volume 1, Release 1.6.1.23 for Windows

E18202-24

Primary Author: Lou Prosperi

Contributor: Steve Pratt

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: xercesImpl.jar, xalan.jar

Use of xercesImpl and xalan 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: Paul Johnston

Modules: md5.js

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: Tom Wu

Module: jsbn library

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

Contents

What's New

New Features in the Oracle Utilities Rules Language User's Guide 1-i
New Features for Release 1.6.0.0 ..... 1-i

Chapter 1

Introducing the Oracle Utilities Rules Language 1-1
What is the Oracle Utilities Rules Language? 1-2
What Data is Used to Perform Calculations? 1-3
Rate Forms 1-4
Rate Form Building Blocks 1-4
Rate Form Types 1-5
Rate Form Versions 1-5
Sample Rate Form 1-6
Writing and Editing Rate Forms 1-6
Running Rate Forms 1-6
Factors and Overrides 1-7
Factors 1-7
Overrides 1-7
Cancel/Rebill Rider 1-8

Chapter 2

Using the Rules Language Editor 2-1
The Rules Language Editor 2-2
Adding, Modifying, and Deleting Statements 2-4
Using the Rules Language Elements Editor 2-6
Rules Language Element Types 2-6
Working with the Rules Language Elements Editor 2-9
Saving a Completed Rate Form 2-10
Printing a Rate Form 2-10
Other Tools for Writing Rate Forms 2-11
The Rules Language Text Editor 2-11
The Rate Wizard 2-13

Chapter 3

How Rate Forms are Processed 3-1
Overall Execution Process 3-2
Which Rate Form? 3-3
Shared Symbol Table 3-3
Saving Data from a Rate Schedule 3-4
Types of Data 3-4
Related Bill Pages 3-4
Requirements 3-5
Transactions 3-5
Postponed Saves 3-5

Two Phase Commit 3-5

Chapter 4

Identifiers, Constants, and Expressions 4-1

Identifiers 4-2
Revenue Identifiers 4-4
Bill Determinant Identifiers 4-5
Database Identifiers 4-5
Interval Data Handles 4-6
Time-of-Use Handles 4-6
Factor Identifiers 4-7
Override Identifiers 4-8
Bill History Predefined Identifiers 4-9
Other Predefined Identifiers 4-10
Assignable Predefined Identifiers 4-11
Reserved Identifiers 4-13
Record Identifiers (stem.component) 4-14
Rate Schedule Environment Identifiers 4-16
Report Options Identifiers 4-19
Array Identifiers 4-20
Constants 4-23
Expressions 4-24
String Expressions 4-24
Date Expressions 4-25
Arithmetic Expressions 4-26

Chapter 5

Statements Overview 5-1

General Statements 5-2
Control Statements 5-2
Revenue Computation Statements 5-4
Report Statements 5-4
Miscellaneous Statements 5-5
Financial Management Statements 5-6
WorkFlow Manager Statements 5-7
XML Statements 5-7

Chapter 6

Rules Language Functions Overview 6-1

Interval Data Functions 6-2
Enhanced Interval Data Functions 6-6
Meter Value Functions 6-8
Math Functions 6-9
String Functions 6-11
Other Functions 6-12
Database Functions 6-12
Date/Time Functions 6-14
Historical Data Functions 6-16
Internal Functions 6-17
Season-Based Functions 6-18
Oracle Utilities Receivables Component Functions 6-18
XML/Document Object Management Functions 6-18
Term Functions 6-20
Miscellaneous Functions 6-21
Rules for Using Functions 6-23
Functions and Identifiers 6-23

About Identifiers 6-23
About Parameters 6-24

Chapter 7

Working with Interval Data 7-1
Interval Data Functions Overview 7-2
Interval Data Function Errors 7-2
Types of Interval Data Handles 7-3
Interval Data Function Parameters 7-3
Combining and Comparing Interval Data Handles 7-6
Timezones and DST 7-7
Timezone Support 7-7
DST Support in the US 7-11
Unit-of-Measure Rates and Quantities 7-12
UOM Categories 7-12
UOM Mapping 7-12
Mapping Data 7-13
Rules Language Functions 7-14
Loading Interval Data 7-15
INTDLOAD Functions 7-15
INTDLOADEX Functions 7-19
Loading Overlapping Cuts 7-20
Loading Partial Intervals 7-21
Notes On Loading Interval Data 7-23
Creating Interval Data Masks 7-24
Overview 7-24
Interval Data Mask Functions 7-24
Interval Data Mask Operator Rules 7-25
Other Interval Data Operations 7-26
Interval Data Functions 7-26
Scalar and Block Operations 7-29
Working with Enhanced/Generic Interval Data 7-30
Deriving Billing Determinants and Values from Interval Data 7-36
Overview 7-36
Bill Determinants 7-36
Other Values 7-37
Examples of Working with Interval Data 7-38
Loading Interval Data 7-38
Time-of-Use Periods 7-38
Override Masks 7-39
Time-of-Use and Overrides 7-39
Calculating Coincident and Non-Coincident Peaks 7-40

Chapter 8

Working with COM Components 8-1
Overview 8-2
Working with COM Objects 8-3
Creating COM Objects 8-3
COM Expressions 8-3
COM Object Functions 8-5
COM Error Handling 8-6
VARIANT Data Type 8-7
Rules Language and VARIANT Types 8-7
Examples 8-10

Appendix A

Setting Up Rate Form Records and Rate Codes A-1
Adding a Rate Form Record A-2
Adding Rate Codes A-4
Creating a New Rate Form Version A-5
Importing and Exporting Rate Forms in Batch Mode A-6

Index

What's New

New Features in the Oracle Utilities Rules Language User's Guide

This chapter outlines the new features of the 1.6.0.0 release of the Oracle Utilities Rules Language that are documented in this guide.

New Features for Release 1.6.0.0

Feature	Description	For more information, refer to...
Term-Based Rules Language Functions	This release includes new Rules Language functions to retrieve and save terms and term details to and from the Oracle Utilities Data Repository.	**Term Functions** on page 13-74 in the *Oracle Utilities Rules Language Reference Guid*e See **Term Functions** on page 6-20 for an overview of these functions.
Query Lists	Query lists are structured query language (SQL) queries that can be used by Oracle Utilities Rules Language to access records stored in the Oracle Utilities Data Repository. Query lists are created using the **Lists** function available through the Energy Information Platform user interface.	**Query Lists** on page 7-66 in the *Oracle Utilities Energy Information Platform User's Guide* The following statements and functions have been enhanced to support query lists: • **For Each x In List Statement** on page 3-10 in the *Oracle Utilities Rules Language Reference Guide* • **LISTVALUE Function** on page 13-15 in the *Oracle Utilities Rules Language Reference Guide*
Support for Oracle Business Intelligence Publisher	This release includes support for publishing reports using Oracle Business Intelligence Publisher 10.1.3.4. The CREATEREPORT Rules Language function has been enhanced to support initiation of Oracle BI Publisher reports.	**CREATEREPORT Function** on page 13-106 in the *Oracle Utilities Rules Language Reference Guide*

Chapter 1

Introducing the Oracle Utilities Rules Language

The *Oracle Utilities Rules Language User's Guide* describes how to work with the Oracle Utilities Rules Language. This includes:

- An overview of the Rules Language and how it's used by Oracle Utilities products (this chapter)
- A description of the tools used to work with the Rules Language (**Chapter 2**: **Using the Rules Language Editor**)
- A description of how the Rules Language is processed by Oracle Utilities products (**Chapter 3**: **How Rate Forms are Processed**)
- Descriptions of the basic building blocks of the Rules Language (**Chapter 4**: **Identifiers, Constants, and Expressions**)
- An overview of the individual statements and functions available in the Rules Language (**Chapter 5**: **Statements Overview**, and **Chapter 6**: **Rules Language Functions Overview**)

Detailed information about the statements and functions available in the Rules Language is in the *Oracle Utilities Rules Language Reference Guide*.

What is the Oracle Utilities Rules Language?

The Oracle Utilities Rules Language is used to write sets of instructions (or statements) used by Oracle Utilities products to define various types of calculations and operations, such as:

- How to compute and report customer bills, trial pricing, or trial revenues (using Oracle Utilities Billing Component, Oracle Utilities Quotations Management, or Oracle Utilities Rate Management)
- How to perform load profiling and settlement (using the Oracle Utilities Load Profiling and Settlement)

These sets of instructions and statements are called *rate forms*.

Rate forms describe operations and calculations, and can also define the conditions under which specific operations and calculations should be performed (for example, based on rate codes, seasons, and/or customer characteristics). In addition, rate forms can provide instructions needed to report results, for outputting results in an electronic format that's compatible with a Customer Information System (CIS) or printer, and for saving the results of the calculations back to the Oracle Utilities Data Repository.

The language statements used in Rules Language are based on programming language statements, and are extremely powerful and flexible, enabling you to describe virtually any type of calculation, from simple to complex, and traditional to highly innovative. Many Rules Language statements use terminology and structures specific to the utility market, making them easy to learn and use.

The Rules Language's computational ability is based on simple mathematical expressions. The Rules Language also includes a range of special functions that provide access to customer interval data, historical billing determinant values, seasonal information, and utility-specific capabilities. These features include:

- Selection statements that execute different groups of statements based on specified conditions
- If-then-else statements that support complex logical expressions
- Statements that assign a label to an identifier so that reports will display the label instead of the identifier name
- The ability to override standard calculations for special events, such as backups or interruptions
- Special identifiers that accumulate values for monthly and grand summaries
- Predefined identifiers that contain customer and bill period data
- Two types of block statements that support an unlimited number of blocks and prices.

Writing Rules Language Statements and Scripts

Oracle Utilities provides three tools for constructing Rules Language statements:

- **The Rules Language Editor**
- **The Rules Language Text Editor**
- **The Rate Wizard**

All of these are described in **Chapter 2**: **Using the Rules Language Editor**. Oracle Utilities recommends using the **Rules Language Editor** because its graphical user interface simplifies the rate form creation. Instead of typing each statement, you can pick from a series of statement "templates" and fill in the blanks. This eliminates unnecessary typing and ensures that statements are constructed with the correct syntax.

What Data is Used to Perform Calculations?

The Rules Language enables you to specify, via rate forms, how various computations should be performed. The rate forms include references to the appropriate data in the Oracle Utilities Data Repository, which enables the Oracle Utilities products to perform these calculations.

For this reason, you need to understand how and where this data is stored. The following is a brief overview. A more complete description is provided in the *Data Manager's User Guide*.

Usage Data

Account usage data is stored in the Bill History Table and the Bill History Value Table. For each account, there are Bill History records for the current bill period, as well as past (historical) periods. Each bill history record is keyed by account ID and bill month (month and year), and contains the start and stop date of the bill period, and the billing determinant values for the period.

For billing purposes, all determinants in an account are billed, and the resulting revenues are added together to form the bill. Determinants are not added across accounts. This is the basic definition of an account: a unit whose determinants are computed and billed together.

Interval Data

In addition to billing determinant values stored in the Bill History tables, customer bills can include charges based on *interval data*. Interval data (or time- series data) measures customer demand or other quantities at regular intervals (such as every 5, 15, 30, or 60 minutes). You can derive billing determinants from this data, and incorporate them in bill calculations or revenue analyses, using the various interval data *functions* included in the Rules Language (see **Chapter Nine: Interval Data Functions** in the *Oracle Utilities Rules Language Reference Guide*).

Rate Forms

A *rate form* is a set of instructions (or statements) you provide to one of the Oracle Utilities products to define specific calculations and operations. The following section describes the types of statements used in creating rate forms, the various types of rate forms, and how to write, edit, and run a rate form.

Rate Form Building Blocks

The Rules Language is designed to enable you to describe *any* type of calculation.

Statements

The basic building blocks of the language are *statements*. You combine the statements into a rate form. The Rules Language consists of several types of statements, each with its own purpose and rules for use. The statements are composed of *identifiers*, *constants*, and *expressions*.

- *Identifiers* are variables used in Rules Language statements. They are the equivalent of variables in programming languages and algebra, with some added features designed specifically for use with Oracle Utilities products.
- *Constants* are values that don't change. The Rules Language supports several types of constants, including numbers, text strings, dates, and interval data *recorder,channel* references.
- *Expressions* in statements describe an operation to be performed between variables and constants (variable/variable, variable/constant, or constant/constant). The Rules Language supports three types of expressions: string, date, and arithmetic.

See **Chapter 5**: **Statements Overview** for more information about Rules Language statements. See **Chapter 4**: **Identifiers, Constants, and Expressions** for more information about identifiers, constants, and expressions.

Functions

The Oracle Utilities Rules Language includes an extensive library of *functions* designed for use in rate forms. These functions provide for arithmetic calculations, reporting and other general-purpose functions, and utility-specific calculations such as billing and rate analysis. Functions are used in Assignment statements (see **Chapter 5**: **Statements Overview**), which assign an identifier to the result of the given function. For example, an Assignment Statement could assign the 'HOURS_PER_MONTH' identifier to the result of the MONTHHOURS function (see the **MONTHHOURS Function** on page 13-35 in the *Oracle Utilities Rules Language Reference Guide*).

Rate Form Types

The three types of rate forms you can create using the Oracle Utilities Rules Language are *rate schedules, contracts,* and *riders*.

- A rate schedule is the most common type of rate form. It describes a set of calculations and/or operations to be performed by one of the Oracle Utilities products. Rate schedules are the only type of rate form that can be used as input to Oracle Utilities products. They can include one or more riders and/or contracts. When used with Oracle Utilities Billing Component or Oracle Utilities Rate Management, rate schedules describe the bill calculations for a class of customers. In other words, *a rate schedule is the Rules Language form of a rate tariff*. When used as input to one of these products, rate schedules compute revenue from billing determinants.
- A contract is a set of Rules Language statements that applies to a single account. Contracts cannot be used as input to billing or analysis programs; they must be included in rate schedules.
- The term "rider" refers to any sub-form or subroutine. A common calculation that can be used in several rate schedules or contracts can be put it in a rider. For example, a rider could contain a common determinant extraction script that creates time-of-use determinants from interval data. A rider by itself cannot be used as input to billing or analysis programs, but can include one or more riders and/or contracts.

See the description of the INCLUDE Statement in **Chapter 4**: **Identifiers, Constants, and Expressions** for more information about restrictions on the rate form versions that can be included in other rate forms.

Important note: You ***must*** include a "CANCEL/REBILL" rider in every rate schedule that will be used for Oracle Utilities Billing Component applications, if Oracle Utilities Billing Component will be used to cancel bills. This rider tells Oracle Utilities Billing Component how to process cancelled bills. See **Cancel/Rebill Rider** on page 1-8 for more information.

Rate Form Versions

Rate forms are created using the **Rules Language Editor**. Whenever changes are made to a rate form, you should save the rate form as new version. This preserves the current and previous version of the rate form for future reference or reuse.

You can create and save three types, or *versions*, of a rate form:

- **Current -** This is the version of the rate form now in effect. For any rate form, there can only be one Current version at a time.
- **Historical -** Historical versions are previous versions of a rate forms. Whenever a rate form is changed, the previous 'Current' version becomes a Historical version of that rate form. There can be any number of Historical versions of a rate form.
- **Trial -** Trial versions are used for trial analysis only. These are convenient when you are first learning how to create rate forms, and when running trial calculations based on potential changes to a rate form.

Before you can create any version, you must first set up a Rate Form record. The Rate Form record is the parent record that makes it possible to keep track of all of its versions (from the database perspective, its "child" records). See **Appendix A**: **Setting Up Rate Form Records and Rate Codes** for more information.

Sample Rate Form

The following sample rate form is a simple rate schedule used with Oracle Utilities Billing Component for a straight-line meter rate with a customer charge.

> **Note:** Text in a rate form that is located between the /* */ symbols contains comments; these do not affect processing.

```
/* Residential rate 1 */
/* Customer charge */
$CUST_CHARGE = $5.00;
/* Compute simple energy charge */
ALL KWH CHARGE $0.05 INTO $KWH_CHARGE;
/* Total */
$EFFECTIVE_REVENUE = $CUST_CHARGE + $KWH_CHARGE;
```

Writing and Editing Rate Forms

The process of creating rate forms with the Rules Language Editor includes:

1. Set up the parent record
2. Define the version
3. Write the Rules Language statements.

To bill an account using a rate form, you must also create a rate code record and an account rate code history record to link the account to the rate schedule. The mechanics of performing these steps are described in the *Data Manager User's Guide*. The remainder of this guide focuses on the features and applications of the Rules Language.

Running Rate Forms

You "run" a completed rate form by using it as input to one of the Oracle Utilities products. To run a rate form, the user selects a customer/account ID or list, a date range, and one or more rate schedules. For billing calculations performed using Oracle Utilities Billing Component, the user also selects a bill month. More information about running rate forms can be found in the appropriate Oracle Utilities documentation, including the Oracle Utilities Billing Component and Oracle Utilities Rate Management User's Guides.

Factors and Overrides

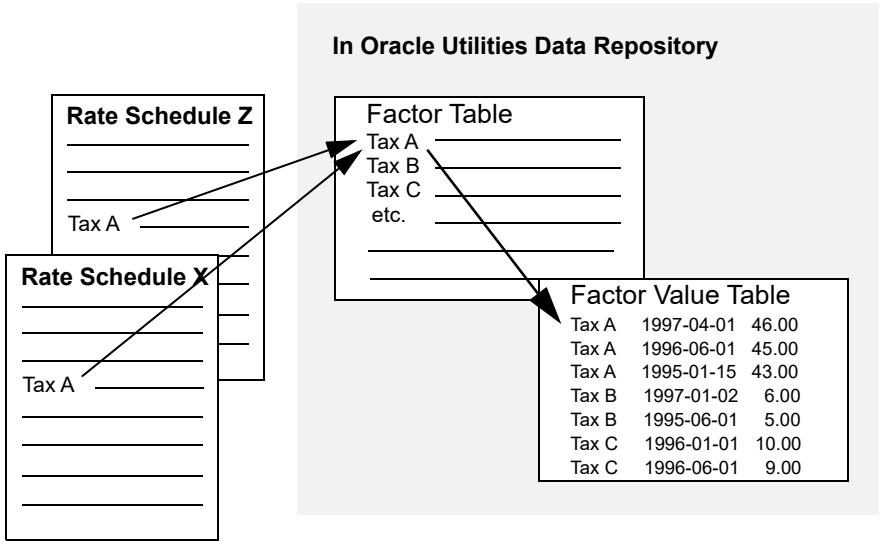
Values that vary over time and special events are handled by the Rules Language Factor and Override features, respectively.

Factors

Using factors is an efficient way to deal with values that are used widely in rate forms but are expected to change over time. Instead of using the actual value in the rate form, which would require you to update every occurrence when the value changed, you can reference the item in the rate form using a variable called a *factor name*.

Factors work as follows. The charge or other value is assigned a name in the Factor Table in the Oracle Utilities Data Repository. The current and historical values associated with the factor name, along with the values' effective dates and prorate methods, are stored in the Factor Value Table. When the value for the factor changes, you enter the new value in the Factor Value Table. See the *Data Manager's User Guide* for information on how to enter and update data in the Oracle Utilities Data Repository. See **Factor Identifiers** on page 4-7 for more information about using Factor Identifiers in the Rules Language.

When the application reads the factor name in the rate form, it automatically finds the correct value for the analysis or billing period in the database tables.



Overrides

Special events, such as charging accounts at a different rate during interruptions or backups, require the use of overrides to perform different calculations based on these special events or occurrences. You can store the dates of these periods, and their associated values, in the Oracle Utilities Data Repository. For more information about overrides and their application, see **Chapter Four**: **Setting Global Defaults and Applying Settings to Accounts, Rate Codes, and Rate Schedules** in the *Oracle Utilities Billing Component User's Guide*. See **Override Identifiers** on page 4-8 for more information about using Override identifiers in the Rules Language.

Cancel/Rebill Rider

The Oracle Utilities Rules Language enables you to specify how Oracle Utilities Billing Component processes bills that are cancelled and rebilled. For example, you can have the system update information (such as the Bill Code and Bill Time) in the Oracle Utilities Data Repository to record the event, or you can specify that the Cancel or Rebill transaction records be issued to CIS without changes to the Oracle Utilities database. Because the instructions specified in the rate schedule, not "hardcoded" in the system, you can specify whatever processing is appropriate to your utility's practices.

When Oracle Utilities Billing Component processes an account's bill, it assigns one of the following values to the predefined identifier "BILL_TYPE": "TRIAL", "CANCEL/REBILL", "ADJUSTMENT", "CANCEL", "REBILL", "CURRENT", or "FINAL". You can incorporate this identifier and values into the required rider to direct how the bills are processed.

In addition, the CISFORMT.TXT file includes a CANCEL section (the contents of this section, along with the rest of the file, are utility-specific—see your System Administrator for details). Your rider must create a record whose components match the field names specified in this section.

> **Note:** If you want to issue transaction records for the cancel/rebills, but you do not want to update the Oracle Utilities Data Repository, you must create a dummy stem identifier for the CIS records, as shown below.

Important Note

If Oracle Utilities Billing Component is used to process cancelled bills, you **must** create a rider that defines how the system processes cancelled bills, and you must include it in every rate schedule.

Example

Following is a sample CANCEL/REBILL rider:

```
/* CANCEL and CANCEL/REBILL Rider */
IF ((BILL_TYPE = "CANCEL") OR (BILL_TYPE = "CANCEL/REBILL"))
THEN
   /* Process all the CISACCOUNTs here. */
   /* */
   /* ... */
   /* */
   /* Force CANCEL_CIS_REC to be a stem identifier. */
   CANCEL_CIS_REC.DUMMY = 0;
   /* Write CIS Record, format based on the CANCEL section */
   SAVE CANCEL_CIS_REC to CIS SECTION "CANCEL";
   CLEAR CANCEL_CIS_REC;
   /*  */
   /*  All done if CANCEL */
   /*  */
IF (BILL_TYPE = "CANCEL")
THEN
   /* Set total revenue to zero.*/
   $EFFECTIVE_REVENUE = $0.00;
   /* Stop  */
   DONE:
   END IF:
END IF;
```

Chapter 2

Using the Rules Language Editor

This chapter explains how to use the **Rules Language Editor** to assemble Rules Language statements and other elements into a rate form script. This includes:

- **The Rules Language Editor**
- **Adding, Modifying, and Deleting Statements**
- **Using the Rules Language Elements Editor**
- **Saving a Completed Rate Form**
- **Other Tools for Writing Rate Forms**

The instructions on using the Rules Language Editor to create and modify rate form scripts assume that you have already created the prerequisite Rate Form records in the Oracle Utilities Data Repository, as described in detail in **Appendix A**: **Setting Up Rate Form Records and Rate Codes**.

The Rules Language Editor is a component of Data Manager. To use it, begin at the Data Manager desktop (if you are unfamiliar with how to start Data Manager, see the *Data Manager User's Guide*).

The Rules Language Editor

How you open the Rules Language Editor depends upon whether you are creating a new version of a rate form, or editing an existing version.

How to open a new rate form version:

1. From the Data Manager desktop, select **File->New->[rate form type] Version**. You can use the editor to write all three types of rate forms:

   Rate Schedule: A set of Rules Language statements that calculates charges for a rate class or an account. A rate schedule is the only type of rate form that can be used as input to the analysis or billing programs.

   Contract: A set of Rules Language statements that applies to a single account. (A contract alone cannot be used as input to a billing or analysis program. It must be included in a rate schedule.)

   Rider: A set of Rules Language statements that can be included in rate schedules or in other riders. A rider is typically used to write standardized routines, such as saving data to the database or writing transaction records, that you'll use in many rate schedules. A rider can also be a tariff rider.

   When you've specified the type of rate form you want to write, the **New [Rate Form] Version** dialog box appears.

2. Select the appropriate **Operating Company** and **Jurisdiction** for the rate form you wish to open.

   The list of currently available Rate Form records appears in the list box. If the desired rate form title does not appear in the list box, it may belong to an operating company/jurisdiction pair other than you one you specified. If you have not created the rate form record in the Oracle Utilities Data Repository, see **Appendix A**: **Setting Up Rate Form Records and Rate Codes**.

3. Highlight the desired Rate Form in the list box.

4. For a current or historical version, enter its start date (but no version number). See **Rate Form Versions** on page 1-5 for more information. For a trial version, enter a version number (but no start date).

   If you are creating a trial rate schedule and plan to include a trial contract in it, specify a version number between 9000-9999 (inclusive) for both the rate schedule and the contract. Otherwise, the system will automatically include the current version of the contract instead of the trial version.

   The difference between historical and current versions is that the current version has the most recent start date. Current versions are for billing. Current, trial, and historical versions can be used for analysis. You can create virtually any number of trial or historical versions, but there can be only one current version.

5. Click **OK**. The descriptive "header" information you just input is saved to the database, and an empty Rules Language Editor window opens. You can now create a rate form script using the statements and elements described in later chapters of this guide. The remainder of this chapter describes the use of the Editor to create and modify the statements. See **Adding, Modifying, and Deleting Statements** for more information.

6. *Optional.* If you want to import the contents of a previously exported rate form (*.prg) into the new rate form, click **Import**. The File Import New Rate Schedule Version dialog opens.

   Navigate to the appropriate file and click **Open**. A Rules Language Editor window will open, displaying the contents of the imported rate form. If there is an error importing a rate form and you fix the rate form without closing the New Rate Form dialog, when you click on Import, the previous file name will be remembered.

How to open an existing rate form version:

1. At the Data Manager desktop, select **File->Open->[rate form type] Version**. The **Open [Rate Form] Version** dialog box appears.
2. Select the **Operating Company** and **Jurisdiction** your rate form belongs to. The list of currently available Rate Form records appears in the list box. If the desired rate form title does not appear in the list box, it may belong to an operating company/jurisdiction pair other than you one you specified.
3. Highlight the desired Rate Form in the list box.

   A list of the versions that were previously created and stored for the selected rate form appears in the middle list box. You can "filter" this using the **Type** checkboxes: check the types you want to see in the list, and uncheck those that you don't.

   The **Locked** checkbox indicates that the selected rate form is locked and cannot be edited (though they can be opened).

   When you highlight a version in the middle list box, the text of the rate form script appears in the lower box. This view is read-only. The rate form cannot be edited in this view.
4. *Optional.* If you want to export the contents of the selected rate form to a file, click **Export**. The **File Export Rate Form As** dialog opens.

   Navigate to the desired destination directory, type in the desired filename, and click **Save**.
5. In the middle list box, highlight the version you wish to edit and click **Open**. The **Rules Language Editor** window opens. You can now update the rate form script using any of the methods described in this chapter.

   If you assigned color-coding to specific statements, the keywords for those statements appear in the designated colors. See **Rules Language Settings** on page 2-18 in the *Data Manager User's Guide* for more information about color-coding Rules Language Statements.

   If you selected the **Display Line Numbers** option on the **Rate Analysis** tab of the **Default Options** dialog, each line in the rate form begins with a line number and colon. See **Rate Analysis Options** on page 2-11 in the *Data Manager User's Guide* for more information.

   > **Note:** You can open multiple rate forms (of the same type) at the same time by pressing the **Shift** or **Ctrl** key while selecting the rate forms. The selected rate forms will open in separate Rules Language Editor windows.
   >
   > **Note:** Only one copy of a single rate form can be opened at any one time. If you attempt to open a rate form that is currently being edited by another user, a warning dialog will appear on the screen informing you that the rate form is locked.

Adding, Modifying, and Deleting Statements

Opening the Rules Language Editor accesses the variety of available tools for creating, modifying, and deleting statements.

> **Note:** The maximum number of characters possible in a rate form using Windows 98 is 32,000.

Working with the Rules Language Editor

Rules Language statements and other editor functions can be accessed either from the **Statements** menu or by selecting a line in the **Rules Language Editor** and clicking the right mouse button. If you select a line on the **Rules Language Editor** and click the *right* mouse button, a menu containing the **Cut**, **Copy**, **Paste,** and **Undo** functions from the **Edit Menu**, and the contents of the **Statements Menu**, opens. You can select options from this menu, or from the Menu Bar menus. If you select the **Text Editor** option, the cursor will be on the line you selected in the **Rules Language Editor**.

How to add a statement to a rate form:

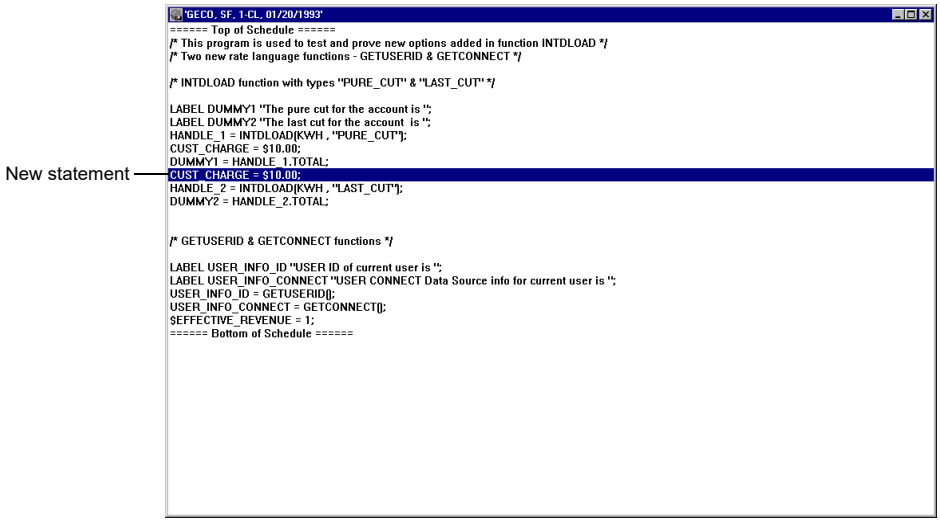
1. Select **Statements->[statement type]**.

   A template for the selected statement type appears. Complete it as desired. For detailed information about a specific statement type, see **Chapter 6**: **Rules Language Functions Overview** and the corresponding chapter in the *Oracle Utilities Rules Language Reference Guide*.

   You can type the elements into the fields, or you can use a second editing tool, called the **Rules Language Elements** editor, to pick them. To open the **Rules Language Elements** editor, position the mouse pointer in any template field and click the *right* mouse button. See **Using the Rules Language Elements Editor** on page 2-6 for more information.

   Note: The Rules Language requires the use of straight quotes (" "). If copying/pasting text from other editors, take note of any non-straight quotes (“ ”) and correct as needed.

2. When you have completed the template, click **OK**. Your statement appears in the Editor window.



How to delete a statement from a rate form:

1. Highlight the statement.
2. Select **Edit->Delete,** or click the *right* mouse button and select **Delete**.

How to insert a statement between existing lines:

1. Highlight the line in the script that's above the desired position of the new statement.
2. Select **Statements-›[statement type]**. The template for the selected statement type appears. For detailed information about a specific statement type, see **Chapter 6**: **Rules Language Functions Overview** and the corresponding chapter in the *Oracle Utilities Rules Language Reference Guide*.
3. When you have completed the template, click **OK**. Your statement appears in the Editor window.

How to move a statement within a rate form:

1. Highlight the statement you wish to move.
2. Select **Edit-›Cut,** or click the *right* mouse button and select **Cut**.
3. Highlight the line above the desired position for the statement.
4. Select **Edit-›Paste,** or click the *right* mouse button and select **Paste**.

How to copy lines from another rate form:

1. Open the rate form that contains the lines you want to copy, and the rate form you want to copy those lines to. You will have two Editor windows open on your desktop.
2. Highlight the lines you wish to copy.
3. Select **Edit-›Copy,** or click the *right* mouse button and select **Copy**.
4. In the first window, highlight the line above the desired position for the copied statements.
5. Select **Edit-›Paste,** or click the *right* mouse button and select **Paste**.
6. Close the second window by clicking on the Close (X) button in the upper right corner.

How to view an INCLUDEd rate form:

1. Open the rate form that contains the INCLUDE statement.
2. Highlight the line that contains the INCLUDE statement.
3. Click the *right* mouse button and select **Open Rider**.
4. The INCLUDEd rate form opens in a new **Rules Language Editor**.

Using the Rules Language Elements Editor

An additional editing tool, the **Rules Language Elements** editor, is built into the Rules Language Editor. This tool helps you construct statements correctly and efficiently by enabling you pick any of the Rules Language elements from menus instead of typing them manually. It is a "smart" editor that tailors the list of choices presented to you based on the type of statement, or the portion of the statement, that you're constructing.

This section describes the Rules Language Elements Editor, including:

- **Rules Language Element Types**
- **Working with the Rules Language Elements Editor**

Rules Language Element Types

The Rules Language Elements Editor lists the different Rules Language Element types in the Element Types box. When you select an Element Type, the available elements of that type appear in the lower box of the editor. The available element types include:

- **Revenue Identifiers**: Revenue identifiers established in the open rate form. The pre-supplied $EFFECTIVE_REVENUE is always included in this list. See **Revenue Identifiers** on page 4-4 for more information.
- **Bill Determinant Identifiers**: Bill determinant identifiers as defined in the Bill Determinants table. See **Bill Determinant Identifiers** on page 4-5 for more information.
- **Database Identifiers**: Available database identifiers. See **Database Identifiers** on page 4-5 for more information.
- **Interval Data Handles**: Interval data handles in the open rate form. See **Interval Data Handles** on page 4-6 for more information.
- **Time-of-Use Handles**: Time-of-use interval data handles in the open rate form. See **Time-of-Use Handles** on page 4-6 for more information.
- **Other Identifiers**: Other identifiers established in the open rate form. See **Identifiers** on page 4-2 for more information.
- **Identifiers as Factor Value Key**: Identifiers established in the open rate form used as the key for a factor identifier. When inserted into the rate form, these appear in the FACTOR[<identifier>].VALUE format. See **Factor Identifiers** on page 4-7 for more information.
- **Identifiers as Override Value Key**: Identifiers established in the open rate form used as the key for an override identifier. When inserted into the rate form, these appear in the OVERRIDE[<identifier>].VALUE format. See **Override Identifiers** on page 4-8 for more information.
- **Identifiers as Override String Key**: Identifiers established in the open rate form used as the key for a override identifier. When inserted into the rate form, these appear in the OVERRIDE[<identifier>].STRVAL format. See **Override Identifiers** on page 4-8 for more information.
- **Factor Value Identifiers (All)**: Factors from the Factor table used in a factor identifier. When inserted into the rate form, these appear in the FACTOR[<factor_code>].VALUE format. See **Factor Identifiers** on page 4-7 for more information.
- **Factor Value Identifiers (Rate Form)**: Factors from the Factor table used in a factor identifier that have the same Operating Company and Jurisdiction as the open rate form. When inserted into the rate form, these appear in the FACTOR[<factor_code>].VALUE format. See **Factor Identifiers** on page 4-7 for more information.

- **Charge Factor Value Identifiers (All)**: Factors from the Factor table used in a factor identifier that have a Unit-of-Measure of 79 (Dollars). When inserted into the rate form, these appear in the FACTOR[<factor_code>].VALUE format. See **Factor Identifiers** on page 4-7 for more information.
- **Charge Factor Value Identifiers (Rate Form)**: Factors from the Factor table used in a factor identifier that have the same Operating Company and Jurisdiction as the open rate form and that have a Unit-of-Measure of 79 (Dollars). When inserted into the rate form, these appear in the FACTOR[<factor_code>].VALUE format. See **Factor Identifiers** on page 4-7 for more information.
- **Override Float Value Identifiers**: Overrides from the Override table used in an override identifier. When inserted into the rate form, these appear in the OVERRIDE[<identifier>].VALUE format. See **Override Identifiers** on page 4-8 for more information.
- **Override String Value Identifiers**: Overrides from the Override table used in an override identifier. When inserted into the rate form, these appear in the OVERRIDE[<identifier>].STRVAL format. See **Override Identifiers** on page 4-8 for more information.
- **Interval Data/Meter Value Functions**: Interval Data (INTD) and Meter Value (MV) functions. See **Interval Data Functions** on page 6-2 for a list of available Interval Data functions. See **Meter Value Functions** on page 6-8 for a list of available Meter Value functions.
- **Interval Data Function Parameters**: Parameters used by Interval Data functions. See **Interval Data Functions** on page 6-2 for a list of available Interval Data functions.
- **Interval Data Attributes**: Attributes of interval data handles. See **Interval Data Reference Values and Attributes** on page 7-3 for more information.
- **Math Functions**: Math functions. See **Math Functions** on page 6-9 for a list of available Math functions.
- **String Functions**: String functions. See **String Functions** on page 6-11 for a list of available string functions.
- **Other Functions**: All other functions. See **Other Functions** on page 6-12 for a list of available other functions.
- **Stored Procedure Names**: Stored procedures in the Oracle Utilities Data Repository. These are used by the **CALLSTOREDPROC Function** on page 13-4 in the *Oracle Utilities Rules Language Reference Guide*. When inserted into the rate form, these appear as a string.
- **Function Parameters**: Parameters used by Rules Language functions. See **About Parameters** on page 6-24 for more information.
- **IDATTR Attributes**: Attributes available through use of the **IDATTR Function** on page 13-59 in the *Oracle Utilities Rules Language Reference Guide*. When inserted into the rate form, these appear as a string.
- **LSRS Environment (Get)**: Rate Schedule Environment Identifiers that can be read. See **Rate Schedule Environment Identifiers** on page 4-16 for more information.
- **LSRS Environment (Set)**: Rate Schedule Environment Identifiers that can be set. See **Rate Schedule Environment Identifiers** on page 4-16 for more information.
- **LSRPT Options**: Report Options Identifiers. See **Report Options Identifiers** on page 4-19 for more information.
- **Table-Column Lists**: Table-column lists stored in the Oracle Utilities Data Repository. When inserted into the rate form, these appear as a string. See **Chapter 8**: **Working with Lists and Queries** in the *Data Manager User's Guide* for more information.

- **Table-Column Channel Lists**: Table-column lists based on the Channel and/or Channel History tables in the Oracle Utilities Data Repository. When inserted into the rate form, these appear as a string. See **Chapter 8**: **Working with Lists and Queries** in the *Data Manager User's Guide* for more information.
- **Factor Codes (All)**: Factor codes from the Factor table. When inserted into the rate form, these appear as a string.
- **Factor Codes (Rate Form)**: Factor codes from the Factor table that have the same Operating Company and Jurisdiction as the open rate form. When inserted into the rate form, these appear as a string.
- **Override Codes**: Override codes from the Override table. When inserted into the rate form, these appear as a string.
- **UOM Codes**: Unit-of-Measure (UOM) codes from the UOM table. When inserted into the rate form, these appear as a string.
- **End Use Codes**: End Use codes from the End Uses table. When inserted into the rate form, these appear as a string.
- **Service Codes**: Service codes from the Service table. When inserted into the rate form, these appear as a string.
- **Aggregation Group Names**: Aggregation groups defined in the Aggregation Group table. When inserted into the rate form, these appear as a string.
- **Tables**: Tables in the Oracle Utilities Data Repository. When inserted into the rate form, these appear as a string.
- **Time-of-Use Schedules**: Time-of-Use schedules stored in the Oracle Utilities Data Repository. When inserted into the rate form, these appear as a string. See **Time-of-Use Schedules** on page 7-8 in the *Data Manager User's Guide* for more information.
- **Time-of-Use Periods**: Time-of-Use periods stored in the Oracle Utilities Data Repository. When inserted into the rate form, these appear as a string. See **Time-of-Use Schedules** on page 7-8 in the *Data Manager User's Guide* for more information.
- **Season Schedules**: Season schedules stored in the Oracle Utilities Data Repository. When inserted into the rate form, these appear as a string. See **Season Schedules** on page 7-7 in the *Data Manager User's Guide* for more information.
- **Season Periods**: Season periods stored in the Oracle Utilities Data Repository. When inserted into the rate form, these appear as a string. See **Season Schedules** on page 7-7 in the *Data Manager User's Guide* for more information.
- **Holiday Lists**: Holiday lists stored in the Oracle Utilities Data Repository. When inserted into the rate form, these appear as a string. See **Holidays** on page 7-6 in the *Data Manager User's Guide* for more information.
- **Save to CIS Section Names**: Section names of the sections of the CISFORMT.TXT file stored in the C:\LODESTAR\CFG directory. When inserted into the rate form, these appear as a string. See **Creating a CIS Transaction Record Output File** on page 9-1 in the *Oracle Utilities Energy Information Platform Configuration Guide* for more information.
- **Distribution Nodes**: Distribution nodes from the Distribution Node table. When inserted into the rate form, these appear as a string.
- **Account/Customer Lists**: Account/customer lists stored in the Oracle Utilities Data Repository. When inserted into the rate form, these appear as a string. See **Chapter 8**: **Working with Lists and Queries** in the *Data Manager User's Guide* for more information.

Working with the Rules Language Elements Editor

This section describes how you work with the Rules Language Elements Editor.

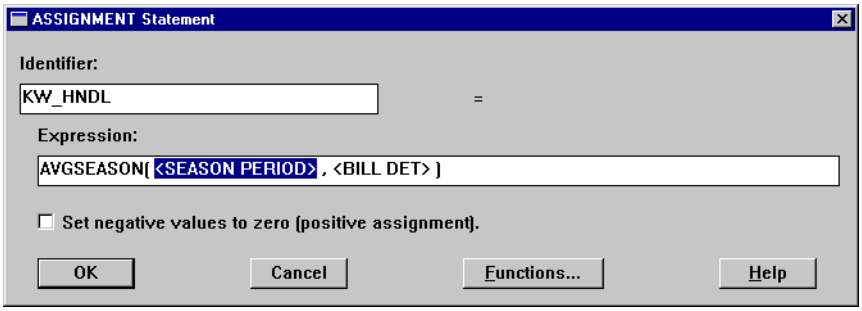
How to open the Rules Language Elements Editor:

1. In any statement template, position the mouse pointer in the field you wish to complete. Click the *right* mouse button. The Rules Language Elements editor appears. The list box on the top of the Editor displays all of the Rules Language element types that you might use to complete the template field you're in.
2. Highlight the desired element type in the upper list box. The list box on the bottom displays all of the elements in the highlighted category.
3. Highlight the desired element in the lower box; it appears in the field above the list box.
4. Click **OK**. The element appears in the template.

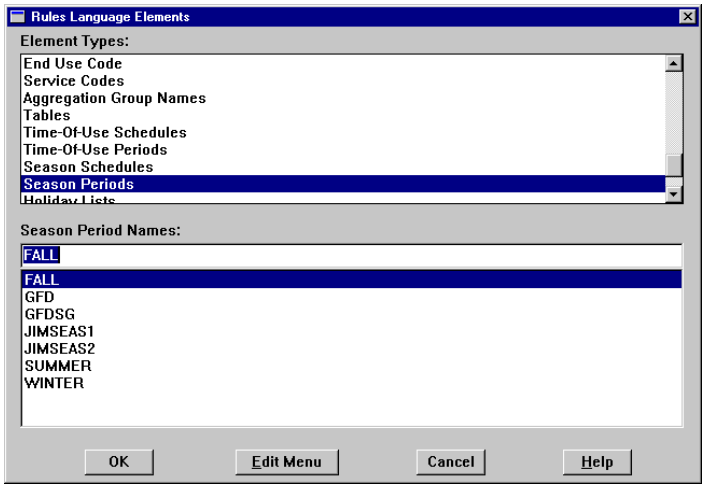
How to select a parameter:

Some elements include other elements that you must specify. For functions, you will usually specify one or more parameters. You can use the Rules Language Elements editor to specify these elements.

1. In the template, highlight the parameter you want to select an element for. Include the caret symbols (< >) on either side.



2. With the mouse pointer still positioned in the field, click the *right* mouse button. The Rules Language Elements Editor opens with the appropriate category for this parameter highlighted in the upper list box.



3. Highlight the desired parameter in the lower list box and click **OK**. Your selection appears in the template.

   See **Chapter 4***:* **Identifiers, Constants, and Expressions** for more information about function parameters.

Saving a Completed Rate Form

How to save a new rate form version script:

1. Select **File->Save**. The message "Rate Schedule Version Saved" appears in the lower left corner of the Data Manager screen.

   Each time a rate form is saved, a "Saved by" comment is included as the first line in the rate form. This comment contains the name, date, and time of the last person to save the rate form. This comment is in the format "Saved by <name> on <date and time>". Comments are added every time the rate form is saved, which provides a history of the changes. The comments will begin with the // delimiter to make the comment a one-line comment (as opposed to a multi-line comment). These regular comments that can be edited and deleted by the user.

   Note: "Saved by" comments can be included at the bottom of the rate form instead of the top by selecting the "Display 'Saved by' comments at end" option on the Rate Analysis tab of the Default Options dialog. See **Rate Analysis Options** on page 2-11 in the *Data Manager User's Guide* for more information.

How to save a modified rate form version script as a new version:

1. Select **File->Save As** and complete the dialog box that appears.

Important Note

Before you can bill an account using a rate schedule, you must create a rate code record for the rate schedule, and an account rate code history record for the account. These two records link the rate schedule to the account. See the *Oracle Utilities Billing Component User's Guide* for instructions.

Printing a Rate Form

How to Print a Saved Rate Form:

1. Open the rate form you wish to print using the Rules Language Editor.
2. Select **File->Print**. Note that this option prints only the text currently displayed in the Rules Language Editor. It does NOT print the contents of INCULDEd riders/contracts.
3. *Optional*. Select one of the **Statements->View** options. The rate form will appear in a report window with a comment that includes the rate form name (including Operating Company code, Jurisdiction code, Rate Form Code, and version), followed by "as of" and the date and time. This allows the printout of the report to include a date/time stamp, as well as any INCLUDEd riders/contracts.

Other Tools for Writing Rate Forms

This section describes the other tools available for writing rate form scripts; **The Rules Language Text Editor** and **The Rate Wizard**.

The Rules Language Text Editor

The **Rules Language Text Editor** allows you to edit the rate form script just as you would edit any text document, using a text editor or word processing application.

> **Note:** The **Rules Language Text Editor** is designed for use by experienced users. If you're just learning to write rate forms, use the Rules Language Editor until you become familiar with the syntax and structure of rate forms.

To open the Rules Language Text Editor, open an existing rate form or create a new rate form. When a rate form is open in the Rules Language Editor select **Statements->Text Editor**, or double-click on the '===Top of Schedule===' or '===Bottom of Schedule===' lines in the Rules Language Editor. Either method opens the Rules Language Text Editor window.

Menus and Commands

The Rules Language Text Editor provides menus and options for editing rate form scripts. The menus are accessible from the Menu bar, or by clicking the *right* mouse button.

File Menu

Validate - This option validates the rate form. If there is an error in the rate form, a 'Validate Error' dialog will open, with the line containing the error highlighted.

Update and Close - This option validates and saves the rate form, and closes the Rules Language Text Editor. If there is an error in the rate form, a 'Validate Error' dialog will open, with the line containing the error highlighted.

Update and Save - This option validates and saves the rate form. If there is an error in the rate form, a 'Validate Error' dialog will open, with the line containing the error highlighted.

Close - This option closes the Rules Language Text Editor. If you haven't saved your work, you will be prompted to do so.

Edit Menu

Undo - This option undoes the last action. The maximum number of Undos allowed is defined on the Rate Analysis tab of the Default Options dialog.

Redo - This option redoes the last action undone.

Cut - This option cuts the selected text and places it on the Clipboard.

Copy - This option copies the selected text to the Clipboard.

Paste - This option pastes the contents of the Clipboard.

> **Note:** The maximum number of characters that can be copied or pasted into the Rules Language Text Editor is approximately 32,000. Copying/pasting more characters may lead to errors.
>
> **Note**: The Rules Language requires the use of straight quotes (" "). If copying/ pasting text from other editors, take note of any non-straight quotes (“ ”) and correct as needed.

Find - This option allows you to search for a word or words in the rate form. When you select this option or press **F3**, the Find dialog opens.

Enter the word or words you wish to find and click **OK**.

Find Next - This option finds the next instance of the word or words last entered in the Find dialog.

Find Previous - This option finds the previous instance of the word or words last entered in the Find Dialog.

Go To - This option opens the Go To tab on the Find dialog, allowing you to go to a specific line in the rate form.

Replace - This option allows you to find a word or words in the rate form and replace them. When you select this option, the Replace dialog opens.

Fixed Font - This option changes the text to a fixed font size.

Rules-Language-Elements

This option opens the **Rules Language Elements Editor**.

Help

This option opens the Data Manager Online Help.

The Rate Wizard

The Rate Wizard is a training tool to help you understand the steps involved in writing rate form statements. Use it to create statements based on Bill Determinants, as well as other charges, by making selections on a number of dialog boxes (similar to wizards found in other Windows applications).

The Rate Wizard doesn't create finished rate forms. After you save and close the rate form, you'll need to go back and edit it using either the **Rules Language Editor** or **The Rules Language Text Editor**.

How To Use the Rate Wizard to Write Rate Form Statements:

1. Select **File->New->Rate Wizard**. The **Rate Wizard** window opens.
2. On the **Pick Determinants** tab, select the determinant (or one of the determinants) you want to use in your rate form, and click **Next**. The **Data Source** dialog box opens.
3. Select the appropriate data source for the determinant you're billing ('From Interval Data' or 'Scalar Data') and click **Next**. If you select 'From Interval Data,' the TOU dialog appears. If you select 'Scalar Data,' the Ratchet dialog opens.
4. Select the appropriate type of interval data and click **Next**. If you select **RTP**, the **RTP** dialog opens. If you select **TOU**, the **Select Period** dialog opens. If you select **Other**, the **Ratchet** dialog opens. Each of these dialogs is described below.
5. If you selected **RTP**, select the appropriate Usage Data, Pricing Data, and Customer Base Line for your rate form using the drop-down lists. When you've filled in the dialog, click **Finish**. This returns you to the opening Rate Wizard window. From there you can select another determinant or other charge to bill, or close and save the rate form.
6. If you selected **TOU**, select the desired TOU schedule from the drop-down list, and the specific TOU period that applies to the rate form statement you're writing. Click **Next**. The **Select Period** dialog opens.
7. The **Ratchet** dialog opens after you complete the **Select Period** dialog, or if you selected 'Scalar Data' in the **Data Source** dialog. Select **Yes** or **No** based on whether the calculation requires ratchet data.
8. If you select **Yes**, the **Do Ratchet** dialog opens. Enter the number of historic months to use, the minimum contract (if any), and whether the current month determinant should be used, and click **Next**.
9. If you select **No** on the **Ratchet** dialog (or if you completed the **Do Ratchet** dialog), the **Set Price** dialog opens.
10. Select the charge type: **Block Charge** or **Flat Charge**.

    If you select **Block Charge**, the BLOCK Statement template opens. See **Block Statements** on page 4-4 in the *Oracle Utilities Rules Language Reference Guide* for information about filling in the template. After you've filled out the template, click **Finish** to return to the **Pick Determinants** tab.

    If you select **Flat Charge**, the ALL Statement template opens. See **All Statement** on page 4-2 in the *Oracle Utilities Rules Language Reference Guide) for* information about filling in the template. After you've filled out the template, click **Finish** to return to the **Pick Determinants** tab.

    If you have other determinants to bill, follow the above procedure for each determinant. If you have other charges to include in your rate form, click on the **Other Charges** tab and enter them, as described in Step 11 below.
11. The **Other Charges** tab allows you to add additional statements based on other charges, such as a flat customer charge, to your rate form. To add a charge using this window, enter the Identifier in the left box (located below the **Add/Update** button), enter the value associated

with the identifier in the right box, and click **Add/Update**. This adds the statement to the **Enter other charges and set their prices one at a time** box in the center of the tab. See **Chapter 4**: **Identifiers, Constants, and Expressions** for more information about Identifiers.

12. To edit an existing statement, highlight the statement, make any desired changes in the left and right boxes, and click **Add/Update**. To delete an existing statement, highlight the statement and click **Delete**.

    If you have other charges to bill, follow the above procedure for each. If you have determinants to include in your rate form, click on the **Pick Determinants** tab and follow the procedure outlined in steps 1 through 10 above.

13. If you're done with your rate form, click the **Save & Close** button. The **Save Rate Form Version As** dialog opens. Save the rate form as described under **Saving a Completed Rate Form** on page 2-10.

Chapter 3

How Rate Forms are Processed

Rate forms must be designed correctly to get the proper results, and the execution process is where the results are determined. When a rate form is processed, it is referred to as a 'run.'

This chapter describes how rate forms are processed, including:

- **Overall Execution Process**
- **Saving Data from a Rate Schedule**

Overall Execution Process

The rate form execution process is essentially the same for all types of rate schedule runs: the user selects a customer/account ID or list, a date range, and one or more rate schedules. For billing calculations performed using Oracle Utilities Billing Component, the user also selects a bill month. Then, the following steps are performed:

> **Note:** Some of the steps outlined below apply only to specific products. For these steps, the appropriate product is shown in parentheses.

1. The appropriate rate schedules or contacts are loaded. All INCLUDEd rate forms are loaded, and the INCLUDE statements are replaced in the rate form by the INCLUDEd rate forms. The result is a "completed" rate form with no INCLUDE statements.
2. Each rate form is validated and compiled. Compilation converts the text of the rate form into a format the program can execute efficiently. Part of this format is the Symbol Table. The Symbol Table contains all the identifiers (and related information) used in the rate form. During this step, the DETERMINANT, REVENUE and LABEL statements in the rate forms are processed, and their corresponding labels are put in the Symbol Table (because they cannot change during a run).
3. A list of determinants used in historical or season functions is extracted from the Symbol Table. This allows retrieval of historical data for only those determinants that need it.
4. The values of SEASON_SCHEDULE_NAME and HOLIDAY _NAME identifiers are extracted from the Symbol Table. If they are assigned in an Assignment Statement, the value is assigned at compile time, rather than at run time. This makes the names available before execution of the rate form, so the appropriate schedules can be loaded and verified.

   If all of the above steps are successful, processing may begin.
5. (Oracle Utilities Rate Management only) Depending on the type of analysis, either all months for an account (Customer Impact), or all accounts for a month (Customer Revenue) are processed.
6. The corresponding bill history record for the supplied account and bill month is read from the Oracle Utilities Data Repository. Determinant and input values are loaded into the Symbol Table, and historical values are read and loaded for the determinants that need them.
7. The names of all database identifiers are retrieved, and the value of each one is read from the Oracle Utilities Data Repository and stored in the Symbol Table.
8. (Oracle Utilities Billing Component/Oracle Utilities Rate Management only) The rate code to be used is found. The hierarchy of decisions that determine the rate code used is:
   a. If the rate code is supplied as user input, the input value is used.
   b. If not supplied and the rate schedule is the account's rate schedule, the "real" rate code is used.
   b. If another rate schedule, use inter-schedule mapping, if one exists.
   b. Use the "real" rate code.
9. The rate form is processed. Any errors are reported.
10. (Oracle Utilities Rate Management only) The Symbol Table is accumulated. This means that the just-computed values of the determinant and revenue identifiers are added to their previously computed values.
11. The Symbol Table is summarized. All identifiers to be reported (including determinant, revenue, input, and other identifiers) are put into a standard format that external report writers can process.
12. The Symbol Table is reset, and all computed values and flags are set to zero.

13. The next account (when using Oracle Utilities Billing Component) or account and month (when using Oracle Utilities Rate Management) is processed. Return to Step 6.

Which Rate Form?

An important consideration in processing rate forms for billing calculations and/or rate analyses is which rate form is to be used in a particular bill month. For rate forms processed through RUNRS or Trial Bill/Calculation, the rate schedule or contract is specified (by the user or in an INCLUDE Statement) and used as-is. Otherwise, the version picked depends on the product and type of analysis.

Oracle Utilities Billing Component

For Oracle Utilities Billing Component, the version of each rate form in effect on the stop date of the account's bill period is used. In this case, each customer's completed rate form is created individually. All INCLUDEd rate forms must have a version effective on the stop date, or an error will occur.

Oracle Utilities Rate Management

For Customer and Bill Frequency Revenues with the "Use historical schedule versions" flag set, the rule is that, for each month selected, a rate form is created that INCLUDEs the versions effective at the beginning of the month. In this case there is one completed rate form per month.

For all other Oracle Utilities Rate Management analyses, the specified contract or rate schedule version is used. For any INCLUDEd rate forms, the version effective on the first day of the analysis bill period is used. In this case there is one completed rate form for the whole analysis.

Shared Symbol Table

Each run reloads and re-uses the same Symbol Table. If several rate forms are created for an initial contract or rate schedule, their compiled format shares a common Symbol Table. This is necessary so that the accumulate step above (Step 10) accumulates across multiple rate forms. This means that the HOLIDAY_NAME and SEASON_SCHEDULE_NAME values and identifier labels are also shared. The value of these will be that of the latest rate form to assign them. If the current version is present, these will be set to its values.

Saving Data from a Rate Schedule

Saving data from a rate schedule via Oracle Utilities Billing Component includes:

- An overview of the types of data that can be saved from a rate schedule
- An overview of how bill pages/rate schedule runs are related
- A description of the requirements for saving data from a rate schedule
- A description of how transactions are processed, and the specific types of data that can be saved via a "Postponed Save" (after user approval)
- A description of a solution for saving all types of data from a rate schedule.

Types of Data

The types of data that can be saved from a rate schedule include:

- Relational Database Records (can be added, updated, or deleted)
- Interval Data Cuts (can be saved or deleted)
- Financial Charges
- Billing Determinants
- Relational Database Column Updates (via the LISTUPDATE function)
- CIS Records.

In addition, account notes may need to be written during a bill calculation.

Related Bill Pages

In all billing modes (Automatic, Approval Required, Current/Final, Bill Correction, and Trial Bill/Report) related bill pages and/or rate schedule runs are grouped. One bill page represents one bill history record processed by one rate schedule. Bill pages are related and grouped when:

a. They are all for accounts that have the same summary customer

b. They are for the same account, with the same scheduled read date. These are done in a nested loop. The outer loop is by bill history record, the inner is by rate schedule.

b. They are for all bill periods to be corrected (for Bill Correction only).

Oracle Utilities Billing Component includes an option to allow for approval of all pages that were error-free.

Requirements

Requirements that affect how data can be saved include:

1. All changes a rate schedule makes to either database (Data Repository or Interval Database) must be available to later, related rate schedule runs.
2. The user must have the capability of approving or rejecting ALL saves, except in Automatic billing mode.
3. For Automatic and Approval Required billing, account notes may be written during the billing process.
4. Interval data saves and deletes should be made only after user approval (see Step 2, above).

Transactions

All changes to the relational database take place in the context of a transaction. All changes within a transaction are committed together, or all are rolled back (removed) together. In all billing modes, all related rate schedules are run in a single transaction that is started before the first rate schedule in the group is run. After the last rate schedule in the group is run, there are several options:

1. If there were any errors, roll back all saves.
2. If the user will approve the group later, roll back the changes and redo them when approved.
3. When using RUNRS or Automatic Billing, commit the saves, then write out CIS records.

These options do not include saving interval data stored in the Oracle Utilities Data Repository.

Postponed Saves

To support user approval, some of the data to be saved is stored in the report data structure (the in-memory copy of the report). Data stored there includes:

- Relational Database records
- Billing Determinants
- CIS records.

No other types of saved data are stored for user approval. All other data must be committed after a run or group of runs.

> **Note:** When the SAVE TO TABLE statement runs, records are usually written - temporarily - to the relational database to verify the correctness of the record. See the configuration parameter NO_TEMP_SAVE for the exception.

Note that interval data, XML data, and financial charge updates stored in the Oracle Utilities Data Repository **are not** exempt from the relational database commits. To save them (do a commit), all other database changes will also be committed. This contradicts the requirement for user approval of saves.

Two Phase Commit

The solution is:

1. All billing modes have additional approval choices:
   a. Automatically save/approve group of related pages if there are no errors
   b. Automatically save/approve each page if there are no errors.

   Either of these options will approve the bill and commit all saves if there are no errors.

3. If the rate schedule(s) are expected to save/delete interval data in the relational database, use the LISTUPDATE function, or possibly modify financial data that the user will want to approve, the user must:

    a. Run the bill with Enable user approve/reject of saves

    b. If there are no errors, approve the bill. The bill will be rerun (in the same billing mode) with Automatic Approval. This second run will generate the same results, and save all the data.

Chapter 4

Identifiers, Constants, and Expressions

This chapter describes in detail the elements that you use when composing Rules Language statements, including:

- **Identifiers**
- **Constants**
- **Expressions**

Specific information concerning the types of statements and function available with the Rules Language can be found in the *Oracle Utilities Rules Language Reference Guide*.

Identifiers

Identifiers are variables used in Rules Language statements. They are equivalent to variables in other programming languages and algebra, but with some additional features specifically for billing and rate analysis.

As in many programming languages, you can assign a value to an identifier by putting it on the left side of an ASSIGNMENT Statement (**Assignment Statement** on page 2-2 in the *Oracle Utilities Rules Language Reference Guide*). When you have assigned its value, you can use the identifier as an argument in other statements. These are called *simple identifiers*.

Several types of identifiers are unique to the Rules Language. Each of these has its own application and rules for use. For example, some identifiers have preassigned names that are automatically recognized by Oracle Utilities Billing Component and Oracle Utilities Rate Management. When you use those predefined identifiers as an argument in a statement, the programs automatically load the correct value for the current account and bill period, or for the current rate form.

There are also classes of identifiers that have special status in the system; specifically, Revenue and Bill Determinant identifiers. They are automatically eligible for reporting and saving.

The following sections contain detailed information about each of the special identifiers used in the Rules Language, including:

- **Revenue Identifiers**
- **Bill Determinant Identifiers**
- **Database Identifiers**
- **Interval Data Handles**
- **Time-of-Use Handles**
- **Factor Identifiers**
- **Override Identifiers**
- **Bill History Predefined Identifiers**
- **Other Predefined Identifiers**
- **Assignable Predefined Identifiers**
- **Record Identifiers (stem.component)**
- **Rate Schedule Environment Identifiers**
- **Report Options Identifiers**
- **Array Identifiers**

Note: The maximum character length for any type of identifier is 259 characters. Attempting to create or assign values with character lengths in excess of 259 characters will result in an error.

Indirect Identifiers

You can use "indirection" to reference an identifier. An "at" sign (@) before an identifier tells the program to reference the identifier whose name is the value of this identifier. For example:

```
Y = 1;
```

and

```
X = "Y";
@X = 1;
```

both set Y to 1.

Indirection is used in FOR EACH loops.

For example:

```
FOR EACH FCTCODE IN SET "KWH_CHARGE", "KW_CHARGE"
FULL_FCT_CODE = RS_OPCO_CODE + "_" + RS_JURIS_CODE + "_" + FCTCODE;
@FCTCODE = FACTOR[FULL_FCT_CODE].VAL;
END FOR;
```

This assigns values to the KWH_CHARGE and KW_CHARGE identifiers, based on the current rate schedule.

Note that the values for Billing Determinants are retrieved from the database *before* a rate schedule is run. Usually only the determinants used in the rate schedule are retrieved, so if you reference a determinant only through indirection it will not be loaded. To load all determinants for each account, check **Retrieve all Account Determinants** on the Billing tab in CIS Billing Options. See the *Oracle Utilities Billing Component User's Guide* for more information about setting CIS billing options.

It is an error to reference a nonexistent identifier through indirection. However, there is one exception: if the reference is alone on the right side of an assignment statement, the left side will remain unassigned.

If a nonexistent identifier is assigned a value through indirection, the identifier will be created.

Multiple Indirect Identifiers

You can use multiple @ signs to do several levels of indirection simultaneously. (However, it may be an error, if intermediate identifiers do not exist or have no value.) You can use multiple indirection to iterate through an array of saved names. For example:

```
/* This part is inside a loop */
/* Get a common name for processing a cut */
SAVE_NAME = BUS_LOCATION + "_" + HNDL.CHANNEL;
@SAVE_NAME = HNDL;
/* Process using @SAVE_NAME */
...
/* Save the SAVE_NAME */
I = I + 1;
X = "ID_" + I;
@X = SAVE_NAME;
...
/* Iterate through the saved handles to get the grand total */
FOR EACH J IN NUMBER I
X = "ID_" + J;
/* @X is the SAVE_NAME */
/* @@X is the handle */
TOTAL = TOTAL + INTDVALUE(@@X, "TOTAL");
END FOR;
```

A Note About Data Loading

The initial value of an identifier is 0 (zero), or "" in the case of a string (two double quotes together indicate a blank string value). If an identifier has not been assigned a value or is NULL and it is used in an expression, the expression uses the value zero or "", depending on the other type in the expression.

Revenue Identifiers

Revenue identifiers in the Oracle Utilities Rules Language are assigned to an account's charges for the bill period being processed. For example, you might assign the customer charge to the revenue identifier '$CUST_CHARGE', the demand charge to '$DEMAND_CHARGE', and a minimum charge to '$MIN_CHARGE'. The values assigned to Revenue identifiers are automatically printed in the bill reports, if the Billing Expert Print Detail Option is set to "Normal" or "All"; see the *Oracle Utilities Billing Component User's Guide*. There are two ways to create a revenue identifier:

- When you introduce the identifier in your rate form using an ASSIGNMENT Statement, use a dollar-sign ($) as its initial character. For example, `$CUST_CHARGE = 5.00;` This approach has the added value of making revenue identifiers "stand out" in the rate form when you or other users read it.
- Use a simple identifier in the ASSIGNMENT Statement. Then apply a REVENUE Statement (see **Revenue Statement** on page 5-10 in the *Oracle Utilities Rules Language Reference Guide*) to it. For example:

  ```
  BILL_KW = MAX(5,KW);
  REVENUE BILL_KW "Demand Charge";
  ```

  Using the REVENUE Statement allows you to add a descriptive label to appear in reports (in the first example, the charge would be labelled $CUST_CHARGE; in the second, Demand Charge). This can also be done using the LABEL Statement (page 5-6 in the *Oracle Utilities Rules Language Reference Guide*).

Oracle Utilities Billing Component bill reports automatically display the account's values for any revenue identifiers included in the rate form (this section appears at the bottom of the report). The revenue identifiers are listed in the report in the same order as in the rate form. This is important to remember when writing a rate form. If you wish the report to display the identifiers in a different order than they appear in the rate form, you can specify that order in a rider to the rate form. For instance, you could create a standard "revenue" rider that lists all of the revenue identifiers used by your utility, in the order that you want for the bill reports. You would then include that rider in all rate schedules via an INCLUDE Statement (see **Include Statement** on page 3-23 in the *Oracle Utilities Rules Language Reference Guide*).

$EFFECTIVE_REVENUE and the Bill Total

There must be one revenue identifier in a rate form that represents the total bill for the account. The predefined identifier $EFFECTIVE_REVENUE is included for this purpose. The system recognizes that the value assigned to this identifier is the account's bill total, and automatically prints the value in the account's bill report, no matter what Oracle Utilities Billing Component print detail option is in effect. If you prefer to use another identifier to represent the bill total, you must specify it to the system using the TOTAL keyword in a REVENUE Statement (see **Revenue Statement** on page 5-10 in the *Oracle Utilities Rules Language Reference Guide*). You must specify a TOTAL clause in every rate schedule for which you want an identifier other than $EFFECTIVE_REVENUE for the bill total.

Bill Determinant Identifiers

Bill determinants (also referred to as "billing determinants" or simply "determinants") are measures of an account's energy consumption for the bill period, and other values that are used to compute charges. Bill determinant values are assigned to bill determinant identifiers.

Bill determinant values are stored in the Bill History Table, the Bill History Value Table, and the Meter Value Table. The Bill History Table stores bill determinants that are used for the majority of your customers. The Bill History Value Table stores bill determinants used for a minority of customers (this is done to optimize system performance). The Meter Value Table stores bill determinant values for billing entities other than accounts; e.g., channels, channel groups, or (at some installations) CIS accounts.

> **Note:** When used in a rate form, it is an error to assign anything but a number (float) to a bill determinant identifier.

Bill determinants have their own set of identifiers, which your utility defines in the BILLDETERMINANT Lookup Table. When these identifiers are defined, you can take advantage of their special properties in the following ways:

- You can calculate a bill determinant value in the rate form (such as from interval data) and assign the results to one of the recognized bill determinant identifiers. You can then use SAVE statements to write that value to the Data Repository (the system automatically puts it in the appropriate table), and to a transaction record for your CIS.
- You can use a bill determinant identifier to get information from the database for processing in the rate form. If you use a bill determinant identifier that is stored in the Bill History Table or Bill History Value Table as an argument in a statement, the billing program automatically gives it the value for the current account for the current bill period. (You can also use any of the Historical Data functions to get bill determinant values for historical periods, and the Meter Value functions to get values from the Meter Value Table. See the *Oracle Utilities Rules Language Reference Manual*.)

  Note: Bill determinant values for periods before the current bill period are referred to in this manual as "historical determinants."

Database Identifiers

A Database identifier is used to retrieve a value from a specified field in a specified table in the Data Repository. You cannot assign values to Database identifiers; you can only use them in comparisons, or on the right side of an ASSIGNMENT Statement. The formats for database identifiers are:

```
table[key].column
```

and

```
table.column
```

TABLE is the name of the desired table in the Oracle Utilities Data Repository; COLUMN is the name of the desired column in that table. KEY is a string constant or identifier that specifies the key of the desired row in the table. The returned value is the contents of the column for the specified row.

The second format (without the key) is only for application to the ACCOUNT and CUSTOMER tables, because the programs automatically assume the key for the account or customer whose bill is currently being processed.

> **Note***:* The Rules Language Elements Editor displays only some of the Table.Columns for the Account and Customer tables when you select 'Database Identifiers', because those are used most often. However, you can apply this technique to any tables and columns in the database.

Appendix A*:* **Oracle Utilities Data Repository Database Schema** in the *Oracle Utilities Energy Information Platform Configuration Guide* provides a diagram of the entire Oracle Utilities Data Repository. Table names are denoted in all caps, and keys are underlined.

Note: If the key consists of multiple components, you must create a text string with each component separated by a comma, and assign the string to an identifier. You can then use the identifier as the key. See **String Expressions** on page 4-24 for more information. For example, to store the UOM for a particular channel in the Channel History Table, do the following:

```
CHHIST_ID = "1701,1,01/01/1997 02:00:00"
CH_UOM = CHANNELHISTORY[CHHIST_ID].UOMCODE
```

The following example uses database identifiers for a column in the Account Table. In this case, the key is not required. If you want to base a portion of a rate on the account's SIC, you could use the database identifier as follows:

```
IF ACCOUNT.SIC = "7900"
   THEN INCLUDE "7900_RIDER";
ELSE
   THEN INCLUDE "OTHER_RIDER";
END IF;
```

You can use Database identifiers as follows to determine whether a field in the database has a value (again, the second approach works only for the ACCOUNT or CUSTOMER tables)*:*

```
table[key].EXISTS
```

```
table.EXISTS
```

If the database has a value in the specified field, the identifier gets a value of 1: if not, 0.

For example, to process the account in a certain way if it has a value in the Meter Value Table, you could use the following two statements to set this up (in the example below, the dots [....] are placeholders to indicate that other statements would follow. Do not use the dots in the rate form).

```
mv_id = account.accountid + "," + readdate + "," + "kwh," + "1701,1";
if (meter value[mv_id].exists) then
....
/* process meter value record*/
```

You can also use the HASVALUE function (see **HASVALUE Function** on page 13-11 in the *Oracle Utilities Rules Language Reference Guide*) to accomplish the same thing.

> **Note:** For non-customer analysis, such as bill frequency revenue, all database identifiers with default keys have value 0 or, if strings, value "", and the HASVALUE function always returns zero for them.

Interval Data Handles

An interval data handle is a convention you use to recognize a reference to interval data. If the right side of the statement is a constant ('recorder,channel'), function, or expression that loads or computes an interval data cut, it is recommended that you assign the result to an "interval data handle." You could use the suffix _HNDL or the prefix CUT_; for example, KWH_HNDL, KVAR_HNDL, CUT_ R1234. This convention has no meaning to the applications, but Oracle Utilities strongly recommends that you use a consistent approach to identify interval data references. When you have assigned data to the handle, you can use the handle as an argument in another statement or function.

Time-of-Use Handles

As with the Interval Data Handles, if the result is a time-of-use cut (e.g., is the result of the **INTDCREATETOUPERIOD Function**), it's recommended that you use an easily recognized identifier, such as TOU_HNDL, as a reference to a time-of-use cut.

Factor Identifiers

Factor identifiers are database identifiers that enable you to get a specific and useful value: the value for the factor that was in effect at a given time. The actual value returned is based on the PRORATEMETHOD value (Prorate flag) in effect on the Effective Date, as follows:

- If the Prorate flag is NULL or 'N', the value in effect on the Bill Stop Date is used.
- If the Prorate flag is 'E', the value in effect on the Effective Date (or date in the factor code) is used.
- If the Prorate flag is 'Y', the factor value will be prorated based on the number of days in the billing period and the effective dates of the values.

The prorate period can be modified using the FACTOR_START_DATE and FACTOR_STOP_DATE identifiers. If they are set and the Prorate flag is not 'E', the value is either retrieved for the FACTOR_STOP_DATE ('N' or NULL), or is prorated between the two dates ('Y').

The Factor identifier formats are:

```
FACTOR["opcocode,juriscode,factorcode"].VALUE

FACTOR["factorcode"].VALUE

FACTOR[identifier].VALUE
```

Where:

- *opcocode,juriscode,factorcode* is the record key for the desired factor in the Factor Table. If this is a global factor (without operating company or jurisdiction) the code value must be "„factorcode". This full-key format is required only if the operating company and jurisdiction for the desired factor are different from that for the current rate form, or if they are or null.

  > **Note:** You can also use the ACCOUNTFACTOR function to determine whether a factor value was effect for the account on the last day of an historic bill period, as well as the current. The HASVALUE function can also be used with a factor to determine if it has a value.

- *factorcode* can be specified as defined in the Factor Table, if the desired factor belongs to the same operating company and jurisdiction as the rate form. For example, FACTOR("STATETAX").VALUE would return the value for the state tax that was in effect on the account's bill stop date. In addition, a specific effective date for the factor value may be specified as "...factorcode:<date>" where <date> is a date value in either of the usual date formats. When a date is supplied it is used as the effective date in retrieving the factor.

  > **Note:** Because the factor codes have a data type of varchar (which means that they are text strings), you must enclose them in double quotes. You can pick a factor code by opening the Rules Language Elements Editor, then selecting **Factor Codes** under **Element Types**.

- *identifier* can be used if you've assigned the factor code or factor key to an identifier earlier in the rate form. This is useful when building a factor code on the fly based on account information. For example

  ```
  FACTOR_NAME = "KWH_CHARGE"

  $KWH_CHARGE = FACTOR[FACTOR_NAME].VALUE
  ```

  In this example, the first line assigns the identifier 'FACTOR_NAME' to "KWH_CHARGE". The second line assigns the revenue identifier '$KWH_CHARGE' to the "KWH_CHARGE" value in the Factor Value Table.

Override Identifiers

Overrides keep track of special events that occur at the account level, such as interruptions, curtailments, or special services.

Override identifiers are a special type of database identifier. They enable you to get values stored in the Override History Table (ACCTOVERRIDEHIST). There are two override identifiers. One retrieves the override's float value (VALUE), and the other retrieves its string value (STRVAL). "VALUE" is a number that represents the override's magnitude, and is used when creating an override mask (see the **INTDCREATEOVERRIDEDAYMASK Function** on page 9-16 in the *Oracle Utilities Rules Language Reference Guide*). "STRVAL" is utility specific: it can be a note or a qualifier. The override whose values are returned when you use one of these identifiers in a rate form is the most recent entry in the Override History Table (ACCTOVERRIDEHISTORY) that belongs to the current account, with the specified override code, and that overlaps the bill stop date. If there is no such override record in the Override History Table, VALUE and STRVAL are both set to NULL. This can be changed so that they are set to 0 and "" respectively, using the configuration file keyword OVERRIDE_NULL_VAL_IS_ZERO (see **Chapter 2: Configuration Files** in the *Oracle Utilities Energy Information Platform Configuration Guide*).

The formats for Override identifiers are:

```
OVERRIDE["override_code"].VALUE

OVERRIDE["account_id:override_code"].VALUE

OVERRIDE[identifier].VALUE

OVERRIDE["override_code"].STRVAL

OVERRIDE["account_id:override_code"].STRVAL

OVERRIDE[identifier].STRVAL
```

Where:

- *override_code* is the override's code, as defined in the Override Lookup Table. For example, OVERRIDE("INTERRUPT").VALUE would return the value to be used when creating an interval data mask for the INTERRUPT override.

  > **Note:** Because the override codes have a data type of varchar (which means that they are text strings), you must enclose them in double quotes. You can pick an override code by opening the Rules Language Elements Editor and selecting **Override Codes** under **Element Types**.

- *account_id* can be used if you've assigned the override code to one account in the database but wish to reference that override for a different account. This allows you to set up "global" override codes associated to a default account and reference them when processing any account.

- *identifier* can be used if you've assigned the override_code (or the combination of an account_id and override_code) to an identifier earlier in the rate form. This is used in building an override code on the fly based on account information. For example:

  ```
  OVERRIDE_NAME = "INTERRUPT_CHARGE"

  INTERRUPTION_CHARGE = OVERRIDE[OVERRIDE_NAME].VALUE
  ```

  In the example, the first line assigns the identifier 'OVERRIDE_NAME' to "INTERRUPT_CHARGE". The second line assigns the identifier 'INTERRUPTION_CHARGE' to the "INTERRUPT_CHARGE" value in the Override History Table.

Overrides Applied to Channels, Channel Groups

Overrides can keep track of special events that occur at levels other than the account level, such as a meter level; for example, interruptions, curtailments, or special services.

These override records are stored in the ACCTNAMEOVERRIDEHIST Table. You can get these values using the identifier format:

```
OVERRIDE["override_code,name"].VALUE
```

```
OVERRIDE["override_code,name"].STRVAL
```

Where:

- *override_code* is the override's code, as defined in the Override Lookup Table.
- *name* is the name of the channel, channel group, or CIS account specified in the Name column in the ACCTNAMEOVVERIDEHIST Table.

Bill History Predefined Identifiers

Bill History Predefined identifiers are used on the right side of ASSIGNMENT statements, or in any other statement or function, and contain values from the Bill History Table for the current account for the current bill period. The five predefined Bill History identifiers are:

Identifier	Description
BILL_PERIOD	The first day in the bill month.
BILL_START	The first day (with time) of the account bill period.
BILL_STOP	The last day (with time) of the account bill period.
READ_DATE	The READDATE in the Bill History record. If Null, defaults to BILL_STOP.
NUMDAYS	The number of days in the bill period (BILL_STOP - BILL_START, rounded to the nearest day).

To get historical values for any of these identifiers except NUMDAYS, apply the HISTVALUE function (see **HISTVALUE Function** on page 13-51 in the *Oracle Utilities Rules Language Reference Guide*) to them.

About Other Values from the Bill History or Bill History Value Tables: To get non-bill determinant values from the Bill History Table or the Bill History Values Table, use the following convention:

```
BILLHISTORY[key].column
```

Where:

- *column* is the column name in either the Bill History Table or the Bill History Value Table
- *key* is the record key for the desired row in the desired table. You supply a key only when you want to get a value for an account other than the one whose bill is currently being computed.

Other Predefined Identifiers

The following identifiers are automatically assigned the appropriate values by the analysis programs. You can use these values on the right side of an ASSIGNMENT Statement, and in any other statement type or function.

Identifier	Description
AUXILIARY_DEMAND	Set as a result of the MAXKW function to indicate which kW was chosen.
CURRENT_DATE	The current system (i.e. today's) date.
RATE_CODE	For Oracle Utilities Billing Component, the link to the rate schedule that the account is on. For Oracle Utilities Rate Management, the link to the rate schedule that the account is on, unless redefined by Interschedule Mapping (see the *Oracle Utilities Rate Management User's Guide*).
RATE_SCHEDULE_CODE	The code of the rate schedule being executed.
REPORT_GUID	The Global Unique ID (GUID) of all Rules Language Reports (Report Type: LSRate) and Billing Component billing processes (Report Type: PLBX) executed via the Report Framework. This includes Automatic Billing, Approval Required Billing, Current/Final Bill, and Bill Correction when executed from the web user interface.
RS_EFFECTIVE_START	In billing applications, this is either the BILL_START or the rate schedule's effective start (as defined in the Rate Code History Table for the account), whichever is later. In rate analysis applications, this is the same as BILL_START.
RS_EFFECTIVE_STOP	In billing applications, this is either the BILL_STOP or the rate schedule's effective stop (as defined in the Rate Code History Table for the account), whichever is earlier. In rate analysis applications, this is the same as BILL_STOP.
RS_JURIS_CODE	The rate schedule's (not rider's) jurisdiction code.
RS_OPCO_CODE	The rate schedule's (not rider's) operating company code.

Assignable Predefined Identifiers

The system automatically assigns default values to the following identifiers. However, you can override these defaults by supplying a different value for them using an ASSIGNMENT Statement. Each of these may appear only once on the left of an ASSIGNMENT Statement in a rate schedule, and must be assigned a constant. The value is assigned at compile time, and may be used before analysis begins.

Identifier	Description
BILL_PERIOD_SELECT	Determines how a season is assigned to a bill period. Its default value of 0 means the system checks the BILL_STOP date against the season dates. If set to 1, the system uses the BILL_START date. If set to 2, the system uses the BILL_PERIOD (bill month) date. If set to 3, the system uses the Scheduled Read Date. If set to 4, the system uses the Governing Date. For example, to specify to use the season period that contains the start date of the bill period, put the following ASSIGNMENT Statement in your schedule: BILL_PERIOD_SELECT = 1;
BILL_TYPE	This identifier can have the following values: TRIAL, CANCEL/REBILL, ADJUSTMENT, CANCEL, REBILL, CURRENT, or FINAL. This value is set by the system when the account's bill is processed. It is evaluated by the Rules Language (in the required "Cancel" rider) to determine what transaction records the program writes for the bill. See **Cancel/ Rebill Rider** on page 1-8 for more information.
HOURS_PER_MONTH	The default value is 730. To apply a different value, use an ASSIGNMENT Statement. Specifically, set HOURS_PER_MONTH equal to either a desired constant value, or to the results of the BILLINGHOURS or MONTHHOURS functions (see **BILLINGHOURS Function** on page 13-21 and **MONTHHOURS Function** on page 13-35, respectively, in the *Oracle Utilities Rules Language Reference Guide*). For example, to specify the actual number of hours in the current billing period for the account, include the following ASSIGNMENT Statement in your schedule: HOURS_PER_MONTH = BILLINGHOURS();
INTD_ERROR_STOP	If 1, stop on interval data error; if 0, don't stop. If you don't specify a value for INTD_ERROR_STOP in the schedule, the program uses the default, which is the setting specified by **Tools-›Options-›Error Handling**. See the *Data Manager User's Guide* for information about Options.
REBILL_REASON	The Rebill Reason Code for the selected Rebill Reason when processing CANCEL/REBILL or REBILL bill corrections.

Identifier	Description
REBILL_REASON_NAME	The Rebill Reason Name for the selected Rebill Reason when processing CANCEL/REBILL or REBILL bill corrections.
SEASON_SCHEDULE_NAME	The name of the season schedule to use when determining seasons. The default is the value specified by **Tools›Options›Rate Analysis**. (See the *Data Manager User's Guide* for information about Options.)

Reserved Identifiers

Reserved identifiers are identifiers that have a specific use, and cannot be used for any other purpose. These identifiers are used to trigger specific operations or functions within the rate schedule in which they appear.

Identifier	Description
LS_SAVE_PROFILE_FILENAME	Specifies the path and file name of a text file that contains the Rules Language Profile of the rate schedule in which this identifier appears. See **Rules Language Profiling** on page 14-2 in the *Oracle Utilities Energy Information Platform Configuration Guide* for more information about using this identifier. **NOTE**: This identifier should only be used when troubleshooting Rules Language performance issues or other problems, as creating the code profile will have a negative impact on performance.
COM_ERROR_STOP	Specifies error handling behavior in the event of an error in a COM method invoked by the Rules Language. Available settings include: 0 - Use the STOP_ON_COM_ERROR configuration parameter. See **LODESTAR.CFG** on page 2-2 in the *Oracle Utilities Energy Information Platform Configuration Guide* for more information about this parameter. 1 - Ignore the error and set value of LASTCOMERROR identifier. 2 - Stop Rules Language processing.
LASTCOMERROR	Error code for the most recent COM error. Automatically populated when COM_ERROR_STOP is set to 1, or when the STOP_ON_COM_ERROR configuration parameter is not present in the LODESTAR.CFG file. Can be used with the IF THEN statement to perform specific processing in the event of a COM error.
LASTCOMERRORTEXT	Error description for the most recent COM error. Automatically populated when the LASTCOMERROR identifier is populated.

Record Identifiers (stem.component)

Some identifiers and functions return a record containing a number of values, rather than a single value. For example, when you apply one of the INTDLOADxxx functions (see **Chapter 6**: **Rules Language Functions Overview**) to load an interval data cut, the analysis programs automatically calculate summary values about the cut, such as the total energy in the cut or the average of all non-missing interval values in the cut. When you apply the FOR EACH x in LIST Statement (see **For Each x In List Statement** on page 3-10 in the *Oracle Utilities Rules Language Reference Guide*), the program gets the entire record from the Data Repository Table for each item in the list.

In all cases, the program stores these records in memory while the rate form processes. That enables you can report or apply calculations to any of the individual values in the temporary record. You specify these values using the following identifier convention:

```
stem.component
```

Where:

- *stem* is the name of the record. You assigned this name in the function or statement that loaded or created the record. For example, in the INTDLOAD function (see **INTDLOAD Functions** on page 9-29 in the *Oracle Utilities Rules Language Reference Guide*), it is the identifier you assigned on the left side of the equal sign in the ASSIGNMENT Statement. In the FOR EACH x statements, it is the identifier that you supplied for the x.
- *component* is the name of the field in the temporary record. If the temporary record is a row from a Data Repository (as in the FOR EACH x statements), it is the name of the column in the database. If the record is computed (as in the INTDLOAD functions), it is an assigned name. To find column names, select **Browse-›Database Schema** from the Data Manager File Menu. The assigned names are listed in the description of the INTDLOAD function (see **INTDLOAD Functions** on page 9-29 in the *Oracle Utilities Rules Language Reference Guide*).

For example, the following statements would list the total energy from an interval data cut (based on recorder,channel '1700,1') on a Billing Report:

```
INT_HNDL = INTDLOAD('1700,1');
TOTAL_ENERGY = INT_NHDL.ENERGY;
LABEL TOTAL_ENERGY "Total Energy";
```

Saving Database Records

You can also use the stem.component convention to save values to the Data Repository tables. The following example illustrates use of the stem.component convention to save values to the Meter Value Table (specifically, the BILLDETERMCODE and VAL columns in the Meter Value Table). The energy value that the system computed for the interval data cut referred to by the handle INT_HNDL will be stored in the VAL column, and the bill determinant code "1" will be stored in the BILLDETERMCODE column:

```
/* Save each kWh to the Meter Value record */
MV.VAL = INT_HNDL.ENERGY:
MV.BILLDETERMCODE = "1";
SAVE MV TO TABLE METERVALUE;
```

Assigning One Stem to Another

You can also assign one stem and its corresponding components to another stem. For example, suppose you had the following stem and components identified in a rate schedule:

```
BH.VALUE1 = "VAL1";
BH.VALUE2 = "VAL2";
```

To assign this stem and its components to another stem, you simply assign the old stem to the new stem as follows:

```
AH = BH;
```

The "AH" stem would now include the "VALUE1" and "VALUE2" components, along with any other components assigned to it.

Writing Temporary Values to Interval Data Handles

You can also use the stem.component convention to write temporary values to an interval data handle in memory. In this case the "stem" is the handle (identifier) assigned to the handle earlier in the rate form, and the "component" is one of the following:

Component	Description
RECORDER	Sets the recorder ID in the handle
CHANNEL	Sets the channel number in the handle
DESCRIPTOR	Sets the 80-character description in the handle
UOM	Sets the unit-of-measure code for the handle
STATUSCODE	Sets the status code of each non-missing interval to the specified value.

For example, to set the descriptor for an interval data handle ('SCALED_HNDL') to "Hourly KWH", you could do the following:

```
KWH_HNDL = INTDLOADUOM("01");
SCALED_HNDL = INTDSCALE(KWH_HNDL,"HOUR","TOTAL");
SCALED_HNDL.DESCRIPTOR = "HOURLY KWH";
SAVE SCALED_HNDL TO CHANNEL '1701,1';
```

> **Note**: if you want to save the values to the database, you must apply a SAVE Statement (see **Save Statements** on page 6-3 in the *Oracle Utilities Rules Language Reference Guide*).

Rate Schedule Environment Identifiers

Many billing options affect the bill calculation before a rate schedule is used. For example, the Effective Date options help determine which rate schedule to use, as well as how many rate schedules to run. The Check options (see **Defining Default Billing Options** in the *Oracle Utilities Billing Component User's Guide*) help determine whether to even attempt to bill an account, long before their account's rate schedule is loaded.

This section addresses the options that affect bill calculation *as* the rate schedule is loaded and *after* the rate schedule is loaded.

The code that runs rate schedules is called the Rate Schedule Environment. Its parameters are determined by settings in the configuration file (see **Chapter 2: Configuration Files** in the *Oracle Utilities Energy Information Platform Configuration Guide*), user and global options, and by command line parameters and user selections set before running the rate schedule. Some of these parameters can be accessed and set from within the rate schedule itself (and its included riders and contracts). Note that these only apply if the rate schedule is loaded and run; they do not apply to actions taken before the rate schedule is loaded.

Using LSRSENV Identifiers

To access a parameter, use an identifier of the form **LSRSENV.attribute**. All such identifiers can be "read," but only some can be assigned. When one of these identifiers is read, the actual parameter value is retrieved and stored in the identifier. When one is set, its value is set first, then the corresponding parameter is set.

To assign an integer to an LSRSENV identifier, assign 1 to turn it on. Assign 0 to turn if off. Any other value is an error. For example, to turn on the ALLOWINTDREASSIGN parameter:

```
LSRSENV.ALLOWINTDREASSIGN = 1
```

Available LSRSENV Identifiers

The available LSRSENV identifiers and their meanings are:

LSRSENV.ACCTNOTE_SAVE_ERRORS: If on, Saves are enabled, and if there is an error while running the rate schedule, the error message will be saved in the ACCOUNTNOTE Table. This applies to all billing modes. If this is not set on, the account note will be written according to the billing rules for Automatic and Approval Required Billing. If this would be set on but an error occurs before running the rate schedule (preventing the rate schedule from running), this setting has no effect.

```
0 - Do not force saves of errors in ACCOUNTNOTE table.
1 - Force saves of errors in ACCOUNTNOTE table.
```

LSRSENV.ACCTNOTE_SAVE_WARNINGS: If on, Saves are enabled, and if there is a warning while running the rate schedule, all such warning messages may be saved in the ACCOUNTNOTE table. They will be saved if 1) there was also an error; 2) the billing mode is Automatic Billing; or 3) one of the Automatic Commit save modes is selected. If this is not set on, the account note will be written according to the billing rules for Automatic Billing. If this would be set on but a warning occurs before running (preventing the rate schedule from running), this setting has no effect.

```
0 - Do not force saves of warnings in ACCOUNTNOTE table.
1 - Force saves of warnings in ACCOUNTNOTE table.
```

LSRSENV.ALLOWINTDREASSIGN: Based on configuration file parameter ALLOWINTDREASSIGN. Has an integer value. Can be set.

```
0 - Flag is off or not set.
1 - Flag is set on.
```

LSRSENV.CISFORMT_FILENAME: The string that this is set to will be used as the CIS control (CISFORMT.TXT) file for the currently executing rate schedule. This setting overrides any other settings for this file.

LSRSENV.COMMIT: Based on user selection or Save command line parameter. Has an integer value. Cannot be set.

```
0 - No save or no auto commit.
1 - Database commit after run if all OK.
```

LSRSENV.DIRECT_WRITE_CIS: Based on configuration file parameter DIRECT_WRITE_CIS. Has an integer value. Can be set.

```
0 - Flag is off or not set.
1 - Flag is set on.
```

LSRSENV.ESTIMATE: Default is on for Oracle Utilities Rate Management, off for Oracle Utilities Billing Component. Has an integer value. Can be set.

```
0 - Do not estimate
1 - Estimate
```

LSRSENV.FACTOR_VALUE_NOTYPE: Enables the user to use a Factor identifier (i.e. FACTOR[key].VALUE) without having to call the FACTORINEFFECT function to determine if the factor exists, and thereby improving performance. If needed, the HASVALUE() function can be used to test the return value. See **Factor Identifiers** on page 4-7 for more information about using Factor identifiers.

```
0 - Not set
1 - FACTOR[key].VALUE identifiers will return a Null value if the
factor does not exist
```

LSRSENV.INIT_TYPE: Equals a constant that indicates the environment the analysis is running in. Values are:

```
0 - Run from a command line executable
1 - Run from a client/server application
2 - Run from a web application
3 - Run as a COM object
```

LSRSENV.INVOICENAME: The string that this is set to will be used as the INVOICENAME (from the Invoice Number table) for invoices generated by Oracle Utilities Billing Component instead of the default specified in the LODESTAR.CFG file. See **Invoice Numbering** in **Chapter 4**: **Billing Rules and Definitions** in the *Oracle Utilities Billing Component Installation and Configuration Guide, Volume 1* for more information about Invoice Numbering.

LSRSENV.LINE_NUMBER: Equals the line number of the statement it is in, in the compiled rate schedule. Cannot be set.

LSRSENV.LISTVALUE_NOTYPE: If on and the LISTVALUE function retrieves no elements, LISTVALUE returns a NULL value (when assigned to an identifier the HASVALUE function will return zero). If off (default) and the LISTVALUE function retrieves no elements, LISTVALUE returns an empty string. This applies to all billing modes.

```
0 - Return empty string if LISTVALUE function retrieve no elements.
1 - Return no value if LISTVALUE function retrieve no elements.
```

LSRSENV.MISSING_INDIRECT_ID_IS_NOTYPE: It is usually an error to reference (for example, in an expression) an indirect identifier that does not exist. If this flag is on and an indirect identifier does not exist, its return value will be set to "none". This enables it to be used in expressions.

```
0 - Flag is off or not set.
1 - Flag is set on.
```

LSRSENV.NO_INTERVAL_DATA: Set to 1 for the Bill Frequency Revenue, Typical Bill, and Crossing Point Oracle Utilities Rate Management Analyses, and if the Do Not Use Interval Data option is set on the Check Options tab of the Default Billing Options dialog. Has an integer value. Can be set.

```
0 - Interval data can be used
1 - Interval data cannot be used
```

LSRSENV.NO_TEMP_SAVE: Based on configuration file parameter NO_TEMP_SAVE. Set to not try to save records temporarily. Only used if saves are off. Default is to try to save and report errors. Has an integer value. Can be set.

```
0 - Try to save records even if save is not enabled.
1 - Do not try to save records if save is not enabled.
```

LSRSENV.OVERRIDE_NULL_VAL_IS_ZERO: Based on configuration file parameter OVERRIDE_NULL_VAL_IS_ZERO. Has an integer value. Can be set.

```
0 - Flag is off or not set - Account OVERRIDE value of NULL stays NULL.
1 - Flag is set on - Account OVERRIDE value of NULL is set to zero.
```

LSRSENV.SAVE: Based on user selection or Save command line parameter. Has an integer value. Cannot be set.

```
0 - No save allowed (RateExpert default)
1 - Saves enabled
```

Report Options Identifiers

Report Options are normally set by a user when running an analysis or creating a report. Report Options Identifiers (LSRPTOPTS) allow some of these options to set within a rate form.

Using LSRPTOPTS Identifiers

To access a parameter, use an identifier of the form LSRPTOPTS.attribute. All such identifiers can be read or assigned. When one of these is read the actual parameter value is retrieved and stored in the identifier. When one is set, its value is set first, then the corresponding parameter is set.

To assign an integer to an LSRPTOPTS identifier, assign 1 to turn it on. Assign zero to turn if off. Any other value will be an error. For example, to turn on the REPORTS_ONLY parameter:

```
LSRPTOPTS.REPORTS_ONLY = 1
```

Available LSRPTOPTS Identifiers

The available LSPRTOPTS identifiers and their meanings are:

LSRPTOPTS.TITLE: The string that this is set to will be used as the page title, replacing the one automatically generated. If set to "" automatic generation will be re-enabled.

LSRPTOPTS.OPTIONAL_TITLE_1: The string that this is set to will be used as the first optional title, under the page title. If set to “” there will be no optional title 1. This corresponds to the Optional Title 1 in Report Options.

LSRPTOPTS.OPTIONAL_TITLE_2: The string that this is set to will be used as the second optional title, under the page title. If set to “” there will be no optional title 2. This matches the Optional Title 2 in Report Options.

LSRPTOPTS.OPTIONAL_TITLE_3: The string that this is set to will be used as the third optional title, under the page title. If set to “” there will be no optional title 3. This corresponds to the Optional Title 3 in Report Options.

LSRPTOPTS.BILLING_UNITS_LABEL: The string that this is set to will be used as the Label for the Billing Units column in the Bill Calculation Results section of a bill report. If set to blank “” there will be no Billing Units column in the report. Used only with billing analyses.

LSRPTOPTS.DISTRIBUTION_LABEL: The string that this is set to will be used as the Label for the Distribution column in the Bill Calculation Results section of a bill report. If set to blank “” there will be no Distribution column in the report. Used only with billing analyses.

LSRPTOPTS.CHARGE_RATE_LABEL: The string that this is set to will be used as the Label for the Charge Rate column in the Bill Calculation Results section of a bill report. If set to blank “” there will be no Charge Rate column in the report. Used only with billing analyses.

LSRPTOPTS.REVENUE_LABEL: The string that this is set to will be used as the Label for the Revenue column in the Bill Calculation Results section of a bill report. If set to blank “” Revenue will be label for revenue column in the report. Used only with billing analyses.

LSRPTOPTS.REPORTS_ONLY: If on then turns on SKIP_PAGE_1 and turns off the display of everything except the title line and lines generated by using the REPORT statement.

```
0 - Display usual data.
1 - Display only the results of REPORT statements.
```

LSRPTOPTS.SKIP_PAGE_1: - If on, skips creating the first page on an analysis that contains its input values.

```
0 - Display inputs as page 1.
1 - Do not display inputs as page 1.
```

LSRPTOPTS.WRITE_TO_FILE: - The generated report will be written to the file whose name is the string that this is set to. This must include a fully qualified path and file name.

Array Identifiers

Array identifiers are used to store series of data elements of the same data type. For example, you could create an array of integers, an array of characters, or an array of interval data references. Array identifiers store single-dimension arrays only.

Using Array Identifiers

Array identifiers use the following format:

#<identifier>[<index>]

where:

- <identifier> is the name of the array identifier.
- <index> is the index for the array identifier.

Example:

```
#INTD_FILE[1] = INTDOPEN("bxayv6.lse");
```

Rules for Creating Array Identifiers

The rules for creating array identifiers are as follows:

- Array identifiers must begin with a pound sign("#")
- The array index must be between open and close brackets.
- The array index must evaluate to an integer type.
- The array index is 1-based.
- Valid index values are 1 to 2147483647 when setting an array element, although the upper bound will probably be much smaller because of the memory needed. When retrieving an array element, valid index values are 1 to the highest index value used when setting the array elements.
- The only time an array index is not required is when clearing the whole array (through use of the **Clear Statement**).
- Array identifiers should be released as soon as they are no longer needed in the rate schedule. This frees memory and can improve performance.
- When running a rate schedule containing an array identifier using the Trial Calculation function, the right hand pane does not show every value in the array, but shows the last referenced value for that array identifier.
- When retrieving interval data handles from an array identifier, always use a different identifier name than the one used when loading.
- When retrieving an array element, if the index is not an integer type or the index is out of bounds, the user will get one of the appropriate run time errors, "Array index is not an INTEGER type" or "Array index value is out of range". There is a Rate Schedule Environment variable "LSRSENV.MISSING_ARRAY_ID_OR_INDEX_IS_NOTYPE" that can be set so that instead of receiving the above error message a NULL value will be returned instead. The **ARRAYUPPERBOUND Function** on page 13-3 of the *Oracle Utilities Rules Language Reference Guide* can be used to return the upper bound of an array identifier.
- If a rate schedule uses CLEAR on an element in an array, Clear frees up memory for that element. Therefore, after CLEAR #my_array[93], HASVALUE(#my_array[93]) will return 0. If a rate schedule uses CLEAR on the array variable, CLEAR frees all memory associated with the array and allows that variable to be used again as an array or non-array variable.

- The data elements in arrays can have the following data types: "NONE", "FLOAT", "STRING", "INTEGER", "DATE", "INTDATA HANDLE", "TOU HANDLE", "TABLE VALUES" and "DATABASE VALUES".
- An uninitialized legal index element will have the type "NOTYPE" or "NONE" such that HASVALUE(#my_array[X]) of uninitialized element X returns 0.
- It is a run-time error to use an identifier as an array identifier and a non-array identifier while running a rate schedule without first using the CLEAR statement. For example, you cannot use #ASD[3] = 3; and ASD = 1; without a CLEAR ASD[]; before the second assignment. For another example, you cannot use ASD = 1; and #ASD[3] = 1; without a CLEAR ASD; before the second assignment. The error message for the preceding error is "Can not use the same identifier name to represent an array identifier and non-array identifier at the same time."

Examples

```
// Create an array of integers
FOR EACH X IN NUMBER 10
    #ASD[X] = X + 3;
    #ASD[X +11] = X + 1;
END FOR;
Y = #ASD[1];
Z = #ASD[3];
CLEAR #ASD[];

// Create an array of Interval Data Handles
#INTD_FILE[1] = INTDOPEN("bxayv6.lse");
#INTD_FILE[2] = INTDOPEN("spring.lse");
#INTD_FILE[3] = INTDOPEN("fmayv154.lse");
WV_COUNT = INTDRECCOUNT(#INTD_FILE[2]);
FOR EACH I IN NUMBER WV_COUNT
    #HNDL[1] = INTDREADFIRST(#INTD_FILE[1]);
    #HNDL[2] = INTDREADFIRST(#INTD_FILE[2]);
    HNDL3 = INTDREADFIRST(#INTD_FILE[3]);
    CLEAR #HNDL[1];
    CLEAR #HNDL[2];
    CLEAR HNDL3;
    IF I  =  1
    THEN
        #HNDL[1] = INTDREADFIRST(#INTD_FILE[1]);
        #HNDL[2] = INTDREADFIRST(#INTD_FILE[2]);
        HNDL3 = INTDREADFIRST(#INTD_FILE[3]);
    ELSE
        #HNDL[1] = INTDREADNEXT(#INTD_FILE[1]);
        #HNDL[2] = INTDREADNEXT(#INTD_FILE[2]);
        HNDL3 = INTDREADNEXT(#INTD_FILE[3]);
    END IF;
    CURRENT_RECORDER3 = HNDL3.CUSTID;
    CURRENT_RECORDER = #HNDL[2].CUSTID;
    CURRENT_CHANNEL3 = HNDL3.CHANNEL;
    CURRENT_CHANNEL = #HNDL[2].CHANNEL;
END FOR;
//Load 10k Interval Data Handles
FOR EACH REC IN LIST CHAN1_ALL
    NUM_HNDLS = NUM_HNDLS + 1;
    HNDL_NAME = REC.RECORDERID + ",1";
    #HNDL[NUM_HNDLS] = INTDLOADDATES(HNDL_NAME , '05/01/1993 00:00:00'
    , '05/31/1993 23:59:59');
    $GET_FIRST_TOTAL = GET_FIRST_TOTAL + #HNDL [ NUM_HNDLS ]. TOTAL;
END FOR;

//Retrieve 10 Interval Data Handles
```

```
FOR EACH X IN NUMBER NUM_HNDLS
   MYHNDL = #HNDL[X];
   GET_TOTAL = GET_TOTAL + MYHNDL.TOTAL;
END FOR;
CLEAR HNDL, NUM_HNDLS,MYHNDL;
$EFFECTIVE_REVENUE = GET_TOTAL;
```

Indirect Array Identifiers

You can use "indirection" to reference an array identifier. Array identifiers have the following syntax:

```
#<identifier>[<index>]
```

Indirect array identifiers will have a similar syntax, except that the identifier is replaced with an indirect identifier:

```
#@<identifier>[<index>]
```

For example:

```
#Y[1] = 1;
```

and

```
X = "Y";
#@X[1] = 1;
```

will both set the first element of Y to 1.

For more information about indirect identifiers, see **Indirect Identifiers** on page 4-3.

You can also indirectly reference an array of stem.tail identifiers. You would use the following stem.tail syntax to reference an array directly:

```
#<stem_identifier>[<index>].<tail>
```

You would reference the array indirectly using the normal array syntax used above:

```
#@<identifier>[<index>]
```

but the indirect identifier points to the <stem_identifier>.<tail>

For example:

```
#Y[1].A = 1;
```

and

```
X = "Y.A";
#@X[1] = 1;
```

will both set Y[1].A to 1.

For more information about stem.tail identifiers, see **Record Identifiers (stem.component)** on page 4-14.

Constants

A constant is a value that doesn't change. The Rules Language supports the following types of constants:

- **Numbers:** Numbers can be represented as integers or decimal numbers. Values for charges can begin with a dollar sign ($) for easier reading.
- **Text Strings:** A string constant is any set of characters (except a double quote) surrounded by double quotes.
- **Dates:** Date are represented as either 'mm/dd/yyyy' or 'yyyy/mm/dd' (International format). These are the only two date formats supported by the Rules Language. Date constants can also include a time: 'mm/dd/yyyy hh:mm:ss'. If you do not include a time, midnight (00:00:00) is assumed. A date constant can be used as a function parameter, or in a logical comparison.

  All date to string conversions are converted into International format when read by the Rules Language by default unless otherwise specified (using the **DATETIMETOSTRING Function**).
- **Recorder, channel:** A specific recorder and channel are indicated by 'recorder,channel' (with no space before or after the comma). The recorder is any combination of uppercase letters and digits, and the channel is any integer (0-9). When the constant on the right side of the equal sign is 'recorder,channel', the interval data for the current bill period is loaded and a reference to it is assigned to the identifier. The identifier is then an "interval data identifier." (If data outside the current bill period is needed, use the **INTDLOAD Function** on page 9-34 in the *Oracle Utilities Rules Language Reference Guide*.)

  Note: You can use 'recorder,channel' in an expression in the same way you would use an identifier that has been assigned to an interval data cut, as described in Operator Rules in the next section.

  Note: You can assign a channel number between 10 and 99 by using the CHANNEL10 configuration file keyword (see **Chapter 2: Configuration Files** in the *Oracle Utilities Energy Information Platform Configuration Guide*).

Expressions

Expressions within statements describe an operation to be performed between variables and constants (variable/variable, variable/constant, or constant/constant) The Rules Language supports the following types of expressions:

- **String Expressions**
- **Date Expressions**
- **Arithmetic Expressions**

String Expressions

String expressions describe operations between strings. The only operator applicable to string expressions is '+' (plus), which tells the application to concatenate the right side of the '+' to the end of the left side. For example:

```
S1 = "This is ";
S2 = S1 + "a " + "test.";
```

Then S2 contains "This is a test."

The expression on the right side of the equal sign can include a string, number, date, or 'recorder,channel' identifier. Integer number conversion is used to convert numbers to strings (floating point numbers are truncated). Date/times are converted to "mm/dd/yyyy", with "hh:mm:ss" appended if the time is not midnight. The string value of a 'recorder,channel' is "recorder,channel".

For example:

```
S1 = "Today's date is ";
S2 = CURRENT_DATE + ".";
LABEL S1 + S2;
```

If today was January 1, 1999, the following line would appear in the report:

```
Today's date is 01/01/1999.
```

String expressions are often used to create names for items to be written to or retrieved from the database. The following example of retrieves the factor value for the Energy Charge:

```
FACTOR_NAME = RS_OPCO_CODE + "," + RS_JURIS_CODE + ",KWHCHG";
$ENERGY_CHG = KWH * FACTOR[FACTOR_NAME].VALUE;
```

If the operating company code was "GPCO" and the jurisdiction was "MN", the identifier FACTOR_NAME would contain "GPCO,MN,KWHCHG". This is a global energy charge factor for the operating company, jurisdiction GPCO, MN.

To include a double quote in a string, use two double quotes.

That is, the string:

```
"version=""1.0"""
```

would return the string:

```
version="1.0"
```

Date Expressions

Date expressions describe operations performed on date identifiers. The two types of operations that can be used with dates are addition and subtraction. A date identifier or constant must be on the left side of the operator. The right side may be a number or a time constant. If a number, it specifies a number of seconds (a date is stored as the number of seconds since January 1, 1970). If it is a time constant, it must be in one of these forms:

Time Constant	Description
'hh:mm:ss'	Hours, minutes, and seconds
'hh:mm'	Hours and minutes
'*n* DAYS'	*n* is a number specifying the number of days
'*n* WEEKS'	*n* is a number specifying the number of weeks.

Addition

Addition can only be used to add a number or constant to a date identifier or constant. For example, to get the date for the day one week in the future from the current date, you could use the following:

```
NEXT_WEEK_DATE = CURRENT_DATE + '7 DAYS'
```

Subtraction

Subtraction can be used between a date identifier and a number, a constant, or another date identifier. For example, to get the date for the day 2 weeks before the current date, you could use the following:

```
LAST_WEEK_DATE = CURRENT_DATE - '2 WEEKS'
```

Two dates can be subtracted—the result is the number of seconds between them. Dates can be compared, but they cannot be added, multiplied, or divided together.

Arithmetic Expressions

Arithmetic expressions describe mathematical operations between variables and constants. The Rules Language supports the standard arithmetic expressions. The following are guidelines for using operators, operation precedence, and operator rules.

Operators

You use the following operators to combine constants and identifiers into expressions:

Assignment Operators	Description
=	Assigns the value(s) on the right side of the equal sign to the identifier on the left side.
=+	Positive assignment; used when a negative value is unacceptable. Assigns the results of the expression on the right side of the equal sign to the identifier on the left side, unless the result is less than zero, in which case zero is used.

Arithmetic Operators	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division

Precedence

If you include more than one operator in an expression, the program evaluates the computations according to the following hierarchy:

Precedence	Operator
1	( )
2	* (Multiplication),/ (Division)
3	+ (Addition),- (Subtraction)

For example:

3 + 2 * 5 equals 13
3 + (2 * 5) also equals 13, but
(3 + 2) * 5 equals 25

Operator Rules

All the mathematical operators, as well as some functions, can have interval data identifiers, determinant identifiers, simple identifiers, and numeric constants as arguments. The functions and expressions will return an interval data cut or a determinant, as appropriate. A math operator can have one of three data types on each side: interval data, determinant (possibly with historical data), and scalar numbers. *Scalar* refers to a single value that completely specifies a quantity, such as a bill determinant value (in contrast to interval data cuts, which consist of many values). The general computation rules for arithmetic expressions are:

a. The operations should do what you expect; left operand value(s) are operated on by right operand value(s). The following makes this explicit.

b. Division by zero equals zero. Missing data has value zero.

b. If the result is interval data, it has the same "structure" (start and stop date, IPH, number of intervals, etc.) as the left operand. If the left operand is not interval data, the result has the same "structure" as the right side interval data.

b. If the result is a determinant, it has the same "structure" (number of historical values) as the left operand.

b. If a determinant has a current month value and at least one historical month value, it is assumed to be a historical determinant. If it is missing a month in a historical operation or function, the missing value is set to zero. If the determinant has only the current month value and no historical data, the current month value is used throughout.

b. If "=+" is used, it applies to all values computed, including interval values and determinant historical values.

Specific Computation Rules

The following table defines the operations for all possible combinations (the Result type is determined by the Left and Right operand types).

Rule	Result =	Left Operand	Right Operand
a)	Interval Data	Interval Data	Interval Data
b)	Interval Data	Interval Data	Determinant
c)	Interval Data	Interval Data	Scalar
d)	Interval Data	Determinant	Interval Data
e)	Determinant	Determinant	Determinant
f)	Determinant	Determinant	Scalar
g)	Interval Data	Scalar	Interval Data
h)	Scalar	Scalar	Determinant
i)	Scalar	Scalar	Scalar

Specific computation rules for these combinations are:

a. Each interval value is the corresponding left value operated on by the corresponding right value (missing right values are assumed to be zero). Type 'a' computations can be performed using the **INTDBLOCKOP Function** on page 9-4 and **INTDBLOCKOPNA Function** on page 9-6 in the *Oracle Utilities Rules Language Reference Guide*).

b. Each interval value is the corresponding left value operated on by the determinant's current month value. The right operand could also be the determinant's historical value (if present) that corresponds to the interval's date and time.

b. Each interval value is the corresponding left value operated on by the scalar.

b. Each interval value is the determinant's current month value operated on by the corresponding right value. The left operand could also be the determinant's historical value (if present) that corresponds to the interval's date and time.

b. Each result value (current month and historical values) is the corresponding left value operated on by the corresponding right value, as determined by rule 3 above.

b. Each result value (current month and historical values) is the corresponding left value operated on by the scalar.

b. Each interval value is the scalar operated on by the corresponding right value.

b. The result value is the scalar operated on by the determinant's current month value.

b. The result value is the left scalar operated on by the right scalar.

Interval Data Mask Operator Rules

If the operation is between two interval data cuts (as in *a)* above) and the right one is a mask, the following rules apply:

a. If the left cut is also a mask, the result is a mask and the math operations are:

Operation	Rule
+	A union ("OR") of the masks; the value is '1' if either corresponding value is '1'; otherwise it is '0'.
-	The value is '1' if the left value was '1' and the right is '0'; otherwise, it is '0' (it removes "on" intervals in the right mask from the left mask).
*,/	An "AND" of the masks; the value is '1' if both corresponding values are '1'; otherwise it is '0'.

b. If the left cut is a simple interval data cut, the result is a similar cut and the math operations are:

Operation	Rule
+,*,/	The value is the left value if right is '1'; otherwise it is '0' (it leaves only "on" intervals in the right mask in the cut).
-	The value is the left value if right is '0'; otherwise it is '0' (it removes "on" intervals in the right mask from the cut).

Computing Load Factor in Masked Cuts

The INTDVALUE "LF" (load factor) is computed as the total / number of non-9 status code values / maximum. In a masked cut, the masked out intervals usually have a non-9 status code; their zero value produces an incorrect load factor, where the load factor for only the masked on values is to be computed. To correct this, you can use the Rules Language to compute the correct load factor.

Assume that KWH_HNDL is an interval data reference, and we want to compute the load factor for the intervals that were in either STANDBY or MAINTENANCE periods. This is done by:

```
/* Create mask for the STANDYBY and MAINTENANCE periods */
STBY_MASK = INTDCREATEOVERRIDEMASK(KWH_HNDL, "STANDBY", "MASK");
MAIN_MASK = INTDCREATEOVERRIDEMASK(KWH_HNDL, "MAINTENANCE", "MASK");
/* Create a mask of the union of the two periods */
BOTH_MASK = STBY_MASK + MAIN_MASK;
/* Get the KWH values in either one of these periods */
BOTH_HNDL = KWH_HNDL * BOTH_MASK;
/* Compute the load factor, using the non-zero count from the mask to
compute the average */
LF = (KWH_HNDL.TOTAL / BOTH_MASK.COUNT_NZ) / KWH_HNDL.ABS_MAXIMUM;
```

This example demonstrates using the mask count to compute the average for the intervals of interest:

```
AVERAGE = KWH_HNDL.TOTAL / BOTH_MASK.COUNT_NZ.
```

Chapter 5

Statements Overview

This chapter provides an overview of the statements available in the Oracle Utilities Rules Language, including:

- **General Statements**
- **Control Statements**
- **Revenue Computation Statements**
- **Report Statements**
- **Miscellaneous Statements**
- **Financial Management Statements**
- **WorkFlow Manager Statements**
- **XML Statements**

Statements are the building blocks of rate forms. Some statement types are specifically for rate computations and reporting, and they reflect the terms and concepts found in actual rate tariffs. Others are less rigidly defined, providing additional flexibility to structure and calculate virtually any type of rate, no matter how innovative or complex.

> **Note:** When reading about and using statements, you may find it helpful to keep in mind that a rate form tells the program how to calculate and report charges for one account at a time. For example, when you specify a bill determinant, the billing program automatically gets the value for the current bill period for the account that's being billed (unless you tell it otherwise). The analysis programs can report usage and trial revenue for one customer, a group of customers, or a bill frequency table, but the schedule you create is the same, regardless.

General Statements

General statements are used throughout rate forms in a number of different ways.

Note: Parenthetical page numbers refer to the Oracle Utilities Rules Language Reference Guide.

Assignment Statement (page 2-2)

Assigns a value to an identifier. The value can be a constant, the result of an arithmetic operation on variables and constants, or the result of one of the Rules Language functions. Applications for the ASSIGNMENT Statement include assigning a charge, loading interval data for the current period, performing block operations on cuts, etc.

Comment Statement (page 2-5)

Annotates your rate form with explanations or reminders. They have no meaning to the program.

Control Statements

Control statements control the order and way in which rate forms are processed.

Abort Statement (page 3-2)

Used with IF-THEN-ELSE to stop processing an account's bill when a condition you specify is met (or not met), and to issue an explanatory message on page 1 of the bill report.

Call Statement (page 3-3)

Used to dynamically execute one rate form while in another. Within a rate schedule, you can call riders and contracts. Within a rider, you can call other riders. See **Rate Form Types** on page 1-5 for more information.

Done Statement (page 3-5)

Stops processing an account's bill (successfully) when a specified condition occurs.

For Each Statements (page 3-7)

Directs the program to repeat a set of statements for each item in a set of items. The set of statements can be a list of items that was retrieved from a table in the Oracle Utilities Data Repository according to a user-specified query, the overrides or factor values that applied to the account during the bill period, a series of numbers, the weeks in a user-specified period, or a set of unlike items that you specify in the statement. There are several different types of FOR EACH statements.

If-Then-Else Statement (page 3-21)

Directs the program to evaluate a condition and take action based on whether or not the condition was met. You could use it to assign a discount or penalty if an account's value for kWh exceeded a certain value.

Include Statement (page 3-23)

The INCLUDE Statement is used to execute one rate form while in another. Within a rate schedule, you can include riders and contracts. You can include a rider within another rider. See **Rate Form Types** on page 1-5 for more information.

> **Note**: If it becomes necessary to include individual sections of a contract at different points in the rate schedule, use the Section option in the INCLUDE Statement.

Leave For Statement (page 3-25)

Directs the program to exit the nearest enclosing FOR EACH Statement.

Leave Rider Statement (page 3-25)

Directs the program to terminate an INCLUDEd or CALLed rider or contract.

Next For Statement (page 3-25)

Directs the program to skip the remaining statements in the enclosing FOR EACH Statement.

Novalue Statement (page 3-26)

Assigns a value to an identifier that would otherwise be null; for example, to supply a value when an account's record field is empty.

Section Statement (page 3-27)

Defines a section in a contract.

Select Bill_Period Statement (page 3-28)

Makes it possible to specify different pricing options for different seasons within a single rate form.

Select Expression Statement (page 3-31)

Makes it possible to specify different actions depending on the value of an identifier or expression.

Select Rate_Code Statement (page 3-33)

Makes it possible to specify different pricing options for a number of rate codes within a single rate form.

Warn Statement (page 3-35)

Similar to ABORT, the WARN Statement is used with IF-THEN-ELSE to issue a warning message in the bill report when a condition you specify is met (or not met). Unlike ABORT, it does not stop processing. It calls the billing analyst's attention to a potential problem. Oracle Utilities Billing Component displays up to 50 warning messages per report.

Revenue Computation Statements

Revenue Computation statements are used to compute revenue based on bill determinants, unit charges, and other factors.

All Statement (page 4-2)

Assigns a unit price to a bill determinant, and reports the unit price, the number of units the account consumed during the bill period, and the total charge for it.

Block Statements (page 4-4)

Defines the block units and unit charges for any type of block rate. It reports the total charge to the account for the block rate, and (optionally) the account's usage and charges for each individual block in the rate.

Unbilled and Ignore Statements (page 4-9)

UNBILLED

Reports the account's usage for a bill determinant for which there is no charge. In some situations, even though the determinant isn't used to calculate the current bill, it's still useful to report. The UNBILLED Statement is typically applied to a determinant that was part of an IGNORE Statement.

IGNORE

Excludes a charge from the account's bill. It is typically used within an IF-THEN-ELSE Statement (page 3-21) to ignore a charge under certain conditions, such as when a usage-based charge is less than a set minimum charge.

Report Statements

Report statements are used with the Print Detail options to identify the values that appear in reports, and how they are labelled.

Clear Statement (page 5-2)

Resets the values of Rules Language identifiers to Null. You might use this statement if your rate schedule has either SAVE or FOR EACH statements that involve stem.component identifiers (see **Record Identifiers (stem.component)** on page 4-14 for more information).

Determinant Statement (page 5-4)

Substitutes a user-defined descriptive label in reports for the bill determinant identifier used in the rate form.

Label Statement (page 5-6)

Label statements are similar to DETERMINANT and REVENUE statements, except that they can be used to label non-revenue and non-determinant identifiers. If the Print Detail option in effect for the account is "All," the value assigned to a labeled identifier is displayed in the reports. This is useful for reporting the results of intermediate calculations, or for displaying specific values of interest.

Remove Statement (page 5-7)

Removes the values of Rules Language identifiers from the Shared Symbol table. Used to free memory and resources.

Report Statement (page 5-8)

Used to label and report values for identifiers (including stem.components) that may be assigned different values during the rate form's execution by the billing or analysis program. While the LABEL, DETERMINANT, and REVENUE statements report whatever value is assigned to their identifier when the rate form is done executing, a REPORT Statement writes out a value each time it is executed. For example, if you nest a REPORT Statement inside a FOR EACH Statement, the statement will report its values for every pass of the FOR EACH loop. If you nest a LABEL inside a FOR EACH loop, only the value for the last pass of the loop will appear in the report.

Revenue Statement (page 5-10)

Replaces the revenue identifier in the rate form with a more descriptive label in bill reports. You can also use it to make a simple identifier a revenue identifier, even if it has no leading $. (Revenue Identifiers are a special class of identifiers used for charges, and whose values are automatically printed in bill reports when the "All" or "Normal" Print Detail option is in effect.) In addition, you can use the optional TOTAL clause in a REVENUE Statement to substitute an identifier of your choosing for the pre-defined identifier for the bill total ($EFFECTIVE_REVENUE).

Miscellaneous Statements

Delete Statement (page 6-2)

Used to remove a record from the database.

Save Statements (page 6-3)

Saves bill determinant values, records, or interval data cuts that were created during the bill calculations to the Oracle Utilities database. You also use the SAVE Statement to write computed values to the transaction records, so that they can be passed to CIS or a bill printer.

There are several versions of the Save Statement:

- **Save As:** Saves a single value for a bill determinant identifier to the Bill History Table or the Bill History Value Table.
- **Save To Table:** Saves values to a specified table in the Oracle Utilities Data Repository.
- **Save To Channel:** Saves interval data to a specified recorder,channel ID in the Oracle Utilities Data Repository.
- **Save To CIS:** Saves billing/calculation results to a CIS output file.
- **Save To XML:** Saves data to in XML format.
- **Save to Staging**: Saves interval data to a specified recorder,channel ID in either the Interval Data Staging (LSINTDSTAGING) or Interval Data Reporting (LSRFINTDHEADER and LSRFINTDVALUES) tables in the Oracle Utilities Data Repository.
- **Save Commit:** Commits database changes due to Save To Table, Save To Channel, and the LISTUPDATE function.
- **Save Rollback:** Removes database changes due to Save To Table, Save To Channel, and the LISTUPDATE function.

Financial Management Statements

Financial Management statements are used with the Oracle Utilities Receivables Component (Oracle Utilities Receivables Component) to Oracle Utilities Billing Component to post and/or cancel charges, statements, and bills.

Post Charge Or Credit Statement (page 7-7)

Posts a charge or credit as a single transaction.

Post Tax Statement (page 7-9)

Posts a tax charge or credit transaction for a specified account.

Post Installment Statement (page 7-11)

Posts a non-deferred charge transaction related to a previously created installment plan against a specified account.

Post Statement Statement (page 7-13)

Posts a single statement transaction against an account.

Post Bill Statement (page 7-15)

Posts a bill transaction against an account.

Post Payment Statement (page 7-17)

Posts a payment transaction against an account.

Post Adjustment Statement (page 7-19)

Posts an adjustment transaction against an account.

Post Refund Statement (page 7-21)

Posts a refund transaction against an account.

Post Writeoff Statement (page 7-23)

Used to write off an account.

Post Deposit Statement (page 7-25)

Posts a deposit charge transaction against an account.

Post Deposit Interest Statement (page 7-27)

Used to post deposit interest as a single transaction.

Post Deposit Application Statement (page 7-29)

Applies a deposit as a single transaction.

Cancel Transaction Statement (page 7-31)

Cancels a single transaction.

WorkFlow Manager Statements

WorkFlow Manager statements are used with WorkFlow Manager to start, suspend, resume, and terminate processes, and to post process events.

Process Start Statement (page 8-3)
Starts a new process instance.

Process Suspend Statement (page 8-5)
Suspends an existing running process instance.

Process Resume Statement (page 8-7)
Resumes an existing suspended process instance.

Process Terminate Statement (page 8-9)
Terminates an existing process instance.

Process Event Statement (page 8-11)
Posts an activity event.

XML Statements

XML statements are used to work with XML files and documents.

Identifier Statement (p. B-4)
Used to define identifiers before they are used.

OPTIONS Statement (p. B-5)
Used to specify that the case of identifiers remain as entered.

XML_ELEMENT Statement (p. B-6)
Used to map an XML format to Rules Language identifiers.

FOR EACH x IN XML_ELEMENT_OF 0 Statement (p. B-8)
Used to repeat a set of nested statements for each element defined in an XML structure.

XML_OP Statement (p. B-9)
Used to perform an operation on one or more XML elements.

Chapter 6

Rules Language Functions Overview

This chapter contains an overview of the functions available with the Oracle Utilities Rules Language, and rules for how functions are used in rate forms.

The Oracle Utilities Rules Language includes an extensive and powerful library of functions. Many of these functions were created specifically for use in rate calculations, while others are general-purpose functions that give you virtually unlimited flexibility in calculating and reporting revenue and other information.

You can use these functions to perform calculations using a variety of data elements, including bill determinants and interval data cuts. For example, you can compute KVA from kW and find the maximum or minimum values from a group of selected values. You can also use the interval data functions to compute bill determinant values from account- or channel-level interval data.

Rules Language functions are accessed by clicking the **Functions** button on the ASSIGNMENT Statement template, or through the **Rules Language Elements** Editor described in **Chapter 3**: **How Rate Forms are Processed**.

The first section of this chapter contains brief overviews of each function, including page references for the complete description of each function in the following chapter. The functions are grouped by category, in the same manner as they are grouped in the **Rules Language Elements** Editor. The four main categories of functions are:

- **Interval Data Functions**
- **Meter Value Functions**
- **Math Functions**
- **String Functions**
- **Other Functions**

General rules for using all of the functions are provided in the section entitled **Rules for Using Functions** on page 6-23.

Note: Parenthetical page numbers refer to the Oracle Utilities Rules Language Reference Guide.

Interval Data Functions

The Interval Data functions were designed specifically for working with interval data. They get data from the Interval Database according to your specifications, making it available for calculations in the rate forms. In addition, all of the INTDLOADxxx functions automatically calculate summary information about the handle, and put the values in a temporary record that is also available for computations and reporting. The computed values are the result of adding, averaging, or taking the maximum of the interval values in the handle. They include the total energy represented by the handle, the peak value and time, and more. See the description of INTDLOAD for specifics about these summary values.

A number of Interval Data functions can be used to create a variety of interval data "masks." A *mask* is a created handle in which the interval values that meet a user-specified condition are distinguished from those that don't. The interval values that meet the condition may be set to 1, to their actual value, or to a value set in a parameter in the function (and all others to 0). In many cases, you can also specify the reverse. The resulting mask handle can be used in any interval data function or expression. Rules for using masks are described in **Chapter 4**: **Identifiers, Constants, and Expressions**.

INTDADDATTRIBUTE Function (page 9-2)

Adds a user-defined attribute to an interval data handle.

INTDADDVMSG Function (page 9-3)

Adds a validation message to an interval data handle.

INTDBLOCKOP Function (page 9-4)

Performs a block operation on the interval data values of one handle using the corresponding values of another handle. Available operations are add, subtract, multiply, divide, find maximum, find minimum, calculate kVa, and calculate ikVa.

INTDBLOCKOPNA Function (page 9-6)

Performs a block operation on the interval data values of one handle using the corresponding values of another handle. This is similar to the INTDBLOCKOP function, but allows use of non-aligned interval data cuts.

INTDCLOSE Function (page 9-8)

Closes an interval data file that was opened using the INTDOPEN function.

INTDCOUNT Function (page 9-9)

Counts the number of intervals in the handle. This function includes a variety of options for "filtering" the type of intervals to be counted; e.g., missing, non-missing, etc.

INTDCOUNTSTATUSCODE Function (page 9-10)

Counts the number of intervals in an interval data handle that match a specified status code.

INTDCREATEDAYMASK Function (page 9-12)

Creates a mask of values consisting of 0 or 1. The interval value in the new handle is 1 for the entire day if the original handle has a 0 value for any interval during that day, or 0 if the original has all nonzero values for the corresponding day. You can also specify the reverse.

INTDCREATEFACTORMASK Function (page 9-13)

Creates a mask of values in which each interval is set to the factor value in effect at that time.

INTDCREATEHANDLE Function (page 9-14)

Creates an interval data handle based on user-specified start time, stop time, and SPI.

INTDCREATEMASK Function (page 9-15)

Creates a mask of values consisting of 0 or 1. The interval value in the new handle is 1 if the corresponding value in the original handle is 0, or 0 if the original has a nonzero value. You can also specify the reverse.

INTDCREATEOVERRIDEDAYMASK Function (page 9-16)

Creates a mask of values for the effective period for an override, with each override period automatically extended to start and end at midnight (or the day start).

INTDCREATEOVERRIDEMASK Function (page 9-17)

Creates a mask of values for the override's effective period.

INTDCREATESTATUSCODEMASK Function (page 9-18)

Creates a handle whose interval values are based on comparison of status codes.

INTDCREATETOUPERIOD Function (page 9-19)

Creates a handle whose interval values that fall within a user-specified time of use period are distinguished from those that fall outside the period. The interval values in the period may be set to 1 (and all others to 0), or to their actual value (and all others to 0). You can also specify the reverse. The resulting "mask" handle can be used in TOU computations.

INTDDELETE Function (page 9-21)

Deletes one or more cuts from the Oracle Utilities Data Repository.

INTDDIPTEST Function (p. 9 - 22)

Examines an interval data handle for dips.

INTDEXPORT Function (page 9-23)

Exports data in a handle to a file.

INTDGETERRORCODE Function (page 9-25)

Returns the error code from the last interval data function call.

INTDGETERRORMESSAGE Function (page 9-26)

Returns an error message from the last function to use a specified interval data reference.

INTDISEQUAL Function (page 9-27)

Compares two cuts to determine if they are to be considered equal.

INTDJOIN Function (page 9-28)

Merges two interval data cuts into one, based on user-specified criteria.

INTDLOAD Function (page 9-34)

Loads and totalizes all interval data for a user-specified bill determinant for the current bill period.

INTDLOADACTUALCUT Function (page 9-35)

Loads a specific interval data cut.

INTDLOADDATES Function (page 9-36)

Loads and totalizes all interval data for a user-specified bill determinant over a user-specified date range.

INTDLOADHIST Function (page 9-38)

Loads and totalizes all interval data for a user-specified bill determinant for a user-specified set of historical bill periods (including the current one, if desired).

INTDLOADLIST Function (page 9-39)

Loads and totalizes the interval data for all channels in a Table.Column channel list, for the current bill period.

INTDLOADLISTDATES Function (page 9-40)

Loads and totalizes the interval data for all channels in a Table.Column channel list, over a user-specified date range.

INTDLOADLISTENERGY Function (page 9-41)

Loads and totalizes the interval data for all channels (or all channels in a list) that are billed and record kW or kWh. This function uses the handle start and stop times instead of the bill period start and stop.

INTDLOADLISTHIST Function (page 9-42)

Loads and totalizes the interval data for all channels in a Table.Column channel list, for a user-specified set of historical bill periods (including the current period, if desired).

INTDLOADRELATEDCHANNEL Function (page 9-43)

Loads interval data for a specified recorder channel.

INTDLOADSP Function (page 9-44)

Loads and totalizes all interval data for channels belonging to an Aggregation Group.

INTDLOADSTAGING Function (page 9-46)

Loads interval data from the Interval Data Staging tables (LSINTDSTAGING) for a user-specified date range.

INTDLOADUOM Function (page 9-47)

Loads and totalizes all interval data for a specified UOM and (optionally) end use for the current bill period.

INTDLOADUOMDATES Function (page 9-48)

Loads and totalizes interval data for a specified UOM and (optionally) end use over a user-specified date range.

INTDLOADUOMHIST Function (page 9-49)

Loads and totalizes interval data for a specified UOM and (optionally) end use for a user-specified set of historical bill periods (including the current one, if desired).

INTDLOADVERSION Function (page 9-50)

Loads interval data from the Interval Data Version Tables (LSCHVERSION and LSCDVERSION) for a specified versioned cut.

INTDOPEN Function (page 9-51)

Opens an interval data file in read-only, write-only, or read/write mode.

INTDREADFIRST Function (page 9-52)

Returns a reference to the first record in an Interval Database.

INTDREADNEXT Function (page 9-53)

Returns a reference to the next record in an Interval Database.

INTDRECCOUNT Function (page 9-54)

Returns the number of records in an Interval Database.

INTDRELEASE Function (page 9-55)

Releases the interval data reference before completion of the rate form, to free computing resources.

INTDREPLACE Function (page 9-56)

Replaces a range of intervals in a loaded interval data handle with a previously loaded handle.

INTDROLLAVG Function (page 9-57)

Calculates the rolling average (or total) of interval values in a handle.

INTDROLLPEAK Function (page 9-58)

Calculates the rolling peak of interval values in a handle.

INTDSCALAROP Function (page 9-59)

Performs a scalar operation on each interval in the handle.

INTDSCALE Function (page 9-61)

Aggregates values in an existing handle according to user-specified parameters, such as to 15 minutes, 30 minutes, 60 minutes, hours, days, weeks, months, etc.

INTDSETATTRIBUTE Function (page 9-63)

Sets attributes of a specified data handle.

INTDSETDSTPARTICIPANT Function (page 9-65)

Changes the DST Participant flag for a previously-loaded interval data handle, and optionally adjusts the handle's Start Time and Stop Time as needed.

INTDSETSTRING Function (page 9-66)

Sets the status codes of all non-missing intervals in an existing handle.

INTDSETVALUE Function (page 9-67)

Sets an interval value of the interval data handle.

INTDSETVALUESTATUS Function (page 9-68)

Changes the status code and/or value of intervals in a handle.

INTDSHIFTSTARTTIME Function (page 9-70)

Shifts the start time of an interval data handle.

INTDSMOOTH Function (page 9-71)

Smooths gaps in interval data.

INTDSORT Function (page 9-72)

Sorts the values in an interval data handle.

INTDSPIKETEST Function (p. 9 - 73)

Examines an interval data handle for spikes.

INTDSUBSET Function (page 9-74)

Returns a subset of specified interval data.

INTDTOU Function (page 9-75)

Takes a handle that you've previously loaded and creates a handle for a specified Time-of-Use Schedule and Holiday List. The intervals within the periods keep their actual recorded values; those outside are set to 0. (Internally, the function creates as many cuts as there are periods in the TOU schedule, but you work with them as though there is just one.)

INTDTOURELEASE Function (page 9-76)

Releases the Time of Use reference that was set with INTDTOU before completion of the rate form, to free computing resources.

INTDTOUVALUE Function (page 9-77)

Computes a user-specified summary value for a TOU handle that was created with the INTDTOU function. Possible values include "ENERGY" (total energy in TOU period), "AVERAGE" (average for all non-missing interval values in the TOU period), "MAXIMUM" (peak value in the TOU period), or any of several others.

INTDUPDATESTATS Function (page 9-78)

Updates statistics for an interval data handle operated on the INTDREPLACE function.

INTDVALUE Function (page 9-79)

Computes a user-specified summary value for an interval data handle, such as "ENERGY" (total energy in handle), "AVERAGE" (average for all non-missing interval values), "MAXIMUM" (peak value in the handle), or any of several others.

STDEV Function (page 9-84)

Returns the standard deviation for a previously loaded interval data handle.

Enhanced Interval Data Functions

The following interval data functions work with interval data stored in Enhanced Interval Data tables, such as the Meter Data Channel Cut table used by the Oracle Utilities Meter Data Management application.

INTDDELETEEX Function (page 9-86)

Deletes an interval data cut from a specified Enhanced Interval Data table.

INTDGETATTREXALL Function (page 9-87)

Gets multiple custom and parent attributes of a specified enhanced interval data handle.

INTDLOADEXACTUAL Function (page 9-88)

Loads a specific interval data cut from a specified Enhanced Interval Data table.

INTDLOADEXCUT Function (page 9-89)

Loads a specific interval data cut from an Enhanced Interval Data Versioning table.

INTDLOADEXDATES Function (page 9-90)

Loads interval data for a user-specified date range from a specified Enhanced Interval Data table.

INTDLOADEX Function (page 9-93)

Loads and totalizes all of an account's interval data for a user-specified determinant for the current bill period from a specified Enhanced Interval Data table.

INTDLOADEXLIST Function (page 9-94)

Totalizes the interval data stored in an enhanced interval data table for the current bill period for all parent records in a list.

INTDLOADEXLISTDATES Function (page 9-95)

Totalizes the interval data stored in an enhanced interval data table for all parent records in a list over a specified time range.

INTDLOADEXRELATEDCHANNEL Function (page 9-96)

Loads the interval data for the related meter specified in the Oracle Utilities Meter Data Management Meter table from a specified Enhanced Interval Data table.

INTDSAVEEX Function (page 9-97)
Saves an interval data handle to a specified Enhanced Interval Data table.

INTDSAVEEXP Function (page 9-99)
Saves an interval data handle and its parent to specified Enhanced Interval Data tables.

INTDSETATTREX Function (page 9-101)
Sets an attribute of a specified enhanced interval data handle.

INTDSETATTREXALL Function (page 9-102)
Sets multiple custom and parent attributes of a specified enhanced interval data handle.

INTDVALUEEX Function (page 9-103)
Returns an attribute of a specified enhanced interval data handle.

Meter Value Functions

The Meter Value functions load data from the Meter Value Table. The Meter Value Table stores bill determinant values for entities in the billing hierarchy other than accounts—that is, individual channels, channel groups, or CIS accounts. You can use the Meter Value functions to load a single bill determinant value, to add all values for the same determinant, or to retrieve historical bill determinant values for any of these entities.

Each determinant value is loaded as part of a temporary record that includes other information about that bill determinant value. Specifically, this includes the scheduled read date, earliest start time among totaled values, the latest stop time among totaled values, and a string value (which values are loaded depends on the function). Each record identifier may have historical values. See the description of MVLOAD for specifics about these summary values.

MVLOAD Function (page 10-2)

Loads meter value(s) for the specified billing determinant and (optionally) billing entity (CIS account, channel group, or recorder/channel).

MVLOADACCT Function (page 10-4)

Loads and totalizes the meter values for all records belonging to an account for the current bill period. The "account" is usually the SYSTEM account, which stores the monthly peak time and value for the entire operating company.

MVLOADACCTDATES Function (page 10-5)

Loads and totalizes the meter values for all records belonging to an account for a user-specified date range. The "account" is usually the SYSTEM account, which stores the monthly peak time and value for the entire operating company.

MVLOADACCTHIST Function (page 10-6)

Loads and totalizes the meter values for all records belonging to an account for a user-specified set of historical bill periods (including the current one, if desired). The "account" is usually the SYSTEM account, which stores the monthly peak time and value for the entire operating company.

MVLOADDATES Function (page 10-8)

Loads meter value(s) for the specified billing determinant and (optionally) billing entity (CIS account, channel group, or recorder/channel) for a user-specified date range.

MVLOADHIST Function (page 10-9)

Loads meter value(s) for the specified billing determinant and (optionally) billing entity (CIS account, channel group, or recorder/channel) for a user-specified set of historical bill periods (including the current one, if desired).

MVLOADLIST Function (page 10-10)

Loads and totalizes the meter values for all entities in a Table.Column list (CIS accounts, channel groups, or recorder/channels) for the current bill period.

MVLOADLISTDATES Function (page 10-11)

Loads and totalizes the meter values for all entities in a Table.Column list (CIS accounts, channel groups, or recorder/channels) over a user-specified date range.

MVLOADLISTHIST Function (page 10-12)

Loads and totalizes the meter values for all entities in a Table.Column list (CIS accounts, channel groups, or recorder/channels) for a user-specified set of historical bill periods (including the current one, if desired).

Math Functions

Math functions perform various mathematical operations on input values, and return the result of those operations.

ACOS Function (page 11-2)
Returns the arccosine value of an input value.

ASIN Function (page 11-3)
Returns the acrsine value of an input value.

ATAN Function (page 11-4)
Returns the arctangent value of an input value.

ATAN2 Function (page 11-5)
Returns the arctangent value of the result of the division of the first input value by the second input value.

BITAND Function (page 11-6)
Returns an integer that is the result of a bitand operation on two supplied integer values.

CEIL Function (page 11-7)
Computes the smallest integer greater than or equal to a specified value.

COS Function (page 11-8)
Returns the cosine value of an input value.

COSECANT Function (page 11-9)
Returns the cosecant (1/sin) value of an input value. On an error, returns zero (0).

COSH Function (page 11-10)
Returns the hyperbolic cosine value of an input value.

COTANGENT Function (page 11-11)
Returns the cotangent (1/tan) value of an input value. On an error, returns zero (0).

DIVQUOT Function (page 11-12)
Returns the integral quotient of the result of dividing the first input value by the second input value.

DIVREM Function (page 11-13)
Returns the integral remainder of the result of dividing the first input value by the second input value.

EXP Function (page 11-14)
Returns the exponential value of an input value on success or zero (0) on overflow (input > 709.782712893) and underflow (input < -708.396418532264).

FABS Function (page 11-15)
Returns the absolute value of an input value.

FLOOR Function (page 11-16)
Computes the largest integer less than or equal to a specified value.

FMOD Function (page 11-17)
Returns the remainder of the first input value divided by the second input value.

FREXPM Function (page 11-18)
Breaks down an input value into a mantissa (*m*) and an exponent (*n*), and returns *m*.

FREXPN Function (page 11-19)
Breaks down an input value into a mantissa (*m*) and an exponent (*n*), and returns *n*.

LOG Function (page 11-20)
Returns the base e logarithm value of an input value.

LOG10 Function (page 11-21)
Returns the base 10 logarithm value of an input value.

MAX Function (page 11-22)
Returns the greater (greatest) value of two or more specified parameters.

MAXN Function (page 11-23)
Returns the *nth* greatest value among two or more specified parameters.

MIN Function (page 11-24)
Returns the lesser (least) value of two or more specified parameters.

MINNZ Function (page 11-25)
Returns the lesser (least) non-zero value of two or more specified parameters.

MODF Function (page 11-26)
Returns the signed fractional portion of an input value.

POW Function (page 11-27)
Returns the value of the first input value, raised to the power of the second input value.

ROUND Function (page 11-28)
Rounds a value to a user-specified number of decimal places.

ROUND2VALUE Function (page 11-29)
Rounds a value to the nearest multiple of another value.

ROUNDINT Function (page 11-30)
Rounds a value to the nearest *n* number of digits, where *n* is a user-specified number of places.

SECANT Function (page 11-31)
Returns the secant (1/cos) value of an input value. On an error, returns zero (0).

SIN Function (page 11-32)
Returns the sine value of an input value.

SINH Function (page 11-33)
Returns the hyperbolic sine value of an input value. If the result is too large, returns zero (0).

SQROOT Function (page 11-34)
Computes the square root of a non-negative input value.

TAN Function (page 11-35)
Returns the tangent value of an input value.

TANH Function (page 11-36)
Returns the hyperbolic tangent value of an input value.

String Functions

String functions enable you to rearrange the characters in text strings (that is, values that have data type STRING). You can use them to make bills and reports more readable. Note that the first character in a string is always in position 1.

FLOAT2STRING Function (page 12-2)
Converts the value of an identifier to a string.

FLOAT2STRINGNC Function (page 12-3)
Converts the value of an identifier to a string, but without commas to mark the thousands.

INSTR Function (page 12-4)
Returns the position (denoted with an integer) of the first occurrence of one string (string2) in another (string1).

LEFT Function (page 12-5)
Returns the leftmost *n* characters of a string. If *n* is greater than the length of the string, the entire string is returned (not padded).

LEN Function (page 12-6)
Returns the length of a string (an integer).

LTRIM Function (page 12-7)
Returns the string with leading spaces removed.

MID Function (page 12-8)
Returns a specified number of characters, beginning at the user-specified start position.

RIGHT Function (page 12-9)
Returns the rightmost *n* characters of a string. If *n* is greater than the length of the string, the entire string is returned (not padded).

RTRIM Function (page 12-10)
Returns the string with trailing spaces removed.

STRING Function (page 12-11)
Returns the value of an identifier, converted to a string. Numbers are converted with commas to mark the thousands. Date/times are converted to the date/time display format.

STRINGNC Function (page 12-12)
Similar to the STRING function, except that STRINGNC converts numbers without commas to mark the thousands. This is desirable for formatting years, for example.

TOLOWER Function (page 12-13)
Returns the string with all uppercase letters converted to lowercase, and all other characters unchanged.

TOUPPER Function (page 12-14)
Returns the string with all lowercase letters converted to uppercase, and all other characters unchanged.

TRIM Function (page 12-15)
Returns the string with leading and trailing spaces removed.

Other Functions

The Rules Language offers a wide variety of functions in addition to the Interval Data/Meter Value, Math, and String functions. These other functions include:

- **Database Functions**
- **Date/Time Functions**
- **Historical Data Functions**
- **Internal Functions**
- **Season-Based Functions**
- **Oracle Utilities Receivables Component Functions**
- **XML/Document Object Management Functions**
- **Term Functions**
- **Miscellaneous Functions**

Database Functions

Database functions return information about identifiers and data in the Oracle Utilities Data Repository.

ACCOUNTFACTOR Function (page 13-2)

Returns a value that indicates whether a factor applied to the account on the end date of the specified time period.

ARRAYUPPERBOUND Function (page 13-3)

Returns the upper bound of an array identifier. The upper bound is the highest index of the array that has been assigned a value. Returns a scalar numeric value.

CALLSTOREDPROC Function (page 13-4)

Calls a stored procedure.

GETADOCONNECTION Function (page 13-6)

Gets the ADO database connection used by the Rules Language.

GETCONNECT Function (page 13-7)

Returns a string that is the connection string used to log on to the Oracle Utilities application.

GETDATASOURCE Function (page 13-8)

Returns a variant (COM object) that contains the current database connection used by the Rules Language.

GETQUALIFIER Function (page 13-9)

Gets the qualifier for the current database connection used by the Rules Language.

GETUSERID Function (page 13-10)

Returns a string that is the user id used to log on to the Oracle Utilities application.

HASVALUE Function (page 13-11)

Returns a value that indicates whether an identifier has a value in the database, has been assigned a value in the rate form, or has no value.

LISTCOUNT Function (page 13-12)

Returns a count of the number of items in a Table.Column list.

LISTOP Function (page 13-13)
Performs column functions (AVG, COUNT, MAX< MIN or SUM) on a Table.Column list.

LISTUPDATE Function (page 13-14)
Updates the column value of every record in a Table.Column list.

LISTVALUE Function (page 13-15)
Returns the first element in a Table.Column list.

PRORATEFACTOR Function (page 13-16)
Prorates a factor over a user-specified time period, which may span multiple bill periods.

RSPRORATE Function (page 13-17)
Returns a prorated value based on the time that the rate schedule is in effect during the bill period. This function is used primarily for new accounts.

SETBINPATH Function (page 13-18)
Returns the path to the LODESTAR\Bin directory.

SETDBMONITOR Function (page 13-19)
Turns the Oracle Utilities Transaction Management Database Monitor on or off.

WQ_OPEN Function (page 13-20)
Opens a work queue item record in the Work Queue Open Item table used by the Work Queues application.

Date/Time Functions

Date/Time functions compute time related variables. For some of these functions, the special identifier BILL_PERIOD represents the first day of the current bill period.

BILLINGHOURS Function (page 13-21)
Returns the number of hours in one or more user-specified billing periods.

DATE Function (page 13-22)
Converts a date with a datatype of STRING to one with a datatype of DATE.

DATEFROMFLOAT Function (page 13-23)
Converts a float value to a date/time.

DATETIMEFROMSTRING Function (page 13-24)
Converts a string value to a date/time.

DATETIMETOSTRING Function (page 13-25)
Converts a date/time value to a string.

DATETOFLOAT Function (page 13-26)
Converts a date to a floating point number that can be stored as a determinant.

DAY Function (page 13-27)
Returns the number of the day of the month (1 to 31) of a date identifier or date string.

DAYDIFF Function (page 13-28)
Returns the number of days between two dates.

DAYNAME Function (page 13-29)
Returns the name of the day of the week for a specified date or value assigned to a date identifier.

DBDATETIME Function (page 13-30)
Returns the value of a date/time as a string suitable for use in a database key. The format of the string is database specific.

HOUR Function (page 13-31)
Returns the number of the hour (0 through 23) based on a date identifier or date string.

MINUTE Function (page 13-32)
Returns the number of the minute (0 through 59) based on a date identifier or date string.

MONTH Function (page 13-33)
Returns the number of the month (January is 1, December is 12) for a date identifier or date string.

MONTHDIFF Function (page 13-34)
Returns the number of months between two dates.

MONTHHOURS Function (page 13-35)
Returns the number of hours in one or more user-specified calendar months.

MONTHNAME Function (page 13-36)
Returns the name of the month of the year for a specified date or value assigned to a date identifier.

ROUNDDATE Function (page 13-37)
Returns a date rounded back to the nearest hour, day, week, month, or year.

SAMEWEEKDAYLASTYEAR Function (page 13-38)

Returns the closest date from a year prior to the supplied date that is on the same day of the week.

SECOND Function (page 13-39)

Returns the number of the second (0 through 59) based on a date identifier or date string.

WEEKDAY Function (page 13-40)

Returns the name of the day of the week from Sunday (Sunday=0, Monday=1, ..., Saturday=6), based on a date identifier or date string.

WEEKDIFF Function (page 13-41)

Returns the number of weeks between two dates.

YEAR Function (page 13-42)

Returns the number of the year in a date (all four digits), given a date identifier or date string.

YEARDAY Function (page 13-43)

Returns the number of days of the year since January 1, 0 - 365 (January 1 is 0), based on a date identifier or date string.

YEARSTR Function (page 13-44)

Like the YEAR function, this function returns the name of the year without commas marking the thousands.

Historical Data Functions

Many functions automatically return the bill determinant values for the current bill period. However, some rates require access to values for past bill periods. The **Historical Data Functions** on page 6-16 and **Season-Based Functions** on page 6-18 provide access to this historical data in the Oracle Utilities Data Repository. Specifically, Historical Data functions provide access to historical bill determinant values stored in the Bill History Table, the Bill History Value Table, and the Meter Value Table.

In these functions, the first parameter (identifier) must always be the name of a bill determinant stored in the Data Repository, or a retrieved METERVALUE record identifier. The parameter represents all of the values from the past stored in the database, not just the current one. You can specify the desired time period using the start and end bill period parameters described earlier in this chapter.

> **Note:** If the historical records in the database do not include a requested period (as can happen with new or inactive customers, for example) the program will assume that the period's value is 0. If the specified start and end periods are completely outside the range of the available data for the determinant, the program will issue an error message in the bill report.

COMPSUM Function (page 13-46)

Totals values for a historical determinant over user-specified bill periods.

HISTCOUNT Function (page 13-47)

Returns the number of historic values loaded.

HISTMAX Function (page 13-48)

Compares two or more sets of historical values and/or constants, and returns the greater value in each set.

HISTMIN Function (page 13-49)

Same as HISTMAX, except it finds the minimum of the corresponding values.

HISTMINNZ Function (page 13-50)

Same as HISTMIN, except it finds the nonzero minimum of the corresponding values.

HISTVALUE Function (page 13-51)

Returns the historic value for the specified bill period.

MAXNRANGE Function (page 13-52)

Finds the *n*th maximum of the historical determinant values between the start bill period specified through the end bill period specified.

MAXRANGE Function (page 13-53)

Finds the maximum of the historical determinant values between the start bill period specified through the end bill period specified.

MINRANGE Function (page 13-54)

Finds the minimum of the historical determinant values between the start bill period specified through the end bill period specified.

Internal Functions

Internal functions compute a variety of values, such as kVA or the maximum of specified values.

COMPIKVA Function (page 13-55)
Computes rkVA from kVA and kW, or computes kW from kVA and rkVA.

COMPKVA Function (page 13-56)
Computes kVA from rkVA and kW.

COMPKVARHFROMKQKW Function (page 13-57)
Computes kVARH from kQh and kWh.

COMPLF Function (page 13-58)
Computes load factor.

IDATTR Function (page 13-59)
Returns the value of an attribute of an identifier.

FLAG Function (page 13-61)
Returns the setting for analysis flags.

LF2KW Function (page 13-62)
Computes kW from kWh and load factor.

LF2KWH Function (page 13-63)
Computes kWh from kW and load factor.

MAXKW Function (page 13-64)
Compares values for metered kW, contract kW, historical kW, and optionally minimum kW, and assigns a value to the special identifier AUXILIARY_DEMAND based on the result.

POWERFACTOR Function (page 13-65)
Returns the ratio of real power (kWh) to apparent power (kVARH) for any given load and time.

READING2USAGE Function (page 13-66)
Returns the computed usage for a selected billing determinant.

Season-Based Functions

Season-Based functions support seasonal rates. Like the Historical Data functions, Season-Based functions work with historical determinant values.

AVGSEASON Function (page 13-67)
Finds the average value of a selected bill determinant during a specified season period.

MAXSEASON Function (page 13-69)
Finds the maximum historical value of a selected bill determinant during a specified season period.

MINSEASON Function (page 13-70)
Finds the minimum historical value of a selected bill determinant during a specified season period.

MONTHLYMERGE Function (page 13-71)
Combines all values in each bill month according to type.

SEASONVALUE Function (page 13-72)
Finds the portion of the specified historical identifier for the current month in the specified season.

SUMSEASON Function (page 13-73)
Finds the sum of the historical determinant values of a selected bill determinant during a specified season period.

Oracle Utilities Receivables Component Functions

FMGETBILLINFO Function (page 7-34)
Obtains bill information for an account.

XML/Document Object Management Functions

XML/DOM functions are used to manipulate XML documents and files, and to set and retrieve XML node values.

DOMDOCCREATE Function (p. B-13)
Creates an XML document with a root element node.

DOMDOCLOADFILE Function (p. B-14)
Loads and parses an XML file.

DOMDOCLOADXML Function (p. B-15)
Loads and parses an XML document.

DOMDOCSAVEFILE Function (p. B-16)
Saves an XML file based on a specified XML document.

DOMDOCGETROOT Function (p. B-17)
Retrieves the root node of an XML document.

DOMDOCADDPI Function (p. B-18)
Adds a processing instruction to an XML document.

DOMNODEGETNAME Function (p. B-19)
Retrieves the name of an XML node.

DOMNODEGETTYPE Function (p. B-20)
Retrieves the type of an XML node.

DOMNODEGETVALUE Function (p. B-21)
Retrieves the value of an XML node.

DOMNODEGETCHILDCT Function (p. B-22)
Retrieves the number of child nodes of an XML node.

DOMNODEGETFIRSTCHILD Function (p. B-23)
Retrieves the first child of an XML node, if any.

DOMNODEGETSIBLING Function (p. B-24)
Retrieves the next (right side) child of an XML node, if any.

DOMNODECREATECHILDELEMENT Function (p. B-25)
Creates a child node in an XML node.

DOMNODESETATTRIBUTE Function (p. B-26)
Sets the value of an attribute of an XML node.

DOMNODEGETCHILDELEMENTCT Function (p. B-27)
Retrieves the number of child nodes of an XML node that are elements.

DOMNODEGETFIRSTCHILDELEMENT Function (p. B-28)
Retrieves the first child of an XML node that is an element, if any.

DOMNODEGETSIBLINGELEMENT Function (p. B-29)
Retrieves the next (right side) child of an XML node that is an element, if any.

DOMNODEGETATTRIBUTECT Function (p. B-30)
Retrieves the number of attribute nodes of an XML node, if any.

DOMNODEGETATTRIBUTEI Function (p. B-31)
Retrieves the *index*th attribute of an XML node, if any.

DOMNODEGETATTRIBUTEBYNAME Function (p. B-32)
Retrieves the attribute of an XML node with a specified name, if any.

DOMNODEGETBYNAME Function (p. B-33)
Retrieves the first node under a specified XML node with a specified name, if any.

Term Functions

Term functions retrieve and save terms used in contracts to and from the Oracle Utilities Data Repository.

LOADCONTRACTTERM Function (page 13-75)
Loads a specified contract term from the Contract Terms table.

LOADCONTRACTTERMALL Function (page 13-77)
Load all contract terms for a specific contract from the Contract Terms table.

LOADGROUPTERM Function (page 13-79)
Loads a specified group term from the Contract Item Group Terms table.

LOADGROUPTERMALL Function (page 13-82)
Loads all group terms for a specified contract from the Contract Item Group Terms table.

LOADITEMTERM Function (page 13-84)
Loads a specified contract item term from either the Contract Item Terms table, Contract Item Product Terms table, or Contract Item Details table.

LOADITEMTERMALL Function (page 13-87)
Loads all contract item terms for a specified contract and contract item from either the Contract Item Terms table, Contract Item Product Terms table, or Contract Item Details table.

SAVECONTRACTTERM Function (page 13-90)
Saves a specified contract term to the Contract Terms table.

SAVECONTRACTTERMALL Function (page 13-92)
Saves all contract terms for a specified contract to the Contract Terms table.

SAVEGROUPTERM Function (page 13-93)
Saves a specified group term to the Contract Item Group Terms table.

SAVEGROUPTERMALL Function (page 13-95)
Saves all group terms for a specified contract and contract group to the Contract Item Group Terms table.

SAVEITEMTERM Function (page 13-97)
Saves a specified contract item term to either the Contract Item Terms table, Contract Item Product Terms table, or Contract Item Details table.

SAVEITEMTERMALL Function (page 13-99)
Saves all contract item terms for a specified contract and contract item to either the Contract Item Terms table, Contract Item Product Terms table, or Contract Item Details table.

Miscellaneous Functions

The remaining functions are used for various purposes.

ACCTREADDATES Function (page 13-101)
Returns the read dates for a specified account.

ACCTTABLELOAD Function (page 13-102)
Returns all specified records within a specified date range.

CONFIGADD Function (page 13-103)
Adds the string value of parameters within a configuration file to the internal configuration settings.

CONFIGGET Function (page 13-104)
Returns the string value of a configuration file parameter.

CREATEOBJECT Function (page 13-105)
Creates a COM object, based on the object's ProgID.

CREATEREPORT Function (page 13-106)
Generates a report based on supplied parameters.

EMAILCLIENT Function (page 13-109)
Sends an email based on supplied parameters.

EXPBLKMDMUSAGE Function (page 13-112)
Exports usage for a specified account or service point over a specified date range to a Oracle Utilities Meter Data (*.lsm) file.

EXPMDMUSAGE Function (page 13-114)
Exports a specified usage reading to a Oracle Utilities Meter Data (*.lsm) file.

EXPORT_USAGE Function (page 13-116)
Exports interval data associated to a supplied Account ID to a Microsoft Excel (*.xls) file.

FACTORINEFFECT Function (page 13-118)
Checks to determine if a specified factor has a factor value on a specified date. Returns 1 (true) or 0 (false).

GETUSERSPECIFIEDSTOP Function (page 13-119)
Gets the "User Specified Stop" date (if specified).

INEFFECT Function (page 13-120)
Indicates whether or not a specified tariff rider, rate code, or override was in effect for the account on a specified date.

ISHOLIDAY Function (page 13-121)
Returns a value of 1 if the specified date is a holiday.

RUNRATE Function (page 13-122)
Executes a rate as a new process and continues current processing.

SAVE_PROFILE Function (page 13-123)
Saves a Rules Language code profile to a specified file.

SETREPORTTITLE Function (page 13-124)

Sets the report title for Rules Language reports.

USEREXIT Function (page 13-125)

Calls a user-written function for use in the rate form.

WAITFORRUNRATE Function (page 13-126)

Causes a rate to wait for rates created using the RUNRATE function.

Rules for Using Functions

This section explains the rules for using functions, including:

- **Functions and Identifiers**
- **About Identifiers**
- **About Parameters**

Functions and Identifiers

Most often, you assign the results of a function to an identifier using an ASSIGNMENT Statement. You can also use a function anywhere you can use an expression. In the descriptions in the following chapter, required keywords are shown in capital letters. Parameters you supply are indicated inside < >. You don't supply the pointed brackets in the function; they are used here to distinguish a parameter from other elements of the statement. The format of every function call is:

```
FUNCTION(<parameters>);
```

About Identifiers

You can assign many kinds of values to the identifier on the left side of the equal sign, depending on the function. Some functions return a simple scalar numeric value that can be assigned to any identifier except an interval data reference. Others return historical determinant values, and must be assigned to a determinant identifier (**Bill Determinant Identifiers** on page 4-5). Many of the interval data functions return an interval data reference that must be assigned to an interval data handle (see **Interval Data Handles** on page 4-6).

> **Note**: If you are unfamiliar with the concepts of identifiers, determinant identifiers, and interval data references, see **Chapter 4**: **Identifiers, Constants, and Expressions**.

In the detailed descriptions in the *Oracle Utilities Rules Language Reference Guide*, the types of identifiers are noted, as follows:

- <identifier> - The function returns a scalar numeric value that you can assign to any identifier except an interval data handle.
- <determinant_identifier> - The function returns a bill determinant value (including historical values) that you can assign to a determinant identifier.
- <interval_data_handle|'recorder,channel'> - The function returns an interval data handle that you should assign to an interval data handle or specific recorder,channel constant.

Although it is allowed, a function name should not be used as an identifier.

About Parameters

Each function requires you to specify some parameters (noted inside pointed brackets < >). Depending on the function you're using, you can supply any of the following for each parameter:

- The name of a bill determinant (for example, kW or rkVA)
- a constant
- a locally-defined identifier that you assigned to the results of an expression or function elsewhere in the rate form
- an expression that evaluates to the appropriate type of parameter.

  Note: If you supply the name of a bill determinant, the current month's value will be used. If you want to use historical billing determinant values, see **Historical Data Functions** on page 6-16.

Start and End Bill Period Parameters

In many of the functions described in this chapter, the last two parameters are <start_bill_period_previous>, <end_bill_period_previous>. Use 0 to specify the current period, 1 for the previous, and so on (the higher the value, the further back in time). The end period that you specify must be greater than or equal to the start period.

These parameters are optional, though you must specify a value for the start period if you specified one for the end period. The default value for the start period is 0 (the current bill period). The default for the end period is usually the last month of data stored for the determinant, but in some cases the default may be the start period. In other words, if you omit these two parameters altogether, the value is computed either over all values available, or just for the current month, depending on what is appropriate for the particular function.

About missing historical data: In some cases, determinant values may not be available for all requested bill periods, such as for a new account. If data is available for the newer periods but not the older periods, the program automatically adjusts the end period to reflect the data available. If the start period you specify is earlier than the end of the available data, the program returns an error message in the bill report.

For example, suppose an account has data for the last six bill periods but nothing prior to that. If the start and end bill period parameters in a function were (0,12), the program would automatically adjust the end period parameter to 6 for processing that account. If the start and end bill period parameters in the function were (7,12), the program would return an error message for that account, because there was no data available for the account for any bill period in the specified range.

Parameter Types

A parameter of a function can usually be either a constant or an identifier; though some function parameters must be identifiers.

The parameter value type may be IDENTIFIER, INTEGER, FLOAT, STRING, or DATE.

- If the type is IDENTIFIER, then an identifier is required.
- An INTEGER has no decimal point.
- A FLOAT has a decimal point.
- A STRING is any set of characters enclosed in quotes (“ ”).
- A DATE is in the form ‘mm/dd/yyyy’ or ‘mm/dd/yyyy hh:mm’.

IDENTIFIER is used for database identifiers, determinant identifiers, interval data handles, and Time of Use handles.

An INTEGER may be used if a FLOAT is required, but not vice versa. If the type is INTEGER, FLOAT, STRING, or DATE, a constant of that type or an identifier assigned a value with that type must be used as the parameter value. The different types of parameters are listed on the next page.

There is one special case. When specifying parameters for some functions, you must choose from a set of predefined options that are expressed as a text string. For example, to specify the <type> parameter for the INTDCOUNT function (page 9-9), you must select from “EXCLUDE,” “INCLUDE,” “NON-ZERO,” “ALL,” “HOURS,” or “DAYS.” You can supply any of these parameters using the identifier name format—that is, without quote marks—as long as you do not use the same set of characters on the left side of an ASSIGNMENT Statement elsewhere in the rate form.

For example, the following statement is acceptable, assuming that the variable name EXCLUDE is not assigned a value elsewhere in the rate form:

```
NUM_MISS = INTDCOUNT(KWH_HNDL, EXCLUDE);
```

The following is also acceptable, because EXCLUDE appears once as a STRING parameter and once as a variable name:

```
NUM_MISS = INTDCOUNT(KWH_HNDL, "EXCLUDE");
EXCLUDE = 1
```

The following is not acceptable:

```
NUM_MISS = INTDCOUNT(KWH_HNDL, EXCLUDE);
EXCLUDE = 1
```

Parameter Name	Type	Notes*
<contract_kw>	FLOAT	
<database_code>	STRING	(3)
<database_identifier >	IDENTIFIER	(1)
<date_identifier|date_constant>	DATE	
<determinant_identifier>	IDENTIFIER	(2)
<end_bill_period_previous>	INTEGER	
<end_use_code>	STRING	(3)
<historical_kw>	FLOAT	
<historical_identifier>	FLOAT	(5)
<holiday_name>	STRING	(3)
<identifier|constant>	FLOAT	(4)
<interval_data_handle>	IDENTIFIER	
<'recorder,channel'>	STRING	
<kva>	FLOAT	
<kw>	FLOAT	
<kwh>	FLOAT	
<lf>	FLOAT	
<metered_kw>	FLOAT	
<min_kw>	FLOAT	
<bill_period_previous>	INTEGER	
<operation>	STRING	(3)
<period>	STRING	(3)
<places>	INTEGER	
<rkva>	FLOAT	
<rkva_kw>	FLOAT	
<schedule_name>	STRING	(3)
<season_name>	STRING	(3)
<scalar_value>	FLOAT	
<start_bill_period_previous>	INTEGER	
<tou_handle>	IDENTIFIER	
<type>	STRING	(3)
<uom_code>	STRING	(3)

***Notes:**

1. Only for parameters that must be a database identifier of the form *table[key].column* or FACTOR[*key*].VALUE.
2. Only for parameters that must be a determinant. These identifiers are automatically marked as historical, and their historical values are loaded before the rate form executes.
3. Parameter can be any literal identifier or constant (see discussion on previous page). If the parameter you supply is an identifier and you have not assigned it a string value elsewhere in the rate form, the program uses the identifier name as a string value. If TOTAL=1 (or is unassigned) and OP_NAME= "TOTAL", then TOTAL, OP_NAME, and "TOTAL" all have the same string value.
4. Parameter can be any numeric identifier or constant.
5. Can be any determinant or record identifier with historical values.

Chapter 7

Working with Interval Data

This chapter describes how you work with Interval Data using the Oracle Utilities Rules Language, including:

- **Interval Data Functions Overview**
- **Timezones and DST**
- **Unit-of-Measure Rates and Quantities**
- **Loading Interval Data**
- **Creating Interval Data Masks**
- **Other Interval Data Operations**
- **Working with Enhanced/Generic Interval Data**
- **Deriving Billing Determinants and Values from Interval Data**
- **Examples of Working with Interval Data**

Interval Data Functions Overview

The Oracle Utilities Rules Language interval data functions compute values from interval data cuts stored in the Oracle Utilities Data Repository. Cuts are referenced in the Rules Language by interval data identifiers, **interval data reference** or **interval data handles**.

Interval data functions return either a new interval data reference or a scalar value (though in some case several values are computed at the same time). While most interval data functions use only one interval data reference as input, some take one or more interval data references as input and return either another reference or a value. The assigned interval data identifier may be the same as one of the input identifiers.

> **Note:** If you assign an interval data reference in an identifier directly to another identifier, the interval data in memory is copied so that changes to one identifier do not affect the other.

Interval data functions take a start and end month as optional input parameters. If the end month is omitted it is the same as the start month, if the start month is omitted it is assumed to be zero (the current billing period).

Interval Data Function Errors

If an error occurs in any interval data function there are several possible outcomes. These are determined first by the value of the INTD_ERROR_STOP identifier (if present), then by the value of the **On Rules Language Interval Data Error** option (set on the **Error Handling** tab of the **Default Options** dialog. The choices are:

INTD_ERROR_STOP Option	Result
<1 or >3	Ignore Return scalar integer 0
1	Error Stop with error message
2	Warning Issue warning message, return scalar integer 0
3	Informational Issue informational message, return scalar integer 0

The **On Rules Language Interval Data Error** option is used only if there is no INTD_ERROR_STOP value.

If an interval data reference is returned from a function, the user can check the return value, and if it is zero (0), the interval data function failed. In addition, the INTDATA_ERROR identifier is set by every interval data function call. This identifier is zero (0) if the function was successful, one (1) if an error occurred. Also, the **INTDGETERRORCODE Function** can be used to get the internal error code if there is an error, and the **INTDGETERRORMESSAGE Function** can be used to get the error message if there is an error.

Types of Interval Data Handles

There are two kinds of interval data handles: data handles and "masks". Data handles contain interval values, while masks consist of zeros and ones only. Operators and functions produce different results depending on whether the handle contains data or a mask. See **Interval Data Mask Operator Rules** on page 4-28 for more information.

A mask results from any interval data operation (except the **INTDLOAD Functions**) that produces a handle with only zeros and ones. **INTDLOAD Functions** load data cuts from the Data Repository. To convert a data handle to a mask, divide it by itself. The result has only zeros (where the original was zero) and ones everywhere else, and is a mask.

An interval data reference returned from a function may represent a cut retrieved directly from a database, or it may represent a computed handle. Computed handles are handles that comprise several recorder,channels, or are the result of non-INTLDOADxxx functions that return an interval data reference (such as the **INTDBLOCKOP Function** or the **INTDSCALAROP Function**).

Interval Data Function Parameters

Several of the interval data functions accept "operation", "type" or "attribute" parameters. You can use several different types of strings (including full name, abbreviation, or symbol) to represent the same operation. The following table lists the correspondences. All operations on a line are equivalent. As function parameters, all should be enclosed in double quotes (omitted here). The symbols are not listed below with the specific function parameters.

Full Name	Abbreviation	Symbol
MAXIMUM	MAX	>=
MIMIMUM	MIN	<=
TOTAL	ADD	+
AVERAGE	AVG	
AVERAGE_NZ	AVG_NZ	
ABSOLUTE	ABS	
SUBTRACT		-
MULTIPLY		*
DIVIDE_BY		/

Interval Data Reference Values and Attributes

Along with the interval data, the Rules Language automatically computes a group of summary values about each handle. These are the result of adding, averaging, or taking the maximum of the interval values in the handle. This data is stored in memory until the program determines that the rate form no longer needs it, or until you explicitly release it using the **INTDRELEASE Function**. You can apply statements to this group of values by identifying them with the convention <HNDL.ATTRIBUTE>, where HNDL is the interval data handle that you assigned in the INDTLOAD statement (which automatically refers to the entire handle), and ATTRIBUTE is the name of a particular attribute of the handle. For example, one of the computed summary values is AVERAGE, which is the average of all non-missing values in the interval data record. If you used the handle INT_MY_HNDL, you could retrieve the average value in using an Assignment Statement as follows:

```
HNDL_AVG = INT_MY_HNDL.AVERAGE;
```

Following is a list of values that are automatically loaded whenever you load an interval data cut using one of the **INTDLOAD Functions**. They are the result of adding, averaging, or taking the maximum of the interval values.

Value	Description
TOTAL	The sum of all interval values in the interval data handle.
ENERGY	Total energy represented by the handle, computed properly for its UOM according to the TOTAL flag in the UOM Table. The UOM for the interval values must be either KW or KWH. If not, result is 0.
AVERAGE	Average of all non-missing interval values.
AVERAGE_NZ	Average of all nonzero interval values.
MAXIMUM	Peak value (computed using actual values, not the absolute value of the values).
MAXIMUMn	Value of *n*th peak (e.g., MAXIMUM2 reports second highest peak. For *n*, you may supply any value from 2 through 10).
KW_MAXIMUM	The maximum KW value in the handle. If the UOM is KWH, the actual maximum is multiplied by the IPH (intervals per hour) to get this value.
ABS_MAXIMUM	Peak value (computed using absolute maximum). "Absolute" means the program converts negatives to positives and selects the largest.
MAXDATE	Date and time of the peak interval.
MAXDATEn	Date and time of *n*th peak interval (e.g., MAXDATE2 reports the date and time of second highest peak. For *n*, you may supply any value from 2 through 10).
ABS_MAXDATE	Date and time of peak interval (computed using ABS_MAXIMUM).
MINIMUM	Minimum interval value in the handle (computed using the actual values, not the absolute value of the values).
MINIMUM_NZ	Minimum of all nonzero values in the handle.
MINDATE	Date and time that the minimum occurred.
LF	Load factor (load factor=average interval/maximum interval value) (computed using the absolute maximum).
STARTTIME	Date and time of the start of the handle data.
STOPTIME	Date and time of the end of the handle data.
COUNT	Total number of intervals in the handle.
COUNT_NZ	Total number of nonzero intervals in the handle.
IPH	Intervals per hour.
SPI	Seconds per interval.
UOM	Unit of measure, denoted by a code (1 - 99).

Value	Description
DSTTOTAL	Total of the interval values in the fall Daylight Savings Hour (the 2:00 hour in the last Sunday in October). If the handle does not include this hour, the value is 0.
DSTENERGY	Total of the interval values in the fall Daylight Savings Hour (the 2:00 hour in the last Sunday in October). If the handle does not include this hour, the value is 0. The Unit of Measure for the intervals must be KW or KWH; otherwise, the value returned is 0.
MULTIPLIER	Pulse multiplier*.
OFFSET	Pulse offset*.
RECORDER	Recorder identifier. Used only with standard interval data tables. When an interval data handle is computed, this is set to "COMPUTED."
CHANNEL	Channel number. Used only with standard interval data tables. When an interval data handle is computed, this is set to "0."
RECORDERCHAN	Recorder,channel. Used only with standard interval data tables. When an interval data handle is computed, this is set to "COMPUTED,0."
PARENTKEY	A string containing the identity of the parent of the handle. Used only with enhanced interval data tables. When an interval data handle is computed, this is set to "COMPUTED."

In addition, some of these values may be set in the Rules Language using a statement of the form:

```
HNDL.ATTRIBUTE = <new_value>;
```

Attributes that can be used in this way are:

- **UOM**: Sets the Unit of Measure for the handle (numeric 1 - 99).
- **RECORDER**: Sets the Recorder ID for the handle.
- **CHANNEL**: Sets the Channel number for the handle.
- **DESCRIPTOR**: Sets the Description field for the handle
- **STATUSCODE**: Set all interval value status codes in the handle to the specified single character.
- **ORIGIN**: Sets the Origin field for the handle to Metered, Profiled or Computed ("M", "P", or "C").
- **PARENTKEY**: Sets the Parent Key for the cut.

Note: The following functions set the ORIGIN to Computed and the TIMESTAMP to now: **INTDCREATEMASK Functions**, INTDSCALE, INTDSCALAROP, INTDROLLAVG, INTDROLLPEAK, INTDJOIN, INTDSMOOTH, INTDBLOCKOP, and INTDSHIFTSTARTTIME. In addition the **INTDLOAD Functions** set these values if several handles are added together. In any Computed handle, the following meter values are set to 0: "START_READING", "STOP_READING", "METER_MULT", "METER_OFFSET", and "READING_VALUE".

Combining and Comparing Interval Data Handles

The following are some ways that handles can be combined or compared:

If HNDL contains a valid interval data handle, then

```
HNDL = HNDL;
```

will do nothing.

However, the statement

```
HNDL1 = HNDL2;
```

creates a whole new handle and copies the data, so freeing or changing either handle has no effect on the other.

The statement

```
HNDL3 = HNDL1 + HNDL2;
```

where HNDL1 is valid and HNDL2 is zero will return HNDL1 for the + and - operators, a zero handle for any other operator.

You can compare an interval data handle to zero. A zero handle is one that was created due to an error, such as a load failing. The expression in the IF statement

```
HNDL = INTDLOAD(...);
IF HNDL <> 0 THEN
```

will return true if the HNDL has been assigned a valid handle, and false if the load failed.

You can also compare interval data references by comparing the identifiers or using the **INTDISEQUAL Function**. For example:

```
HNDL1 = INTDLOAD(...);
HNDL2 = INTDLOAD(...);
EQUAL = INTDISEQUAL (HNDL1, HNDL2)
IF EQUAL > 0
    THEN ...
```

Timezones and DST

This section describes support for timezones and Daylight Savings Time (DST) in the Oracle Utilities Rules Language, including:

- **Timezone Support**
- **DST Support in the US**

Timezone Support

The enhanced interval data format supports timezones, represented by a number that is the number of 1/2 hours between GMT and the timezone, to the west. Thus EST has a timezone value of 10, and Germany would have one of 46. There is the ability on interval data import to specify the imported cut's timezone (see **Importing Interval Data** on page 3-5 of the *Data Manager User's Guide*). This results in start and stop time changes if the cut is from a different timezone, otherwise it simply sets the timezone value. The table below lists all the timezones supported by the Rules Language.

Timezone Codes

CODE	NAME	TIMEZONE	DST
GMT	Greenwich Mean Time	0	N
BST	British Summer Time	47	N
BSTA	British Summer Time - Adjusted	0	A
IST	Irish Summer Time	47	N
ISTA	Irish Summer Time - Adjusted	0	A
WET	Western Europe Time	0	N
WEST	Western Europe Summer Time	0	Y
WESTA	Western Europe Summer Time - Adjusted	0	A
CET	Central Europe Time	46	N
CEST	Central Europe Summer Time	46	Y
CESTA	Central Europe Summer Time - Adjusted	46	A
EET	Eastern Europe Time	44	N
EEST	Eastern Europe Summer Time	44	Y
EESTA	Eastern Europe Summer Time - Adjusted	44	A
MSK	Moscow Time	42	N
MSD	Moscow Summer Time	42	Y
MSDA	Moscow Summer Time - Adjusted	42	A
AST	Atlantic Standard Time	8	N
ADT	Atlantic Daylight Time	8	Y
ADTA	Atlantic Daylight Time - Adjusted	8	A
EST	Eastern Standard Time	10	N

CODE	NAME	TIMEZONE	DST
EDT	Eastern Daylight Time	10	Y
EDTA	Eastern Daylight Time - Adjusted	10	A
CST	Central Standard Time	12	N
CDT	Central Daylight Time	12	Y
CDTA	Central Daylight Time - Adjusted	12	A
MST	Mountain Standard Time	14	N
MDT	Mountain Daylight Time	14	Y
MDTA	Mountain Daylight Time - Adjusted	14	A
PST	Pacific Standard Time	16	N
PDT	Pacific Daylight Time	16	Y
PDTA	Pacific Daylight Time - Adjusted	16	A
UTC-12	Universal Time Coordinated - 12 hours	24	N
UTC-12D	Universal Time Coordinated - 12 hours (DST)	24	Y
UTC-12DA	Universal Time Coordinated - 12 hours (24 Hr Adjusted)	24	A
UTC-11	Universal Time Coordinated - 11 hours	22	N
UTC-11D	Universal Time Coordinated - 11 hours (DST)	22	Y
UTC-11DA	Universal Time Coordinated - 11 hours (24 Hr Adjusted)	22	A
UTC-10	Universal Time Coordinated - 10 hours	20	N
UTC-10D	Universal Time Coordinated - 10 hours (DST)	20	Y
UTC-10DA	Universal Time Coordinated - 10 hours (24 Hr Adjusted)	20	A
UTC-9	Universal Time Coordinated - 9 hours	18	N
UTC-9D	Universal Time Coordinated - 9 hours (DST)	18	Y
UTC-9DA	Universal Time Coordinated - 9 hours (24 Hr Adjusted)	18	A
UTC-8	Universal Time Coordinated - 8 hours	16	N
UTC-8D	Universal Time Coordinated - 8 hours (DST)	16	Y
UTC-8DA	Universal Time Coordinated - 8 hours (24 Hr Adjusted)	16	A
UTC-7	Universal Time Coordinated - 7 hours	14	N
UTC-7D	Universal Time Coordinated - 7 hours (DST)	14	Y
UTC-7DA	Universal Time Coordinated - 7 hours (24 Hr Adjusted)	14	A

CODE	NAME	TIMEZONE	DST
UTC-6	Universal Time Coordinated - 6 hours	12	N
UTC-6D	Universal Time Coordinated - 6 hours (DST)	12	Y
UTC-6DA	Universal Time Coordinated - 6 hours (24 Hr Adjusted)	12	A
UTC-5	Universal Time Coordinated - 5 hours	10	N
UTC-5D	Universal Time Coordinated - 5 hours (DST)	10	Y
UTC-5DA	Universal Time Coordinated - 5 hours (24 Hr Adjusted)	10	A
UTC-4	Universal Time Coordinated - 4 hours	8	N
UTC-4D	Universal Time Coordinated - 4 hours (DST)	8	Y
UTC-4DA	Universal Time Coordinated - 4 hours (24 Hr Adjusted)	8	A
UTC-3	Universal Time Coordinated - 3 hours	6	N
UTC-3D	Universal Time Coordinated - 3 hours (DST)	6	Y
UTC-3DA	Universal Time Coordinated - 3 hours (24 Hr Adjusted)	6	A
UTC-2	Universal Time Coordinated - 2 hours	4	N
UTC-2D	Universal Time Coordinated - 2 hours (DST)	4	Y
UTC-2DA	Universal Time Coordinated - 2 hours (24 Hr Adjusted)	4	A
UTC-1	Universal Time Coordinated - 1 hour	2	N
UTC-1D	Universal Time Coordinated - 1 hour (DST)	2	Y
UTC-1DA	Universal Time Coordinated - 1 hour (24 Hr Adjusted)	2	A
UTC	Universal Time Coordinated	0	N
UTCD	Universal Time Coordinated (DST)	0	Y
UTCDA	Universal Time Coordinated (24 Hr Adjusted)	0	A
UTC+1	Universal Time Coordinated + 1 hour	46	N
UTC+1D	Universal Time Coordinated + 1 hour (DST)	46	Y
UTC+1DA	Universal Time Coordinated + 1 hour (24 Hr Adjusted)	46	A
UTC+2	Universal Time Coordinated + 2 hours	44	N
UTC+2D	Universal Time Coordinated + 2 hours (DST)	44	Y
UTC+2DA	Universal Time Coordinated + 2 hours (24 Hr Adjusted)	44	A
UTC+3	Universal Time Coordinated + 3 hours	42	N

CODE	NAME	TIMEZONE	DST
UTC+3D	Universal Time Coordinated + 3 hours (DST)	42	Y
UTC+3DA	Universal Time Coordinated + 3 hours (24 Hr Adjusted)	42	A
UTC+4	Universal Time Coordinated + 4 hours	40	N
UTC+4D	Universal Time Coordinated + 4 hours (DST)	40	Y
UTC+4DA	Universal Time Coordinated + 4 hours (24 Hr Adjusted)	40	A
UTC+5	Universal Time Coordinated + 5 hours	38	N
UTC+5D	Universal Time Coordinated + 5 hours (DST)	38	Y
UTC+5DA	Universal Time Coordinated + 5 hours (24 Hr Adjusted)	38	A
UTC+6	Universal Time Coordinated + 6 hours	36	N
UTC+6D	Universal Time Coordinated + 6 hours (DST)	36	Y
UTC+6DA	Universal Time Coordinated + 6 hours (24 Hr Adjusted)	36	A
UTC+7	Universal Time Coordinated + 7 hours	34	N
UTC+7D	Universal Time Coordinated + 7 hours (DST)	34	Y
UTC+7DA	Universal Time Coordinated + 7 hours (24 Hr Adjusted)	34	A
UTC+8	Universal Time Coordinated + 8 hours	32	N
UTC+8D	Universal Time Coordinated + 8 hours (DST)	32	Y
UTC+8DA	Universal Time Coordinated + 8 hours (24 Hr Adjusted)	32	A
UTC+9	Universal Time Coordinated + 9 hours	30	N
UTC+9D	Universal Time Coordinated + 9 hours (DST)	30	Y
UTC+9DA	Universal Time Coordinated + 9 hours (24 Hr Adjusted)	30	A
UTC+10	Universal Time Coordinated + 10 hours	28	N
UTC+10D	Universal Time Coordinated + 10 hours (DST)	28	Y
UTC+10DA	Universal Time Coordinated + 10 hours (24 Hr Adjusted)	28	A
UTC+11	Universal Time Coordinated + 11 hours	26	N
UTC+11D	Universal Time Coordinated + 11 hours (DST)	26	Y
UTC+11DA	Universal Time Coordinated + 11 hours (24 Hr Adjusted)	26	A

DST Support in the US

Daylight Savings Time (DST) changes the clock the users see; for instance in the United States, the clock goes straight from 1:59:59AM to 3:00:00AM on the Spring DST day, and there are two 1:00:00AM hours in the Fall DST day. These are called "DST transitions". Standard time has no adjustments: every day is 24 hours. Interval data usage is measured by meters with internal clocks that may or may not adjust for DST, and as the interval data is processed it may or may not be "adjusted" to 24 hour days. Thus the start and stop times of a cut of interval data may be in one of several states:

- The cut times are on standard time, and that matches the user's time.
- The cut times are on standard time and overlap the Spring DST day, when the user goes on DST. In this case, the cut stop time (the time from the meter of the last interval) is one hour before the time the user would indicate. For example, the cut stop is 8:59:59AM, but the user sees the last interval ending at 9:59:59AM.
- The cut times are on standard time and overlap the Fall DST day, when the user goes off DST. In this case, the cut start time (the time from the meter of the first interval) is one hour before the time the user would indicate. For example, the cut start is 8:00:00AM, but the user sees the first interval starting at 9:00:00AM.
- The cut times are on standard time and fall entirely within the user's DST period. In this case all cut times are one hour before the time the user would indicate.
- The cut times reflect DST - one hour less in the Spring, one hour more in the Fall, and changes as the user changes. All cut and interval times match what the user sees.
- The cut times reflect DST - one hour less in the Spring, one hour more in the Fall, and the user stays on Standard time.
- The cut is a DST cut adjusted to 24 hours per day. If the cut overlaps the April transition, intervals with zero value and missing status is inserted to make up the hour, if the cut overlaps the October transition, corresponding intervals are averaged to get one hour from the two 1:00AM hours.

Oracle Utilities Billing Component supports interval data reads where the meter's clock is set differently from the account's clock. This occurs because the meter is set to Standard time, while the account may be on Daylight Savings Time. In addition, 3rd party data may come in where its times are based on a timezone different form the account's that uses the data. It is also possible that one account may have two meters that are in different timezones.

When Oracle Utilities Billing Component reports a date and time that is based on interval data (such as the date and time of a maximum), the date and time is reported as seen by the account, and not by the meter.

When Oracle Utilities Billing Component computes a real time price by multiplying the price data (possibly from a 3rd party) by the usage data, the corresponding intervals are aligned such that the price corresponds to the usage in the same real time.

All interval data is stored with the time adjusted to match the accounts that use it. An option on the interval data import dialog and command line (see **Importing Interval Data** on page 3-5 on the *Data Manager User's Guide*) allows the user to specify the timezone of the input file and the timezone of its target accounts.

Important Note

When loading or creating cuts whose Start Time is within the Fall DST transition hour (01:00:00 through 01:59:59 in the United States), the second of the repeated hours is used as the start of the cut. This applies to ALL INTDxxx Rules Language functions.

Unit-of-Measure Rates and Quantities

This section describes how different types of Units-of-Measure (UOMs) are supported in the Oracle Utilities Rules Language, including:

- **UOM Categories**
- **UOM Mapping**
- **Mapping Data**
- **Rules Language Functions**

UOM Categories

There are three categories of interval data UOMs: those that measure a rate, those that measure a quantity, and those that measure something else. For example:

- **kW** and **GPM** measure rates (the flow of energy per hour or gallons per minute),
- **kWh** and **Gallons** measure quantities (energy used or gallons)
- Temperature is something else.

In the UOM table in the Oracle Utilities Data Repository, rate UOMs have an AGGREGATE value of 'A' (average), while quantity UOMs have an AGGREGATE value of 'T' (total).

Many UOMs come in complementary pairs, one is the measure of a quantity and the other is a measure of the rate (quantity per time unit). It is possible to convert one to the other, and to express maximums and total of one in terms of the other.

UOM Mapping

Oracle Utilities has defined the mappings for the standard UOMs described in the Oracle Utilities Load Analysis documentation. The mapping must relate the rate UOM, the quantity UOM, and the time unit, where rate = quantity/time unit. The time unit will be measured in seconds. From the example above:

Rate	Quantity	Time Unit (Seconds)
KW	KWh	3600
GPM	Gallons	60

Interval Data Cut Conversion

The user may wish to convert a cut of one UOM into either its rate UOM or its quantity UOM. If a cut whose UOM is a quantity is converted to a quantity UOM, it is unchanged, and similarly for a rate to rate conversion. If a conversion is attempted on a UOM that is a "something else" UOM and is not mapped, it will be unchanged.

The following two cut conversions are supported in the Rules Language. When a cut is converted its UOM must also be changed.

Quantity to Rate

If a quantity UOM is converted to its complementary rate UOM, each value is divided by the seconds per interval and then multiplied by the seconds per time unit:

Rate = (Quantity per interval / seconds per interval) * seconds per time unit

If there is more than one rate UOM for the quantity UOM, the rate with the largest time unit will be used.

Rate to Quantity

If a rate UOM is converted to its complementary quantity UOM, each value is divided by the seconds per time unit and then multiplied by the seconds per interval:

Quantity per interval = (Rate / seconds per time unit) * seconds per interval

Interval Data Value Conversion

There are several values derived from a cut that can also be converted from one UOM to another. The two of interest are the maximums and the total (for rate to quantity - the rate total is not needed). Both of these are converted using the formulas above. It is also possible to retrieve the cut maximums as either a rate or a quantity and the total as a quantity.

Mapping Data

An internal, hard-coded table contains the mapping information. It contains three fields per "record":

- **UOMCODERATE** (VARCHAR(64)): Reference to UOM table. This is the key.
- **UOMCODEQUANTITY** (VARCHAR(64)): Reference to UOM table.
- **SECONDSPERUNITINTEGER** - Not NULL

There can be only one quantity for a given rate, however, several rates may correspond to the same quantity. This mapping does not support time units greater than a day.

The current mappings are:

UOM Code - Rate	UOM Code - Quantity	Seconds Per Unit
02	01	3600
22	03	3600
23	04	3600
52	51	3600
72	53	3600
73	54	3600
60	69	60
63	70	60
64	70	1
77	70	3600
82	81	3600
84	83	3600

Rules Language Functions

A number of interval data functions in the Rules Language can be used to convert UOM rates and quantities.

INTDVALUE

The **INTDVALUE Function** supports the following attributes:

- **QUANTITY_TOTAL**: Total converted to quantity if rate UOM, else total
- **QUANTITY_MAX#**: Maximum converted to quantity if rate UOM, else maximum. # is 1 - 10, or omitted, and follows the same rules as in MAXIMUM#.
- **RATE_MAX#**: Maximum converted to rate if quantity UOM, else maximum. # is 1 - 10, or omitted, and follows the same rules as in MAXIMUM#.

QUANTITY_TOTAL is the same as ENERGY for UOMs 01 and 02. RATE_MAX is the same as KW_MAXIMUM for the same UOMs. MAX, QUANTITY_MAX and RATE_MAX are all equal for UOMs that to not appear in the UOMRATEQUANTITY table.

These attributes are also available in the usual HNDL.ATTRIBUTE form. See **Interval Data Reference Values and Attributes** on page 7-3 for more information.

INTDSCALAROP

The **INTDSCALAROP Function** supports two operations: QUANTITY and RATE. Neither will use the scalar value parameter. If the operation is QUANTITY the handle will be converted to its quantity UOM, if the parameter is RATE it will be converted to its rate UOM. If the handle's UOM is not mapped or is the same as the operation UOM, the handle will be copied unchanged.

Interval Data Browser

The interval data browser (see **Browsing Interval Data in the Interval Database** on page 6-7 on the *Data Manager User's Guide*) displays additional information related to the UOM of a cut, if the cut's UOM is stored in the Oracle Utilities Data Repository. The UOM information shown may include:

- **UOM Code, Name and Unit**: These values as they appear in the database.
- **UOM Aggregate**: The default operation when scaling or aggregating data within one cut (A (Average), M (Maximum) or T (Total).
- **UOM Totalize**: The default operation when performing an INTDLOADUOM that "combines" several channels (A (Average), M (Maximum) or T (Total).
- **UOM Rate/Quantity**: As defined via the UOMRATEQUANTITY table (Q (Quantity), R (Rate) or O (Other).
- **UOM Related UOM Code**: If the UOM is a Rate this will be the corresponding Quantity UOM, and vice versa.
- **UOM Seconds Per Unit**: If the UOM is a Rate this will be seconds per unit quantity.

Loading Interval Data

This section describes loading interval data in the Rules Language, including:

- **INTDLOAD Functions**
- **Loading Overlapping Cuts**
- **Loading Partial Intervals**
- **Notes On Loading Interval Data**

INTDLOAD Functions

The **INTDLOAD Functions** are used to load standard interval data for use in calculations in the Rules Language. For instance, you might load an interval data cut that represents an account's usage for the current bill period in order to derive the billing determinants used in calculating that account's bill.

There are many different **INTDLOAD Functions**, each used for different purposes. The table below outlines how and when each is used. Full descriptions of these functions can be found in **Chapter 9**: **Interval Data Function Descriptions** in the *Oracle Utilities Rules Language Reference Guide*.

To load data based on this:	Use:
A specified bill determinant or recorder,channel for the current bill period	**INTDLOAD Function** on page 9-34
A specific interval data cut, based on recorder,channel, and start time	**INTDLOADACTUALCUT Function** on page 9-35
A specified bill determinant or recorder,channel over a specified date range	**INTDLOADDATES Function** on page 9-36
A specified bill determinant or recorder,channel for a specified number of historical bill periods	**INTDLOADHIST Function** on page 9-38
A list of recorder,channels	**INTDLOADLIST Function** on page 9-39
A list of recorder,channels over a specified date range	**INTDLOADLISTDATES Function** on page 9-40
All recorder,channels (or a list of recorder,channels) that are billed and record kW or kWh	**INTDLOADLISTENERGY Function** on page 9-41
A list of recorder,channels for a specified number of historical bill periods	**INTDLOADLISTHIST Function** on page 9-42
A specified recorder,channel	**INTDLOADRELATEDCHANNEL Function** on page 9-43
All channels belonging to an Aggregation Group	**INTDLOADSP Function** on page 9-44

To load data based on this:	Use:
A recorder,channel for a specified start and stop time [from the Interval Data Staging tables)	**INTDLOADSTAGING Function** on page 9-46
A specified Unit-of-Measure (UOM) for the current bill period	**INTDLOADUOM Function** on page 9-47
A specified Unit-of-Measure (UOM) over a specified date range	**INTDLOADUOMDATES Function** on page 9-48
A specified Unit-of-Measure (UOM) for a specified number of historical bill periods	**INTDLOADUOMHIST Function** on page 9-49

Note: Page numbers in the table refer to the Oracle Utilities Rules Language Reference Guide.

About Cut Start and Stop Times

This applies to all INTDLOADxxx functions, except INTDLOADLISTENERGY. The functions use the cuts whose start and stop time are closest to the bill period start and stop time, as specified for the billing cycle code that applies to the account. If the account itself has a start or stop time, that takes precedence. If the account's channels (or the list's channels for INTDLOADLIST) have different start and stop times, the program automatically applies the earliest start and the earliest stop among the channels.

Accessing Multiple Interval Databases from the same Rate Schedule

You can load interval data from more than one interval data source in the same rate schedule using either INTDOPEN or INTDLOADxxx. When interval data has been opened or loaded in the rate form, you can use other functions as normal on the data.

Using INTDOPEN:

The **INTDLOAD Function** enables you to open multiple Interval Data Databases from a single rate form.

Using INTDLOADxxx functions:

To load interval data from a rate form, use the following format:

```
<interval_data_reference> = INTDLOADxxx("<file and path name to interval
database>;<determinant_identifier|recorder,channel>");
```

Where:

- <file and path name to interval database> is a string containing the absolute path and file name of the Interval Database, followed by a semi-colon, and the <determinant_identifier|recorder,channel>. The string can have no spaces, but can name any supported file type. The interval database file can be in any of the following formats:
  - Enhanced Oracle Utilities Input/Output Format (*.lse)
  - Oracle Utilities Standard Format (*.inp)
  - Oracle Utilities Comma Separated Format (*.csv)
  - Oracle Utilities Standard XML Format (*.xml)

This applies to all INTDLOADxxx functions.

Example: Your Interval Data Options (see the *Data Manager User's Guide*) are set to retrieve interval data from: c:\lodestar\user\getwell.lse. Another interval data file is located at:

d:\lodestar\user\getwell2.lse, and you need to load data from that file also. You could load interval data from the second file with the following statement:

```
HNDL_2 = INTDLOAD ("d:\lodestar\user\getwell2.lse;1700,1");
```

Another example might look like this:

```
// Load test data for the current bill period
CUTNAME = "d:\comndata\testfile.lse;RECORDER_TEST,1";
HNDL = INTDLOAD(CUTNAME);
```

Loading Interval Data from Relational Database Tables

You can also load interval data from multiple relational database tables in the Oracle Utilities Data Repository using the following functions:

- **INTDLOAD Function**
- **INTDLOADDATES Function**
- **INTDLOADHIST Function**
- **INTDLOADLIST Function**
- **INTDLOADLISTHIST Function**
- **INTDLOADLISTDATES Function**

To load interval data from the relational database, use the following format:

```
<interval_data_reference> = INTDLOADxxx("[QUAL/<alternate_qualifier>;]RDB/
<alternate_table>;<recorder,channel>");
```

Where:

- <alternate_qualifier> is a string containing the name of a alternate database qualifier containing the interval data to be loaded.
  - When an alternate qualifier is specified, all database calls for the function will be directed at the specified qualifier, with one exception. In the case of INTDLOADLISTxxx() functions, the list query alone will be fetched from the original qualifier.
  - The meta-data of the alternate qualifier **must** be the same as the original qualifier.
  - When using an alternate qualifier and processing in the context of an Account (such as when running billing via Oracle Utilities Billing Component), the account **must** be present in both the qualifiers.
- <alternate_table> is a string containing the name of a table with the same schema as the LSCHANNELCUTHEADER table. The name of this table must begin with the letters "LSC". Also, this table must have two child tables, one with column VALUESCODES and one with column SEQUENCE, and which have the same schema as the LSCHANNELCUTDATA and LSCHANNELCUTEDITS tables, respectively. This table must also have one parent table, the CHANNEL table, which in turn also has one parent table, the RECORDER table.
- <recorder,channel> is an identifier for a particular recorder-ID, channel-number in the Interval Database.

For example:

```
// Header data is stored in LSCHANNELHEADERVERS table
CUTNAME = "RDB/LSCHANNELHEADERVERS;RECORDER_TEST,1";
HNDL = INTDLOADDATES(CUTNAME, BILL_START, BILL_STOP);
```

Saving Data

You can also save data to an alternate qualifier and/or table using the SAVE TO CHANNEL statement, using the following format:

```
SAVE <HNDL> TO CHANNEL "[QUAL/<alternate_qualifier>;]RDB/
<alternate_table>;<recorder,channel>");
```

Where:

- <HNDL> is the interval data handle you wish to save.
- <alternate_qualifier> is a string containing the name of a alternate database qualifier containing the interval data to be loaded (see above).
- <alternate_table> is a string containing the name of a table with the same schema as the LSCHANNELCUTHEADER table. The name of this table must begin with the letters "LSC" (see above).
- <recorder,channel> is an identifier the recorder-ID, channel-number you wish to save the data to.

For example:

```
// Load data from LSCHVERS table
HNDL = INTDLOADDATES("RDB/LSCHVERS;TEST,1", BILL_START, BILL_STOP);
// Save data to LSCHVERS2 table in PRICING qualifier
SAVE HNDL TO CHANNEL "QUAL/PRICING;RDB/LSCHVERS2;TEST,1";
```

Deleting Data

You can also delete interval data from an alternate qualifier and/or table using the **INTDDELETE Function**, using the following format:

```
<interval_data_reference> = INTDDELETE("[QUAL/<alternate_qualifier>;]RDB/
<alternate_table>;<recorder,channel>");
```

Where:

- <alternate_qualifier> is a string containing the name of a alternate database qualifier containing the interval data to be loaded (see above).
- <alternate_table> is a string containing the name of a table with the same schema as the LSCHANNELCUTHEADER table. The name of this table must begin with the letters "LSC" (see above).
- <recorder,channel> is an identifier the recorder-ID, channel-number you wish to delete.

For example:

```
// Save data to LSCHVERS2 table in PRICING qualifier
SAVE HNDL TO CHANNEL "QUAL/PRICING;RDB/LSCHVERS2;TEST,1";
// Delete cut "Test,1" from LSCHVERS table
DEL_HNDL = INTDDELETE("RDB/LSCHVERS;TEST,1", BILL_START, BILL_STOP);
```

INTDLOADEX Functions

The INTDLOADEX functions are used to load interval data for use in calculations in the Rules Language from Enhanced Interval Data table. For instance, you might load an interval data cut that represents an account's usage for the current bill period in order to derive the billing determinants used in calculating that account's bill.

There are many different INTDLOADEX functions, each used for different purposes. The table below outlines how and when each is used. Full descriptions of these functions can be found in **Chapter 9**: **Interval Data Function Descriptions** in the *Oracle Utilities Rules Language Reference Guide*.

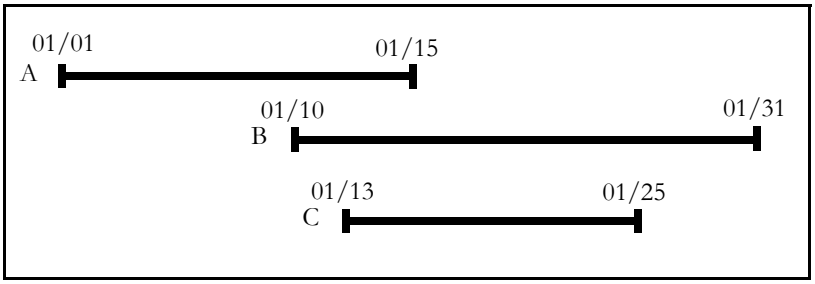
To load data based on this:	Use:
A specific interval data cut from a specified Enhanced Interval Data table	**INTDLOADEXACTUAL Function** on page 9-88
A specific interval data cut from an Enhanced Interval Data Versioning table	**INTDLOADEXCUT Function** on page 9-89
A user-specified date range from a specified Enhanced Interval Data table.	**INTDLOADEXDATES Function** on page 9-90
A specified bill determinant for the current bill period from a specified Enhanced Interval Data table	**INTDLOADEX Function** on page 9-93
A list of parent records with related interval data in a specified Enhanced Interval Data table	**INTDLOADEXLIST Function** on page 9-94
A list of parent records for a user-specified date range with related interval data in a specified Enhanced Interval Data table	**INTDLOADEXLISTDATES Function** on page 9-95
Interval data for the related meter specified in the Oracle Utilities Meter Data Management Meter table from a specified Enhanced Interval Data table	**INTDLOADEXRELATEDCHANNEL Function** on page 9-96

See **Working with Enhanced/Generic Interval Data** on page 7-30 for more information about working with interval data stored in enhanced/generic interval data tables.

Loading Overlapping Cuts

On occasion, cuts belonging to the same recorder,channel stored in the Oracle Utilities Data Repository may overlap. That is, the start time of one cut is earlier than the stop time of another cut. When interval data functions load cuts from the database, they combine overlapping cuts that fall within the specified start and stop times and create a handle that comprises portions of all the overlapping cuts. Where overlaps occur, the Rules Language uses the cut with the **latest Start Time**.

For example suppose the following three cuts for the same recorder,channel ('1700,1') were stored in the database with the same timestamp:



Cut A

- Start Time: 01/01/2000 00:00:00
- Stop Time: 01/15/2000 23:59:50

Cut B

- Start Time: 01/10/2000 00:00:00
- Stop Time: 01/31/2000 23:59:50

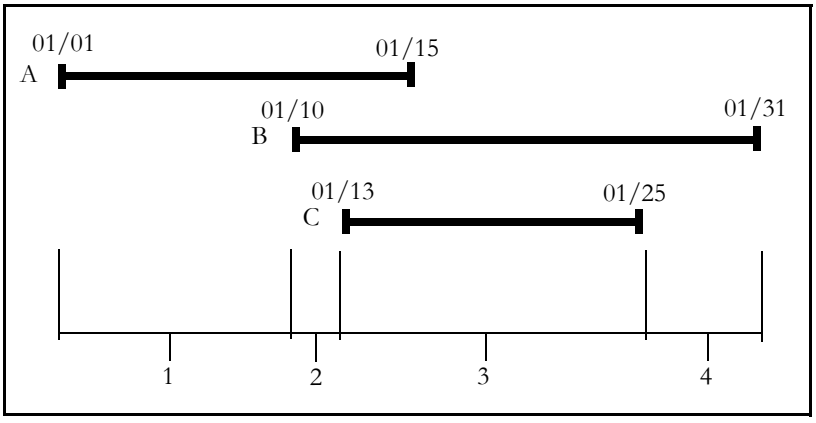
Cut C

- Start Time: 01/13/2000 00:00:00
- Stop Time: 01/25/2000 23:59:59

If you load this recorder,channel for the entire month of January 2000 using the **INTDLOADDATES Function**, as follows:

```
HNDL = INTDLOADDATES ('1700,1', "01/01/2000 00:00:00", "01/31/2000
23:59:59");
```

the handle returned would comprise:



1. 01/01/2000 00:00:00 through 01/09/2000 23:59:59 from Cut A

2. 01/10/2000 00:00:00 through 01/12/2000 23:59:59 from Cut B
3. 01/13/2000 00:00:00 through 01/25/2000 23:59:59 from Cut C
4. 01/26/2000 00:00:00 through 01/31/2000 23:59:59 from Cut B

Loading Overlapping Cuts Based on Timestamp

By default, the Rules Language uses only the start time to select which cut to load. However, the Rules Language can also load cuts based on a combination of the **latest Start Time** and **latest Timestamp**. To enable this option, include the following parameter in the LODESTAR.CFG file:

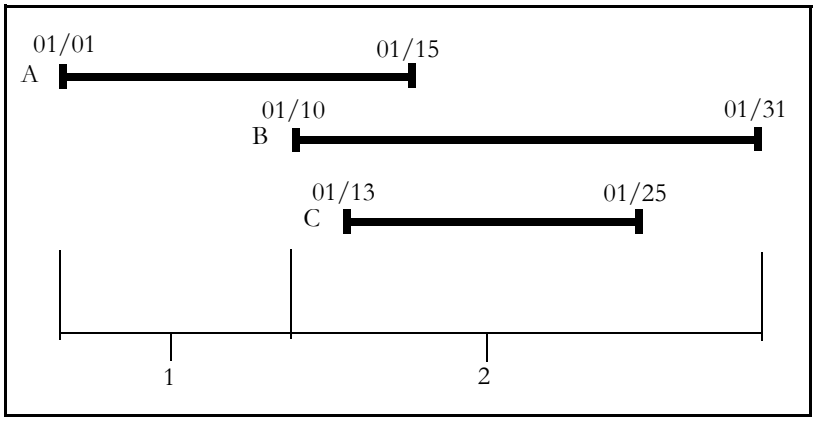
```
INTDJOINTIMESTAMP
```

See **LODESTAR.CFG** on page 2-2 of the *Oracle Utilities Energy Information Platform Configuration Guide* for more information about the LODESTAR.CFG file.

For example, if Cut B in the previous example had a timestamp later than cuts A and C and you load this recorder,channel for the entire month of January 2000 using the **INTDLOADDATES Function**, as follows:

```
HNDL = INTDLOADDATES ('1700,1', "01/01/2000 00:00:00", "01/31/2000
23:59:59");
```

the handle returned would comprise:



1. 01/01/2000 00:00:00 through 01/09/2000 23:59:59 from Cut A
2. 01/10/2000 00:00:00 through 01/31/2000 23:59:59 from Cut B

In this case, because Cut B has a later timestamp than Cut C, the Rules Language loads it and ignores Cut C entirely.

Loading Partial Intervals

Partial intervals refer to situations where an interval data start and/or end cut boundary does not land on an even 15, 30, or 60 minute boundary. This is because the stop time is the time the meter was read. For example, a partial interval cut may end at 12:41 instead of 12:30 or 12:45.

Data Manager Loading/Extraction

The interval data import utilities (see **Importing Interval Data** on page 3-5 on the *Data Manager User's Guide*) allow a partial interval cut to be loaded into the interval data database as is, without adjusting the cut boundaries. For example: an interval data cut which has a start and end read of 07/01/98 01:28– 07/30/98 12:41 is loaded and stored with the same exact reads.

The interval data import utilities also allow a partial interval cut to be extracted from the interval data database as is, without adjusting the cut boundaries.

The interval data browser (see **Browsing Interval Data in the Interval Database** on page 6-7 on the *Data Manager User's Guide*) displays the partial interval data cut information as is, without adjust the boundaries or meter readings.

Oracle Utilities Billing Component Calculation Process

Though partial interval cuts are loaded into the interval data database as is, Oracle Utilities Rules Language calculation processes still use full intervals in processing. Assume the following interval data cut is involved in calculation:

- **Start Reading**: 3425
- **Stop Reading**: 5678
- **Start Time**: 02/01/1998 00:41
- **Stop Time:** 03/02/1998 06:24

Oracle Utilities Billing Component should follow its current rules for determining the start and end of the bill period, except,

- If the bill period end date/time happens to land on a partial interval, the bill period end will become the previous whole interval. In the case above, the bill period end time will be 03/02/1998 06:15 for 15 minute intervals and 06:00 for 30 minute intervals. This applies to aggregated handles as well.
- If it is the first billing for the account, and the billing period start date/time happens to land on a partial interval, Oracle Utilities Billing Component sets the bill period start date/time to the previous whole interval. In the example above, the bill period start date/time will be 02/01/1998 00:30. This applies to aggregated handles as well.

The interval data functions that reference interval values with the adjusted bill period start date/time recognize the first interval which starts (not ends) on 00:41, and assumes zeros for the missing minutes from 00:30 to 00:41. This applies to the very first billing, since subsequent ones use the left over partial data from the previous cut. In this example, assume that the previous cut ends at 02/01/1998 00:40. Then its last interval would have been dropped from its billing, so must be added into the next month's data.

If an interval data function returns a date/time for an interval, such as finding the MAX_KW, the partial interval date/time will be returned, as it is stored in the interval data database. For example, the MAX_KW function may return a value of 500 at date/time 02/15/98 12:41. This only applies when the value is the first one. The start time of any other interval will be on an interval boundary.

Notes On Loading Interval Data

- When one of the **INTDLOAD Functions** retrieves interval data from more than one cut, it "joins" them together. The reading information will be kept only if the meter multipliers and offsets match, and the stop reading of one cut is the same or one less than the start reading of the next cut.
- For **INTDLOAD Functions** that use a date, the date and times are usually given from the first second of an interval through the last second of an interval (intervals start on even SPI or IPH boundaries, and end one second before the next interval). However, if the stop time specified is the start second of an interval, only the data up through the previous interval is loaded. For example:

  ```
  INTDLOADDATES(KWH, "01/01/1999 8:00:00", "01/31/1999 8:00:00")
  ```

  is the same as

  ```
  INTDLOADDATES(KWH, "01/01/1999 8:00:00", "01/31/1999 7:59:59")
  ```

- Interval values that exceed a maximum, or fall between zero and a minimum, are set to the maximum or minimum, respectively, and their status codes downgraded to 'L'. See the INTDVALUEMAX and INTDVALUEMIN configuration parameters in **Chapter 2: Configuration Files** in the *Oracle Utilities Energy Information Platform Configuration Guide* for more information.
- When loading interval data with a Power Flow Direction (DC_FLOW) of R (Received), interval values and totals are inverted (multiplied by negative 1 (-1)).

Creating Interval Data Masks

This section describes how to create interval data masks using the Rules Language, including:

- **Overview**
- **Interval Data Mask Functions**
- **Interval Data Mask Operator Rules**

Overview

An interval data mask is a handle whose values are all zero or one, as opposed to a data handle which may have any value. A zero value means the interval is excluded from the handle, a one means it is included. Masks can be used to remove values from a data handle, or can be combined with other masks to include or exclude additional intervals. Note that masks can be created by mask functions (outlined below), by an interval data function that operates on an existing mask, or by a divide operation that creates a handle with all zeros and ones.

Interval Data Mask Functions

The **INTDCREATEMASK Functions** are used to create interval data masks for use in calculations in the Rules Language. For instance, you might create a mask that includes all intervals that correspond to a specified time-of-use period or a override.

There are many different **INTDCREATEMASK Functions**, each used for different purposes. The table below outlines how and when each is used. Full descriptions of these functions can be found in **Chapter 9**: **Interval Data Function Descriptions** in the *Oracle Utilities Rules Language Reference Guide*.

To create a mask where:	Use:
Interval values in the new handle are 1 for the entire day if the original handle has a 0 value for any interval during that day, or 0 if the original has all non-zero values for the corresponding day.	**INTDCREATEDAYMASK Function** on page 9-12
Interval values are set to the factor value in effect at that time.	**INTDCREATEFACTORMASK Function** on page 9-13
Interval values in the new handle are 1 if the corresponding value in the original handle is 0, or 0 if the original has a non-zero value.	**INTDCREATEMASK Function** on page 9-15
Interval values are based on the effective period for an override, with each override period automatically extended to start and end at midnight (or the day start).	**INTDCREATEOVERRIDEDAYMASK Function** on page 9-16
Interval values are based on the effective period for an override.	**INTDCREATEOVERRIDEMASK Function** on page 9-17
Interval values are based on comparison of status codes.	**INTDCREATESTATUSCODEMASK Function** on page 9-18

To create a mask where:	Use:
Interval values that fall within a user-specified time of use period are distinguished from those that fall outside the period. The interval values in the period may be set to 1 (and all others to 0), or to their actual value (and all others to 0). The resulting "mask" can be used in TOU computations.	**INTDCREATETOUPERIOD Function** on page 9-19

Note: Page numbers in the table refer to the Oracle Utilities Rules Language Reference Guide.

These functions returns a mask if:

1. The operation is one of "MASK", "REVERSE_MASK", "ZERO", "NON_ZERO" or "MISSING", or
2. The input interval data reference is a mask and the operation is one of the above, plus "VALUE" and "REVERSE_VALUE".

Otherwise an interval data handle is returned.

Interval Data Mask Operator Rules

If an operation is between two interval data handles and the right one is a mask, the following rules apply:

a. If the left handle is also a mask, the result is a mask and the math operations are:

Operation	Rule
+	A union ("OR") of the masks; the value is '1' if either corresponding value is '1'; otherwise it is '0'.
-	The value is '1' if the left value was '1' and the right is '0'; otherwise, it is '0' (it removes "on" intervals in the right mask from the left mask).
*,/	An "AND" of the masks; the value is '1' if both corresponding values are '1'; otherwise it is '0'.

b. If the left handle is a simple interval data handle, the result is a similar handle and the math operations are:

Operation	Rule
+,*,/	The value is the left value if right is '1'; otherwise it is '0' (it leaves only "on" intervals in the right mask in the handle).
-	The value is the left value if right is '0'; otherwise it is '0' (it removes "on" intervals in the right mask from the handle).

Other Interval Data Operations

This section describes how you work with previously loaded interval data (handles and masks), including:

- **Interval Data Functions**
- **Scalar and Block Operations**

Interval Data Functions

Interval data functions are used to perform specific operations on loaded interval data (handles and masks). For example, you might need to scale an aggregated handle up or down when performing a final settlement calculation using Oracle Utilities Load Profiling and Settlement.

There are many different interval data functions, each used for different purposes. The table below outlines how and when each is used. Full descriptions of these functions can be found in **Chapter 9**: **Interval Data Function Descriptions** in the *Oracle Utilities Rules Language Reference Guide*.

If you want to:	Use:
Add a user-defined attribute to an interval data handle.	**INTDADDATTRIBUTE Function** on page 9-2
Add a validation message to an interval data handle.	**INTDADDVMSG Function** on page 9-3
Perform a block operation on the interval data values of one handle using the corresponding values of another handle.	**INTDBLOCKOP Function** on page 9-4
Perform a block operation on the interval data values of one handle using the corresponding values of another handle. This is similar to the INTDBLOCKOP function, but allows use of non-aligned interval data handles.	**INTDBLOCKOPNA Function** on page 9-6
Close an interval data file that was opened using the INTDOPEN function.	**INTDCLOSE Function** on page 9-8
Count the number of intervals in the handle.	**INTDCOUNT Function** on page 9-9
Count the number of intervals in an interval data handle that match a specified status code.	**INTDCOUNTSTATUSCODE Function** on page 9-10
Create an interval data handle based on user-specified start time, stop time, and SPI.	**INTDCREATEHANDLE Function** on page 9-14
Delete one or more cuts from the Oracle Utilities Data Repository.	**INTDDELETE Function** on page 9-21

If you want to:	Use:
Delete an interval data cut from a specified Enhanced Interval Data table.	**INTDDELETEEX Function** on page 9-86
Get multiple custom and parent attributes of a specified enhanced interval data handle.	**INTDGETATTREXALL Function** on page 9-87
Examine an interval data handle for dips.	**INTDDIPTEST Function** on page 9-22
Export data in a handle to a file.	**INTDEXPORT Function** on page 9-23
Compare two handles to determine if they are considered equal.	**INTDISEQUAL Function** on page 9-27
Get the error code from the last interval data function call.	**INTDGETERRORCODE Function** on page 9-25
Get the error message from the last interval data function to use a specific interval data reference.	**INTDGETERRORMESSAGE Function** on page 9-26
Merge two interval data handles into one.	**INTDJOIN Function** on page 9-28
Open an interval data file.	**INTDOPEN Function** on page 9-51
Get a reference to the first record in an interval data file.	**INTDREADFIRST Function** on page 9-52
Get a reference to the next record in an interval data file.	**INTDREADNEXT Function** on page 9-53
Get the number of records in an interval data file.	**INTDRECCOUNT Function** on page 9-54
Release an interval data reference before completion of the rate form.	**INTDRELEASE Function** on page 9-55
Calculate the rolling average (or total) of interval values in a handle.	**INTDROLLAVG Function** on page 9-57
Calculate the rolling peak of interval values in a handle.	**INTDROLLPEAK Function** on page 9-58
Save an interval data handle to a specified Enhanced Interval Data table.	**INTDSAVEEX Function** on page 9-97
Save an interval data handle and its parent to specified Enhanced Interval Data tables.	**INTDSAVEEXP Function** on page 9-99
Set an attribute of a specified enhanced interval data handle	**INTDSETATTREX Function** on page 9-101

If you want to:	Use:
Set multiple custom and parent attributes of a specified enhanced interval data handle.	**INTDSETATTREXALL Function** on page 9-102
Perform a scalar operation on each interval in a handle.	**INTDSCALAROP Function** on page 9-59
Aggregate values in a handle according to user-specified criteria.	**INTDSCALE Function** on page 9-61
Set attributes of a specified handle.	**INTDSETATTRIBUTE Function** on page 9-63
Changes the DST Participant flag for a previously-loaded interval data handle, and optionally adjusts the handle's Start Time and Stop Time as needed.	**INTDSETDSTPARTICIPANT Function** on page 9-65
Set the status codes of all non-missing intervals in an existing handle.	**INTDSETSTRING Function** on page 9-66
Set an interval value of an interval data handle.	**INTDSETVALUE Function** on page 9-67
Change the status codes and/or values of intervals in a handle.	**INTDSETVALUESTATUS Function** on page 9-68
Shift the start time of an interval data handle.	**INTDSHIFTSTARTTIME Function** on page 9-70
Smooth gaps in an interval data handle.	**INTDSMOOTH Function** on page 9-71
Sort the values in an interval data handle.	**INTDSORT Function** on page 9-72
Examine an interval data handle for spikes.	**INTDSPIKETEST Function** on page 9-73
Get a subset of an interval data handle.	**INTDSUBSET Function** on page 9-74
Create a handle for a specified Time-of-Use Schedule and Holiday List from a previously loaded cut.	**INTDTOU Function** on page 9-75
Release a Time of Use reference set with INTDTOU before completion of the rate form.	**INTDTORELEASE Function** on page 9-76
Compute a user-specified summary value for a TOU handle that was created with the INTDTOU function.	**INTDTOUVALUE Function** on page 9-77

If you want to:	Use:
Compute a user-specified summary value for an interval data handle.	**INTDVALUE Function** on page 9-79
Get an attribute of a specified enhanced interval data handle	**INTDVALUEEX Function** on page 9-103

Note: Page numbers in the table refer to the Oracle Utilities Rules Language Reference Guide.

Scalar and Block Operations

This section outlines the rules for determining the status code of a handle created through scalar or block operations. When two handles are combined, the status code of each interval is based on the corresponding two original status codes. The rules are:

1. If both codes are '9' the result has code '9', else
2. If only one codes is '9' the result has code '7', else
3. The result has the worse of the two codes based on the comparison order (from highest to lowest): (space) A B C ... Z 0 1 2 ... 7 9

Working with Enhanced/Generic Interval Data

Working with interval data stored in enhanced/generic interval data tables via the Oracle Utilities Rules Language is done via a specific set of functions. These functions are documented in the **Enhanced Interval Data Functions** section of **Chapter 9**: **Interval Data Function Descriptions** in the *Oracle Utilities Rules Language Reference Guide*. This section outlines how you load, save, and delete enhanced/generic interval data using Oracle Utilities Rules Language.

Common Parameters

The Rules Language functions used with enhanced/generic interval data use a specific set of parameters that differ slightly from other interval data functions. These include:

- **<parent_identity>** - the identity of the parent record. This can be in the form of a string that contains the identity or a database identifier that contains the identity. This is used by all enhanced interval data functions.
- **<parent_stem>** - a stem identifier that contains the parent record, including all required columns. Used by the INTDSAVEEXP function.
- **<table_name>** - the name of the interval data table in which the data is stored. This is used by all enhanced interval data functions.
- **<category>** - the optional category code associated with the interval data. This can be in the form of a string that contains the category or a database identifier that contains the category. This is used by all enhanced interval data functions.

Loading Data from Enhanced/Generic Interval Data Tables

Loading data from enhanced/generic interval data tables can be performed using the following functions.

- **INTDLOADEX**: loads and totalizes all interval data for a user-specified determinant or parent record for the current bill period. See **INTDLOADEX Function** on page 9-93 in the *Oracle Utilities Rules Language Reference Guide* for more information.
- **INTDLOADEXACTUAL**: loads a specific interval data cut for a given start time. See **INTDLOADEXACTUAL Function** on page 9-88 in the *Oracle Utilities Rules Language Reference Guide* for more information.
- **INTDLOADEXCUT**: loads a specified historical version of an interval data cut for a specified parent. See **INTDLOADEXCUT Function** on page 9-89 in the *Oracle Utilities Rules Language Reference Guide* for more information.
- **INTDLOADEXDATES**: loads interval data for a user-specified date range from a specified Enhanced Interval Data table. See **INTDLOADEXDATES Function** on page 9-90 in the *Oracle Utilities Rules Language Reference Guide* for more information.
- **INTDLOADEXLIST**: totalizes the interval data stored in an enhanced interval data table for the current bill period for all parent records in a list. See **INTDLOADEXLIST Function** on page 9-94 in the *Oracle Utilities Rules Language Reference Guide* for more information.
- **INTDLOADEXLISTDATES**: totalizes the interval data stored in an enhanced interval data table for all parent records in a list over a specified time range. See **INTDLOADEXLISTDATES Function** on page 9-95 in the *Oracle Utilities Rules Language Reference Guide* for more information.
- **INTDLOADEXRELATEDCHANNEL**: loads the interval data for the meter related to the interval data reference's meter through the MDM Meter table. Used with Oracle Utilities Meter Data Management ONLY. See **INTDLOADEXRELATEDCHANNEL Function** on page 9-96 in the *Oracle Utilities Rules Language Reference Guide* for more information.

Note: You can use the INTDLOADEX, INTDLOADEXACTUAL, and INTDLOADEXDATES to load data from Staging tables. To load an enhanced interval data handle from a Staging table, specify the name of the Staging table in the <table_name> parameter.

Example: Weather Data table - INTDLOADEXACTUAL

Load the interval data cut for weather station WS_01 in Austin, TX (AUSTIN) with a start time of 06/01/2007 with a category of "FINAL" from the Weather Data (LSWEATHERDATA) table.

```
WEATHER_STATION = "WS_01,AUSTIN";
CATEGORY = "FINAL";
TABLE_NAME = "LSWEATHERDATA";
HNDL = INTDLOADEXACTUAL(WEATHER_STATION, CATEGORY, TABLE_NAME, "06/01/
2007");
```

Example: Weather Data table - INTDLOADEXDATES

Load the interval data cut for weather station WS_01 in Austin, TX (AUSTIN) with a category of "FINAL" from the Weather Data (LSWEATHERDATA) table for the month of January 2007:

```
WEATHER_STATION = "WS_01,AUSTIN";
CATEGORY = "FINAL";
TABLE_NAME = "LSWEATHERDATA";
TEMP_HNDL = INTDLOADEXDATES(WEATHER_STATION, CATEGORY, TABLE_NAME,
'01/01/2007', '01/31/2007 23:59:59');
```

OR

```
STARTDT = DATE('01/01/2007 00:00:00');
STOPDT = DATE('01/31/2007 23:59:59');
WEATHER_STATION = "WS_01,AUSTIN";
CATEGORY = "FINAL";
TABLE_NAME = "LSWEATHERDATA";
TEMP_HNDL = INTDLOADEXDATES(WEATHER_STATION, CATEGORY, TABLE_NAME,
STARTDT, STOPDT);
```

> **Note**: Even if a category is not used, the <category> parameter is required when using the INTDLOADEX functions. In this case, specify an empty string ("") for the <category>.

Working with Loaded Data

Once interval data stored in enhanced/generic interval data tables is loaded into memory in the Rules Language, the data can be operated on via any operation applicable to interval data, including other interval data functions (INTDBLOCKOP, INTDSCALAROP, INTDSETATTRIBUTE, INTDSETVALUE, INTDVALUE, etc.), interval data expressions, and other operations. Also, attributes of a loaded interval data handle can be set/obtained using a <stem>.<tail> identifier, where the <stem> is the interval data handle identifier, and <tail> is the interval data attribute to be obtained/set. For example, the following would return the SPI from the MKT_HNDL loaded in the above example:

```
MKT_HNDL_SPI = MKT_HNDL.SPI;
```

In the case of attributes derived from optional columns (see **Optional Columns** on page 11-14 in the *Energy Information Platform Configuration Guide*), the <tail> must be one of the Interval Data Attributes listed in the Optional Columns table. For example, the following would obtain the Total from the MKT_HNDL loaded in the above example:

```
TEMP_HNDL_TOTAL = TEMP_HNDL.TOTAL;
```

> **Note**: Calculated values (such as Minimum, Maximum, Total, and others) are calculated every time the interval data is loaded. Also, columns that store calculated values are automatically updated when the interval data is saved to the database.

In the case of attributes derived from custom columns (see **Custom Columns** on page 11-16 in the *Energy Information Platform Configuration Guide*), the <tail> must match the column name in the interval data table. For example, the following would set the Minimum Interval Value from the MKT_HNDL loaded in the above example to:

```
TEMP_HNDL_MIN_VAL = TEMP_HNDL.MIN;
TEMP_HNDL.MINIMUM = TEMP_HNDL_MIN_VAL;
```

> **Note**: Values for custom columns are NOT automatically updated when the interval data is edited and saved to the database. Custom attributes must be explicitly updated via the Rules Language. However, values stored in custom columns are loaded from the database when a handle is created in the Rules Language.

Enhanced/generic interval data handles have two additional properties that can be set/obtained via the Rules Language:

- **PARENTKEY**: a string that contains the identity of the handle's parent
- **CATEGORY**: a string that contains the handle's category

Both of these can be obtained using <stem>.<tail> identifiers, as well as with the INTDVALUE (to get these properties from a handle). To set these properties, you must use the INTDSETATTRIBUTE function.

Getting and Setting Custom and Parent Attributes

Retrieving and setting multiple custom and/or parent attributes can be performed using the following functions

- **INTDGETATTREXALL** - Retrieves custom and parent attributes from an enhanced interval data handle and assigns them to stem identifiers.
- **INTDSETATTREXALL** - Set multiple custom and parent attributes in an enhanced interval data handle using stem identifiers.

Example: Retrieve custom and parent attributes

Report the C1 and C2 custom columns, and P1 and P2 parent columns in the #SAVE_AR array.

```
FOR EACH I IN NUMBER 100
   CUSTOM.C1 = "C" + I;
   CUSTOM.C2 = I;
   PARENT.P1 = "Parent" + I;
   PARENT.P2 = "Parent2_" + I;
   RET = INTDSETATTREXALL (#SAVE_AR[I], CUSTOM, PARENT);
END FOR;

FOR EACH I IN NUMBER 100
   RET = INTDGETATTREXALL (#SAVE_AR[I], CUSTOM, PARENT);
   REPORT CUSTOM.C1;
   REPORT CUSTOM.C2;
   REPORT PARENT.P1;
   REPORT PARENT.P2;
END FOR;
```

Example: Setting multiple custom and parent attributes

Set the C1 and C2 custom columns, and P1 and P2 parent columns in the #SAVE_AR array.

```
FOR EACH I IN NUMBER 100
   CUSTOM.C1 = "C" + I;
   CUSTOM.C2 = I;
   PARENT.P1 = "Parent" + I;
   PARENT.P2 = "Parent2_" + I;
   RET = INTDSETATTREXALL (#SAVE_AR[I], CUSTOM, PARENT);
END FOR;
```

Saving Data to Enhanced/Generic Interval Data Tables

Saving interval data to enhanced/generic interval data tables can be performed using the following functions:

- **INTDSAVEEX** - Saves an interval data handle. See **INTDSAVEEX Function** on page 9-97 in the *Oracle Utilities Rules Language Reference Guide* for more information.

  Note: To save an enhanced interval data handle to a Staging or Reporting table, specify the name of the Staging or Reporting table in the <table_name> parameter.

- **INTDSAVEEXP** - Saves an interval data handle and its parent. See **INTDSAVEEXP Function** on page 9-99 in the *Oracle Utilities Rules Language Reference Guide* for more information.

  Note: When saving parent records using INTDSAVEEXP, you must use actual UIDs when saving values for UID columns.

Example: Weather Data table - INTDSAVEEX

Save the interval data in TEMP_HNDL to Weather Station WS_01 in Austin, TX (AUSTIN) with a category of "FINAL" to the Weather Data (LSWEATHERDATA) table.

```
WEATHER_STATION = "WS_01,AUSTIN";
CATEGORY = "FINAL";
TABLE_NAME = "LSWEATHERDATA";
SAVE_HNDL = INTDSAVEEX(WEATHER_STATION, CATEGORY, TABLE_NAME,
TEMP_HNDL);
```

Example: Weather Data table - INTDSAVEEX (Reporting Table)

Save the interval data in TEMP_HNDL to Weather Station WS_01 in Austin, TX (AUSTIN) with a category of "FINAL" to the Weather Data Reporting table (used to report on interval data).

```
WEATHER_STATION = "WS_01,AUSTIN";
CATEGORY = "FINAL";
TABLE_NAME = "LSWEATHERDATARPT";
SAVE_HNDL = INTDSAVEEX(WEATHER_STATION, CATEGORY, TABLE_NAME,
TEMP_HNDL);
```

Example: Weather Data table - INTDSAVEEXP

Save the interval data in TEMP_HNDL to a new Weather Station (WS_02 in Dallas, TX) with a category of "FINAL" to the Weather Station (LSWEATHERSTATION), and Weather Data (LSWEATHERDATA) tables.

```
WEATHER_STATION.STATIONCODE = "WS_02";
WEATHER_STATION.JURISCODE = "DALLAS";
CATEGORY = "FINAL";
TABLE_NAME = "LSWEATHERDATA";
SAVE_HNDL = INTDSAVEEXP(WEATHER_STATION, CATEGORY, TABLE_NAME,
TEMP_HNDL);
```

Note: Even if a category is not used, the <category> parameter is required when using the INTDSAVEEX function. In this case, specify an empty string ("") for the <category>.

Bulk Saving Data to Enhanced/Generic Interval Data Tables

You can also save multiple interval data handles stored within an array identifier to enhanced/ generic interval data tables in a single function using either INTDSAVEEX or INTDSAVEEXP.

Bulk Saves using INTDSAVEEX

When performing bulk saves using the INTDSAVEEX function, enter the parameters as follows:

- Use an empty string ("") for the <parent> parameter. The parent identity of each handle is derived from the array identifier that contains the handles to be saved.
- Use an empty string ("") for the <category> parameter. The category of each handle is derived from the array identifier that contains the handles to be saved.
- Supply the table name as usual.
- Use an array identifier (#ARR[]) as the <hndl_array> parameter. This is an array identifier that contains the interval data handles to be saved.

Example: Weather Data table - INTDSAVEEX - Bulk Saves

Load data for weather stations WS_1 through WS_30 in Texas (TEXAS) that have a category of "INITIAL" for the month of June 2007 and save them to existing weather stations WST_1 through WST_30 with a category of "FINAL."

```
TABLE_NAME = "LSWEATHERDATA";
START = DATE ('06/01/2007 00:00:00');
STOP = DATE ('06/30/2007 23:59:59');
FOR EACH REC IN LIST "GET_WS_DATA"
   X = X + 1;
   WS = REC.STATIONCODE;
   JURIS = REC.JURISCODE;
   WEATHER_STATION = WS + "," + JURIS;
   #ARR[X] = INTDLOADEXDATES (WEATHER_STATION, "INITIAL", TABLE_NAME,
START, STOP);
   WST = "WST_" + X;
   PARENT_KEY = WST + "," + JURIS
   CATEGORY = "FINAL";
   SET_PK = INTDSETATTRIBUTE (#ARR[X], "PARENTKEY", PARENT_KEY);
   SET_CAT = INTDSETATTRIBUTE (#ARR[X], "CATEGORY", CATEGORY);
END FOR;
SAVE_HNDL = INTDSAVEEX("", "", TABLE_NAME, #ARR[]);
```

Bulk Saves using INTDSAVEEXP

When performing bulk saves using the INTDSAVEEX function, enter the parameters as follows:

- Use an array identifier (#PK[]) for the <parent> parameter. This array must contain <stem>.<tail> identifiers for the parent records.
- Use an array identifier (#CAT[]) or single category ("FINAL") for the <category> parameter.
- Supply the table name as usual.
- Use an array identifier (#ARR[]) as the <hndl_array> parameter. This is an array identifier that contains the interval data handles to be saved.

Example: Weather Data table - INTDSAVEEXP - Bulk Saves

Load data for weather stations WS_1 through WS_30 in Texas (TEXAS) that have a category of "INITIAL" for the month of June 2007 and save them to new weather stations WSN_1 through WSN_30 with a category of "FINAL."

```
TABLE_NAME = "LSWEATHERDATA";
START = DATE ('06/01/2007 00:00:00');
STOP = DATE ('06/30/2007 23:59:59');
FOR EACH REC IN LIST "GET_WS_DATA"
```

```
   X = X + 1;
   WS = REC.STATIONCODE;
   JURIS = REC.JURISCODE;
   WEATHER_STATION = WS + "," + JURIS;
   #ARR[X] = INTDLOADEXDATES (WEATHER_STATION, "INITIAL", TABLE_NAME,
START, STOP);
   WSN = "WSN_" + X;
   PARENT_KEY = WSN + "," + JURIS
   #PAR[X] = PARENT_KEY;
   CATEGORY = "FINAL";
   SET_PK = INTDSETATTRIBUTE (#ARR[X], "PARENTKEY", PARENT_KEY);
END FOR;
SAVE_HNDL = INTDSAVEEXP(#PAR[], CATEGORY, TABLE_NAME, #ARR[]);
```

Deleting Data from Enhanced/Generic Interval Data Tables

Deleting interval data from enhanced/generic intervals is performed using the following function:

- **INTDDELETEEX** - Deletes an interval data cut. See **INTDDELETEEX Function** on page 9-86 in the *Oracle Utilities Rules Language Reference Guide* for more information.

Example: Weather Data table - INTDDELETEEX

Delete the interval data for Weather Station WS_01 in Austin, TX (AUSTIN) with a start date of 01/01/2007 with a with a category of "FINAL" from the Weather Data (LSWEATHERDATA) table.

```
STARTDT = DATE('01/01/2007 00:00:00');
WEATHER_STATION = "WS_01,AUSTIN";
CATEGORY = "FINAL";
TABLE_NAME = "LSWEATHERDATA";
SAVE_HNDL = INTDDELETEEX(WEATHER_STATION, CATEGORY, TABLE_NAME,
STARTDT);
```

Deriving Billing Determinants and Values from Interval Data

This section describes how to derive billing determinants from loaded interval data, including:

- **Overview**
- **Bill Determinants**
- **Other Values**

Overview

After interval data has been loaded, and operated on as appropriate (such as creating masks or performing block/scalar operations), you can derive billing determinants and other values from the data for use in other Rules Language calculations.

The format for deriving values from an interval data handle is:

```
<value> = HNDL.ATTRIBUTE;
```

where

- **<value>**: an identifier you assign to the value.
- **HNDL**: the interval data handle that you assigned in the INDTLOAD statement (which automatically refers to the entire handle)
- **ATTRIBUTE**: the name of a particular attribute of the handle. See **Interval Data Reference Values and Attributes** on page 7-3 for a list of values that can be derived from interval data handles.

For example, one of the available values is TOTAL, which is the total of all the interval values in the handle. If you used the handle HNDL, you could retrieve the total value in using an Assignment Statement as follows:

```
HNDL_TOTAL = HNDL.TOTAL;
```

Bill Determinants

You derive bill determinants from interval data so that you can then use those determinants in other Rules Language calculations. For example, if you were billing based on interval data, you need the bill determinants in order to calculate charges.

There are three interval data attributes used in deriving bill determinants:

- **TOTAL**: The sum of all interval values in the interval data handle.
- **ENERGY**: Total energy represented by the handle, computed properly for its UOM according to the TOTAL flag in the UOM Table. The UOM for the interval values must be either KW or KWH. If not, result is 0.
- **KW_MAXIMUM**: The maximum KW value in the handle. If the UOM is KWH, the actual maximum is multiplied by the IPH (intervals per hour) to get this value.

Of the three, TOTAL is most often used, as it represents the total of all the interval values in the handle. KW_MAXIMUM can be used when calculating charges based on demand.

Example

If you loaded a cut that measures kWh, to derive the total KWH from the handle, you could use the following statements:

```
HNDL = INTDLOAD (KWH);
HNDL_KWH = HNDL.TOTAL;
```

To derive the energy from the same handle, you could use the following statement:

```
HNDL_ENERGY = HNDL.ENERGY;
```

To derive the maximum kW from the same handle, you could use the following statement:

```
HNDL_KW_MAX = HNDL.KW_MAXIMUM;
```

Other Values

You can derive other values in the same manner as bill determinants. Some examples include:

Start Time

```
HNDL_STARTTIME = HNDL.STARTTIME;
```

Stop Time

```
HNDL_STOPTIME = HNDL.STOPTIME;
```

Average

```
HNDL_AVG = HNDL.AVERAGE;
```

Unit-of-Measure

```
HNDL_UOM = HNDL.UOM;
```

Intervals Per Hour

```
HNDL_IPH = HNDL.IPH;
```

Load Factor

```
HNDL_LF = HNDL.LF;
```

Using Other Values

While not used directly in calculating charges, these others values can be used in determining under which circumstances certain calculations should be performed. For example, suppose a particular charge in a tariff were based on whether the average usage was above or below a particular threshold (stored in the database as a Factor called "LOW_AVERAGE"), you would derive the Average value and compare it the factor in order to determine which rate to use. The following shows one way of doing this.

```
HNDL = INTDLOAD (KWH);
HNDL_AVG = HNDL.AVERAGE;
IF AVERAGE <= FACTOR["LOW_AVERAGE"].VALUE
   THEN
   ALL KWH CHARGE FACTOR["KWH_CHG_1"].VALUE;
   ELSE
   ALL KWH CHARGE FACTOR["KWH_CHG_2"].VALUE;
END IF;
```

Examples of Working with Interval Data

This section includes some examples of working with interval data using the Rules Language, including:

- **Loading Interval Data**
- **Time-of-Use Periods**
- **Override Masks**
- **Time-of-Use and Overrides**
- **Calculating Coincident and Non-Coincident Peaks**

Loading Interval Data

The following examples demonstrate a number of the different **INTDLOAD Functions**.

Bill Determinant

```
HNDL = INTDLOAD (KWH);
```

Bill Determinant and Date Range

```
HNDL = INTDLOADDATES(KWH, BILL_START, BILL_STOP);
```

Historical

```
HNDL = INTDLOADHIST(KWH, 1, 11);
```

List of Recorder,Channels

```
HNDL = INTDLOADLIST("ACCOUNT_CH");
```

UOM

```
HNDL = INTDLOADUOM("01");
```

Time-of-Use Periods

You use the **INTDCREATETOUPERIOD Function** to create time-of-use (TOU) periods, such as on peak or off peak periods of usage. The following example creates On Peak and Off Peak time-of-use periods based on a previously loaded interval data cut.

```
//Load KWH Handle
HNDL = INTDLOAD (KWH);

//Create On Peak KWH Handle
ON_PEAK_HNDL = INTDCREATETOUPERIOD(HNDL, "VALUE", "TOU1", "ON_PEAK",
"2000 HOLIDAYS");

//Create Off Peak KWH Handle
OFF_PEAK_HNDL = INTDCREATETOUPERIOD(HNDL, "VALUE", "TOU1", "OFF_PEAK",
"2000 HOLIDAYS");

//Derive On Peak KWH
ON_PEAK_KWH = ON_PEAK_HNDL.TOTAL;

//Derive Off Peak KWH
OFF_PEAK_KWH = OFF_PEAK_HNDL.TOTAL;
```

In the INTDCREATETOUPERIOD functions in this example:

- **HNDL**: is the interval data reference loaded in the **INTDLOAD Function**.
- **VALUE**: is an operation that sets the values that fall within the TOU period to the same value as in HNDL, and sets all other values to zero.

- **TOU1**: is the time-of-use schedule. See **Time-of-Use Schedules** on page 7-8 in the *Data Manager User's Guide* for more information about creating time-of-use periods.
- **ON_PEAK/OFF_PEAK**: is the time-of-use period.
- **2000 HOLIDAYS**: is a holiday list used by the time-of-use schedule. See **Holidays** on page 7-6 in the *Data Manager User's Guide* for more information about creating holiday lists.

Override Masks

Override masks are a type of interval data mask based on an override. You use the **INTDCREATEOVERRIDEMASK Function** to an override mask. The following example creates override masks for curtailment/non-curtailment periods based on a previously loaded interval data cut.

```
//Load KWH Handle
HNDL = INTDLOAD (KWH);

//Create Curtailment KWH Handle
CURTAILMENT_HNDL = INTDCREATEOVERRIDEMASK(HNDL, "CURTAILMENT",
"VALUE");

//Create Non-Curtailment KWH Handle
NON_CURTAILMENT_HNDL = INTDCREATEOVERRIDEMASK(HNDL, "CURTAILMENT",
"REVERSE_VALUE");

//Derive Curtailment KWH
CURTAILMENT_KWH = CURTAILMENT_HNDL.TOTAL;

//Derive Non-Curtailment KWH
NON_CURTAILMENT_KWH = NON_CURTAILMENT_HNDL.TOTAL;
```

In the INTDCREATEOVERRIDEMASK functions in this example:

- **HNDL**: is the interval data reference loaded in the **INTDLOAD Function**.
- **CURTAILMENT**: is an override stored in the Oracle Utilities Data Repository.
- **VALUE**: is an operation that sets the values that fall within the curtailment period to the same value as in HNDL, and sets all other values to zero.
- **REVERSE_VALUE**: is an operation that sets the values that fall outside the curtailment period to the same value as in HNDL, and sets all other values to zero.

Time-of-Use and Overrides

You can also combine time-of-use periods and overrides, if for example an override were in effect during both on peak and off peak times. The following example creates override masks for curtailment/non-curtailment periods for both on peak and off peak periods.

```
//Load KWH Handle
HNDL = INTDLOAD (KWH);

//Create On Peak KWH Handle
ON_PEAK_HNDL = INTDCREATETOUPERIOD(HNDL, "VALUE", "TOU1", "ON_PEAK",
"2000 HOLIDAYS");

//Create On Peak Curtailment KWH Handle
ON_PEAK_CURTAILMENT_HNDL = INTDCREATEOVERRIDEMASK(ON_PEAK_HNDL,
"CURTAILMENT", "VALUE");

//Create On Peak Non-Curtailment KWH Handle
ON_PEAK_NON_CURTAILMENT_HNDL = INTDCREATEOVERRIDEMASK(ON_PEAK_HNDL,
"CURTAILMENT", "REVERSE_VALUE");

//Create Off Peak KWH Handle
```

```
OFF_PEAK_HNDL = INTDCREATETOUPERIOD(HNDL, "VALUE", "TOU1", "OFF_PEAK",
"2000 HOLIDAYS");

//Create Off Peak Curtailment KWH Handle
OFF_PEAK_CURTAILMENT_HNDL = INTDCREATEOVERRIDEMASK(OFF_PEAK_HNDL,
"CURTAILMENT", "VALUE");

//Create Off Peak Non-Curtailment KWH Handle
OFF_PEAK_NON_CURTAILMENT_HNDL = INTDCREATEOVERRIDEMASK(OFF_PEAK_HNDL ,
"CURTAILMENT", "REVERSE_VALUE");

//Derive On Peak Curtailment KWH
ON_PEAK_CURTAILMENT_KWH = ON_PEAK_CURTAILMENT_HNDL.TOTAL;

//Derive On Peak Non-Curtailment KWH
ON_PEAK_NON_CURTAILMENT_KWH = ON_PEAK_NON_CURTAILMENT_HNDL.TOTAL;

//Derive Off Peak Curtailment KWH
OFF_PEAK_CURTAILMENT_KWH = OFF_PEAK_CURTAILMENT_HNDL.TOTAL;

//Derive Off Peak Non-Curtailment KWH
OFF_PEAK_NON_CURTAILMENT_KWH = OFF_PEAK_NON_CURTAILMENT_HNDL.TOTAL;
```

In the INTDCREATETOUPERIOD functions in this example:

- **HNDL**: is the interval data reference loaded in the **INTDLOAD Function**.
- **VALUE**: is an operation that sets the values that fall within the TOU period to the same value as in HNDL, and sets all other values to zero.
- **TOU1**: is the time-of-use schedule. See **Time-of-Use Schedules** on page 7-8 in the *Data Manager User's Guide* for more information about creating time-of-use periods.
- **ON_PEAK/OFF_PEAK**: is the time-of-use period.
- **2000 HOLIDAYS**: is a holiday list used by the time-of-use schedule. See **Holidays** on page 7-6 in the *Data Manager User's Guide* for more information about creating holiday lists.

In the INTDCREATEOVERRIDEMASK functions in this example:

- **HNDL**: is the interval data reference loaded in the **INTDLOAD Function**.
- **CURTAILMENT**: is an override stored in the Oracle Utilities Data Repository.
- **VALUE**: is an operation that sets the values that fall within the curtailment period to the same value as in HNDL, and sets all other values to zero.
- **REVERSE_VALUE**: is an operation that sets the values that fall outside the curtailment period to the same value as in HNDL, and sets all other values to zero.

Calculating Coincident and Non-Coincident Peaks

A coincident peak is the peak in an aggregated handle (a handle that comprises multiple handles added together). By identifying the time of a coincident peak, you can also determine the value at that time in the individual cuts that comprise the aggregated cut.

For example, suppose a customer has three accounts, each with interval data cuts for the same bill period. If you load those cuts and add them together, you can use the time of the peak value of the aggregated cut to determine the coincident peak for each account (the usage at the time of the coincident peak).

The non-coincident peak for a cut is the maximum value of the cut.

Rate tariffs can include different charges based on coincident and non-coincident peaks. For instance, the demand charge for an account might be based on a ratio between the aggregated peak and the coincident peak for that account.

The following is a simple example of calculating a coincident time from an aggregated cut and then determining the coincident and non-coincident peaks for each of the individual cuts that comprise the aggregated cut.

```
//Load and aggregate cuts
ACCT1_HNDL = INTDLOADDATES('A20991,1', BILL_START, BILL_STOP);
ACCT2_HNDL = INTDLOADDATES('A20992,1', BILL_START, BILL_STOP);
ACCT3_HNDL = INTDLOADDATES('A20993,1', BILL_START, BILL_STOP);

AGGREGATE_HNDL = ACCT1_HNDL + ACCT2_HNDL + ACCT3_HNDL;

/* Calculate Coincident Peak Time */
COINCIDENT_TIME = INTDVALUE(AGGREGATE_HNDL, "MAXDATE");

/* Clear Aggregate Handle */
DUMMY = INTDRELEASE(AGGREGATE_HNDL);

//Find Coincident and Non-Coincident Peaks for Account 1
ACCT1_NC_PEAK = INTDVALUE(ACCT1_HNDL, "MAX");
ACCT1_COINCIDENT_PEAK_INDEX = INTDVALUE(ACCT1_HNDL, "DATE_INDEX",
COINCIDENT_TIME);
ACCT1_COIN_PEAK = INTDVALUE(ACCT1_HNDL, "INDEX",
ACCT1_COINCIDENT_PEAK_INDEX);

//Find Coincident and Non-Coincident Peaks for Account 2
ACCT2_NC_PEAK = INTDVALUE(ACCT2_HNDL, "MAX");
ACCT2_COINCIDENT_PEAK_INDEX = INTDVALUE(ACCT2_HNDL, "DATE_INDEX",
COINCIDENT_TIME);
ACCT2_COIN_PEAK = INTDVALUE(ACCT2_HNDL, "INDEX",
ACCT2_COINCIDENT_PEAK_INDEX);

//Find Coincident and Non-Coincident Peaks for Account 3
ACCT3_NC_PEAK = INTDVALUE(ACCT3_HNDL, "MAX");
ACCT3_COINCIDENT_PEAK_INDEX = INTDVALUE(ACCT3_HNDL, "DATE_INDEX",
COINCIDENT_TIME);
ACCT3_COIN_PEAK = INTDVALUE(ACCT3_HNDL, "INDEX",
ACCT3_COINCIDENT_PEAK_INDEX);
```

In the INTDVALUE functions in this example:

- **AGGREGATE_HNDL**: is an aggregated interval data handle comprising the three previously loaded handles (using the **INTDLOADDATES Function**).
- **MAXDATE**: is an operation that retrieves the date and time of the maximum value of the aggregated handle.
- **ACCTx_HNDL**: is the interval data reference loaded in the corresponding **INTDLOADDATES Function**.
- **MAX**: is an operation that retrieves the maximum value of the handle.
- **DATE_INDEX**: is an operation that retrieves the index that corresponds to COINCIDENT_TIME.
- **COINCIDENT_TIME**: is an identifier equal to the date and time of the maximum value of the aggregated handle (derived from MAXDATE).
- **ACCTx_COINCIDENT_PEAK_INDEX**: is an identifier equal to the index of the coincident peak in the corresponding interval data handle (derived from DATE_INDEX).
- **INDEX**: is an operation that retrieves the index that corresponds to ACCTx_COINCIDENT_PEAK_INDEX.

The above example uses hard-coded values for the individual handles and identifiers. The same calculations could be performed more efficiently using a list of accounts or recorder,channels and

the **For Each x In List Statement**, and using **Indirect Identifiers** to determine the coincident and non-coincident peaks An example of this approach is shown below.

Example using FOR EACH statements and Indirect Identifiers

```
//Load & aggregate usage data, and find & store time of coincident
peak.
//
FOR EACH ACCOUNT IN LIST GET_COINCIDENT_ACCOUNTS
   RECORDER_ID = ACCOUNT + ",1";
   AGGREGATED_LOAD_HNDL = AGGREGATED_LOAD_HNDL +
INTDLOADDATES(RECORDER_ID, START_DATE, STOP_DATE);
END FOR;
COINCIDENT_TIME = INTDVALUE(AGGREGATED_LOAD_HNDL, MAXDATE);
CLEAR AGGREGATED_LOAD_HNDL;

//For each account, find coincident & non-coincident peaks
//
FOR EACH ACCOUNT IN LIST GET_COINCIDENT_ACCOUNTS
   RECORDER_ID = ACCOUNT + ",1";
   ACCOUNT_LOAD_HNDL = INTDLOADDATES(RECORDER_ID, START_DATE,
STOP_DATE);
   X = ACCOUNT + "_NC_PEAK";
   @X = INTDVALUE(ACCOUNT_LOAD_HNDL , MAX);
   COINCIDENT_PEAK_INDEX = INTDVALUE(ACCOUNT_LOAD_HNDL, "DATE_INDEX",
COINCIDENT_TIME);
   X = ACCOUNT + "_COIN_PEAK";
   @X = INTDVALUE(ACCOUNT_LOAD_HNDL, "INDEX", COINCIDENT_PEAK_INDEX);
END FOR;
```

Chapter 8

Working with COM Components

This chapter describes how you work with COM components using the Oracle Utilities Rules Language, including:

- **Overview**
- **Working with COM Objects**
  - **Creating COM Objects**
  - **COM Expressions**
  - **COM Object Functions**
  - **COM Error Handling**
- **VARIANT Data Type**
- **Examples**

Overview

The Oracle Utilities Rules Language can be used to create COM objects and invoke methods available from COM components. This allows users to create Rules Language logic to invoke COM methods (including those available from COM interfaces) as part of their business processing.

Working with COM Objects

Working with COM objects in the Oracle Utilities Rules Language involves the following:

- **Creating COM Objects**
- **COM Expressions**
- **COM Object Functions**
- **COM Error Handling**

Creating COM Objects

The first step in using COM objects in the Oracle Utilities Rules Language is to create an instance of the COM object you wish to work with. This is done via the **CREATEOBJECT Function**, which creates a COM object based on the object's program ID (ProgID).

The format of this function is as follows:

```
<identifier> = CREATEOBJECT(<ProgID>);
```

where:

- <ProgID> is a string that contains the ProgID of the COM object to be created.

Example: Create a DOMDocument COM object.

```
//Create a DOMDocument object
OBJECT = CREATEOBJECT ("MSXML.DOMDocument");
```

For more information about this function, see the **CREATEOBJECT Function** on page 13-105 in the *Oracle Utilities Rules Language Reference Guide*.

COM Expressions

Once an instance of a COM object has been created by the CREATEOBJECT function, users can get and set an object's properties, or to call an object's method(s) using a specific Rules Language syntax.

Get Property

To obtain a property from a COM object created in the Rules Language, use the following syntax:

```
<identifier> = [object]->[property];
```

or

```
<identifier> = [object]->[property](<param1>|<expression>,
<param2>|<expression>);
```

where:

- [object] is the COM object created in the Rules Language.
- [property] is the property to be obtained from the object.
- <param> is a parameter required to obtain the property from the object. If multiple parameters are required, they must be separated by a comma (",").
- <expression> is a Rules Language expression that evaluates to a parameter required to obtain the property from the object.

Set Property

To set a property in a COM object created in the Rules Language, use the following syntax:

```
[object]->[property] = <identifier | expression>;
```

where:

- [object] is the COM object created in the Rules Language.
- [property] is the property to be set in the object.
- <identifier> is an identifier that contains the value to which the property in the object is to be set.
- <expression> is a Rules Language expression that evaluates to the value to which the property in the object is to be set.

Invoking Methods

To invoke (or call) a method in a COM object created in the Rules Language, use the following syntax:

```
[object]->[method]();
```

or

```
<identifier> = [object]->[method]();
```

or

```
<identifier> = [object]->[method](<param1>|<expression>,
<param2>|<expression>);
```

or

```
[object]->[method](<param1>|<expression>, <param2>|<expression>);
```

where:

- [object] is the COM object created in the Rules Language.
- [method] is the property to be invoked in the object. If the method does not require parameters, empty parentheses ("()")are still required.
- <param> is a parameter required to invoke the method. If multiple parameters are required, they must be separated by a comma (",").
- <expression> is a Rules Language expression that evaluates to a parameter required to invoke the method.

COM Object Functions

The Rules Language also provides other functions for working with COM objects, and include the following.

GETADOCONNECTION Function

The Rules Language uses an internal ADO database connection. This same database connection may be required by some third-party COM components. The **GETADOCONNECTION Function** can provide access to this connection.

The format of this function is as follows:

```
<identifier> = GETADOCONNECTION();
```

Example: obtain an ADO database connection to execute a query

```
//Invoke the "ExecuteQuery" method of an LSDB DataSource COM object.
OBJECT = CREATEOBJECT ("LSDB.DataSource");
CON = GETADOCONNECTION ();
RES = OBJECT->ExecuteQuery(CON, XML_QUERY);
```

For more information about this function, see the **GETADOCONNECTION Function** on page 13-6 in the *Oracle Utilities Rules Language Reference Guide*.

FOR EACH X IN IENUM Statement

The **For Each x In COM IENUM Statement** provides a method of executing statements for a set of variants.

The following sample statements set the values of the "ACCOUNTID" nodes in an XML document to the value of the "TEXT" property.

```
OBJECT = CREATEOBJECT ("MSXML.DOMDocument");
XMLNODES = OBJECT->SELECTNODES ("//ACCOUNTID");
    FOR EACH X IN IENUM XMLNODES
        ACCOUNTID = X->TEXT;
    END FOR;
```

For more information about this statement, see the **For Each x In COM IENUM Statement** on page 3-20 in the *Oracle Utilities Rules Language Reference Guide*.

COM Error Handling

When using COM expressions and functions, the following reserved identifiers are created in the Rules Language to aid in error handling. See **Reserved Identifiers** on page 4-13 for more information about reserved identifiers.

COM_ERROR_STOP

Specifies error handling behavior in the event of an error in a COM method invoked by the Rules Language. Available settings include:

- 0 - Use the STOP_ON_COM_ERROR configuration parameter. See **LODESTAR.CFG** on page 2-2 in the *Oracle Utilities Energy Information Platform Configuration Guide* for more information about this parameter.
- 1 - Ignore the error and set value of LASTCOMERROR identifier.
- 2 - Stop Rules Language processing.

LASTCOMERROR

The error code for the most recent COM error. This identifier is automatically populated when COM_ERROR_STOP is set to 1, or when the STOP_ON_COM_ERROR configuration parameter is not present in the LODESTAR.CFG file. This identifier can be used with the IF THEN statement to perform specific processing in the event of a COM error.

LASTCOMERRORTEXT

The error description for the most recent COM error. This identifier is automatically populated when the LASTCOMERROR identifier is populated.

VARIANT Data Type

This section provides technical details concerning the VARIANT data type and how it is used by COM objects in the Rules Language.

The VARIANT data type encapsulates all simple (BSTR, INT, BOOL...) types and allows a COM object's encapsulating (including IUnknown and IDispatch interfaces). The Rules Language has a similar unionized type that encapsulates the types used by Rules Language. By encapsulating the VARIANT type into the Rules Language type, the VARIANT type can be used in Rules Language.

Conversion from a Rules Language type to a VARIANT and vise versa is handled automatically by the Rules Language.

The Trial Calculation module shows identifiers with the VARIANT type.

For the COM interfaces (IUnknown, IDispatch) the Edit ID dialog displays a type of "VARIANT", with the value set to the interface name, address and VT type.

For the other identifiers of type "VARIANT", the Edit ID dialog displays only the VT type.

Mathematical expressions are not supported for the VARIANT type identifiers. Only the following VT types can be used in logical expressions: VT_IDISPATCH, VT_IUNKNOWN, VT_ERROR. These VT types can be compared with a NUMBER.

Example:

```
X = DOCELEMENT->SELECTSINGLENODE("//TEST");
If(X = 0) THEN
//print error
End if;
```

Rules Language and VARIANT Types

The following tables indicates how VARIANT types correspond to Rules Language data types.

VARIANT to Rules Language

Variant type	RSCD type	RSCL type
VT_EMPTY VT_NULL	RSCD_NOTYPE	RSCL_NONE
VT_I1 VT_I2 VT_I4 VT_BOOL VT_UI1 VT_UI2 VT_UI4 VT_I8 VT_UI8 VT_INT VT_UINT	RSCD_INTEGER	RSCL_INTEGER
VT_R4 VT_R8 VT_CY VT_DECIMAL	RSCD_FLOAT	RSCL_FLOAT
VT_BSTR*	RSCD_STRING	RSCL_LITERAL
VT_DATE	RSCD_DATE	RSCL_DATE

VT_DISPATCH VT_ERROR VT_VARIANT VT_UNKNOWN VT_RECORD VT_VOID VT_HRESULT VT_PTR VT_SAFEARRAY VT_CARRAY VT_USERDEFINED VT_BLOB VT_STREAM VT_STORAGE VT_STREAMED_OBJECT VT_STORED_OBJECT VT_VERSIONED_STREAM VT_BLOB_OBJECT VT_CF VT_CLSID VT_VECTOR VT_ARRAY VT_BYREF VT_BSTR_BLOB	RSCD_VARIANT	RSCL_VARIANT

* if length of string will be more than 260 symbols this string will be handled as VARIANT type.

VARIANT to Rules Language

RSCD type	RSCL type	Variant type
RSCD_NOTYPE	RSCL_NONE	VT_EMPTY
RSCD_INTEGER	RSCL_INTEGER	VT_I4
RSCD_FLOAT	RSCL_FLOAT	VT_R4
RSCD_STRING	RSCL_LITERAL	VT_BSTR
RSCD_DATE	RSCL_DATE	VT_DATE

RSCD_VARIANT	RSCL_VARIANT	VT_DISPATCH VT_ERROR VT_VARIANT VT_UNKNOWN VT_RECORD VT_VOID VT_HRESULT VT_PTR VT_SAFEARRAY VT_CARRAY VT_USERDEFINED VT_BLOB VT_STREAM VT_STORAGE VT_STORED_OBJECT VT_VERSIONED_STREAM VT_BLOB_OBJECT VT_CF VT_CLSID VT_VECTOR VT_ARRAY VT_BYREF VT_BSTR_BLOB

Examples

This section provides additional examples of using COM objects in the Oracle Utilities Rules Language.

Using Scripting.FileSystemObject

The following example creates a text file containing the line "Hello" using the File System Object.

```
FSO = CREATEOBJECT("Scripting.FileSystemObject");
TF = FSO->CreateTextFile("results.dat", 1);
TF->WriteLine("Hello");
```

Using LSDB.DataSource

The following example executes a query using the ExecuteQuery method of the LSDB.DataSource COM object.

```
//Create new instance of the object
OBJLSDB = CREATEOBJECT("LSDB.DataSource");
//Call method with parameters
DATASOURCE = CreateObject("MSXML2.DOMDocument.4.0");
DATASOURCE->load("DataSource.xml");
X = DATASOURCE->XML;
//Create new instance of the object
EXECQUERY = CREATEOBJECT("MSXML2.DOMDocument.4.0");
EXECQUERY->load("Query.xml");
Y = EXECQUERY->XML;
QUERYOUT = OBJLSDB->ExecuteQuery(X,Y);
DOMREQUEST = CREATEOBJECT("MSXML2.DOMDocument.4.0");
DOMREQUEST->loadXML(QUERYOUT);
SAVE COMMIT;
```

Appendix A

Setting Up Rate Form Records and Rate Codes

This appendix explains how to create rate form records and rate codes. These records must exist in the Oracle Utilities Data Repository before you can write rate forms using the Oracle Utilities Rules Language. The Oracle Utilities Rules Language allows you to create and save three versions of a rate form:

- **Current**—the version now in effect for your customers.
- **Historical**—the versions previously in effect.
- **Trial**—for trial analysis only.

Before you can create any version, you must first set up a Rate Form record. The Rate Form record is the parent record that makes it possible to keep track of all of its versions (from the database perspective, its "child" records.)

You must also set up descriptive records for rate codes that you wish to incorporate in rate schedules.

This chapter explains how to set up the three types of records in the Oracle Utilities Data Repository: rate forms, rate form versions, and rate codes. Writing the rate form scripts (encoding the English language description of a rate form into a computer-usable format) is described in the *Rules Language User's Guide*.

Adding a Rate Form Record

Information is entered via the Browser.

How to create a rate form record:

1. Select **File-›Browse-›Customer Database**. The Browser window appears.
2. Click on the table icon for **Rate Forms**. A list of the existing rate form records appears on the right side of the Browser window.
3. Select **Records-›Insert New,** or click the *right* mouse button and select **Insert New**. A data entry form appears on the right side of the Browser.
4. Select an **Operating Company** for the rate form. Position the cursor in the field and click the mouse button. Select a code from the list that appears—be sure to click in the first column. When the desired selection is highlighted, press the ENTER key or click **OK**.
5. Select a **Jurisdiction** for the rate form. Select a code using the same method you used for operating company.
6. Enter a **Code** for the rate form. This is a unique identifier (up to 64 characters) for the rate form.
7. Select the **Type** of rate form
   - **Contract**: A contract describes bill calculations that apply to a single customer. Contracts are the only rate form type that allow use of a "Null" value for the Operating Company and Jurisdiction codes.
   - **Rider:** "Rider" is used here to refer to any sub-form. A sub-form is a set of statements that you want to make available for use in several rate schedules. This could include tariff riders, company-wide rules and regulations, or other common calculations such as a set of statements that creates time-of-use billing determinants from interval data.
   - **Rate Schedule:** A rate schedule describes the bill calculations for a class of customers. In another way of looking at it, *a rate schedule is the rules language form of a rate tariff.* When used as input to a Oracle Utilities program, rate schedules are used to compute revenue from billing determinants.

     See the *Oracle Utilities Rules Language User's Guide* for additional information about the three types of rate forms.
8. Enter a **Name** or other description for the rate form. This name will help users recognize the code in displays and reports. It can be up to 64 characters long.
9. Enter an optional **Note**, such as a description or comments, up to 254 characters.
10. Set the **Billing Mode Flag** for the rate form *(used with Oracle Utilities Billing Component).*
11. Set the **Print Detail** for the rate form. This specifies the level of information to be included in reports and transaction records for the account *(used with Oracle Utilities Billing Component).*
12. Set the **Full Day Bill** flag for the rate form. This specifies whether the end of the bill period, as recorded in the Bill History records for the accounts on this rate, is the last full 24-hour period (ending at midnight) prior to the meter read date and time, or whether it is the actual meter read date and time. If you specify Yes, the midnight of the last full day is stored; if you specify No, the actual date and time are stored. Keep in mind that if you specify Yes, it is the functions supplied in the rate schedule that determine which stop date and time is applied when calculating energy and/or demand—the cut's actual stop date and time, or that stored in the Bill History record. Also, any setting you supply for Full Day for the individual account will override this setting *(used with Oracle Utilities Billing Component).*
13. Specify whether or not the rate form will be editable by setting the **Editable** column as appropriate (Yes or No). Non-editable rate forms are considered "locked" and cannot be

edited or changed (and the same applies to Rate Form Version records and Rate Form Version Text records for locked rate forms).

14. When you have completed the fields for this record, select **Edit->Add,** or click the *right* mouse button and select **Add**, to save it to the database.

If you are creating a rate schedule record, you may wish to now add its "child" rate code records using the following instructions.

Adding Rate Codes

Accounts are related to rate schedules through rate codes. For that reason, each rate schedule can include the definition of several rate codes, but it must include *at least one*.

A rate code represents a subdivision of the class of accounts billed according to a rate schedule. You could have a residential rate schedule, R, with two kinds of customers, each assigned a special rate code: the first group of customers on rate code RH has hot water heating; the second group on rate code RNH do not. RH customers will have an additional charge just for hot water heating. To distinguish between the two groups, the Rules Language lets you access those rate codes within a rate schedule script.

However, before you can do that, you must define the codes in the Oracle Utilities Data Repository.

A Rate Code record must have a parent rate schedule Rate Form record. A rate schedule can have an unlimited number of "child" rate codes associated with it, but it must have at least one.

How to add rate codes:

To create a Rate Code record, click the small plus sign next to the icon for its parent Rate Form record in the Tree pane. Near the bottom of the expanded tree for the Rate Form record, you'll see a table icon for Rate Code. Click on it, and select **Records->Insert New** (or click the right mouse button and select **Add**).

Most of the fields are automatically filled in using information from the parent record. There are two remaining fields:

Code: Enter a unique identifier for the rate code. It can be up to 64 characters long.

Note: Optional description or comments, up to 254 characters.

When you have completed the fields for this record, select **Edit->Add**, or click the *right* mouse button and select **Add**, to save it to the database.

Creating a New Rate Form Version

You can create three different versions of a rate form: Current, Trial, or Historical.

Each rate form version has an associated effective start date. Rate forms that have been superseded (have earlier start-dates than another) are considered "historical." The rate form version with the latest start date is the "current" version.

In addition, you may want to test changes to a rate form without affecting the current or historical versions. You can create a "trial" version, which has no date associated with it.

Many of the Oracle Utilities analyses have a simple option—select the rate forms based on the bill month, or based on your selection. If you choose the bill month option, the analysis automatically selects the correct current or historical version based on the bill month. Otherwise, you can select a specific rate form, including any current, historical, or trial version.

How to create a new rate form version record via the Browser:

To use the Browser method, expand the tree for the parent Rate Form record. Click on the table icon for Rate Form Versions, and select **Records->Insert New,** or click the *right* mouse button and select **Insert New**. Complete the fields as desired (see following section for description of the fields). Finally, select **Edit->Add,** or click the *right* mouse button and select **Add**.

How to create a new rate form version record:

1. Select **File->New->Rate Schedule Version**.

   The **New Rate Form Version** dialog box appears.

2. Select the **Operating Company** and **Jurisdiction** to which your rate form belongs. The list of currently available Rate Form records appears in the list box.

3. Highlight the desired Rate Form in the list box.

4. Your input in the remaining two fields depends upon which type of version you wish to create. For a trial version, enter a version number (but no start date). For a current or historical version, enter its start date (but no version number).

   The difference between historical and current versions is that the current version has the most recent start date.

   You can create virtually any number of trial or historical versions, but there can be only one current version. If you are creating a trial rate schedule and plan to include a trial contract in it, specify a version number between 9000 - 9999, inclusive, for both rate forms. Otherwise, the system will automatically include the current version of the contract.

5. Click **OK**. The record you just created is saved to the database, and a Rules Language Editor window opens. You can now create a rate form script using any of the statements described in the *Oracle Utilities Rules Language User's Guide*.

Importing and Exporting Rate Forms in Batch Mode

In addition to setting up and creating rate forms, you can move a rate form from an older database to a newer one, while verifying that the rate form follows all the rules that apply to the new database. This is done using the RFIMPEXP.EXE command line program. As it validates the rate form, RFIMPEXP also ensures that all associated riders, contracts, etc. are exported with it.

RFIMPEXP Command Syntax

RFIMPEXP uses the following syntax. Parameter switches are case insensitive; 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, -f "11/01/1999 12:00:00"). 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 **rfimpexp -?** at the command prompt.

rfimpexp [**-c** *connectstring* [**-q** *qualifier*]] **-ffile** *configfilename* [**-lcfg** *logging configuration filename*]

Parameter	Description
-c	*connectstring* is database connection information for the Oracle Utilities Data Repository. This parameter is **required** and must be 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.
-q	*qualifier* is an optional database qualifier. The default is PWRLINE.
-ffile	*configfilename* is the name of the configuration file that defines the working environment of the 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 *configfilename*, the system uses the default (LODESTAR.CFG). For information about the contents of this configuration file, see the *Oracle Utilities Energy Information Platform Configuration Guide*.
-lcfg	*logging configuration filename* 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 RFIMPEXP.LOG in the LOG directory.

The parameters explained above are only those that apply to *all* RFIMPEXP modes. Additional modes are explained on the following pages.

Import Mode

These parameters are specific to Import Mode only:

Parameter	Description
-i	Rate form file, if supplied then it is read and saved. The values in its first line are used for r, o, j, v, and ds, if they are not supplied.
-l	Compile a new rate schedule.
-su	Update rate form if it already exists.

The following are used to rename the rate form:

Parameter	Description
-rCODE	CODE is rate form code (all CODEs must be in UPPERCASE).
-oCODE	CODE is Operating Company code.
-jCODE	CODE is Jurisdiction code.
-v###	### is version number - 1, 2, ... The default is 0.
-dsmm/dd/yyyy	Start date for rate form version. The default is 0.

Note: Either version number or start date may be supplied, but not both.

Export Mode

These parameters are specific to Export Mode only

Parameter	Description
-w	Write (export) to this file.
-ssbiq	Small comment, blank lines, and indentation. q is quick; other s options are ignored, the text is written as-is.
-x	Expand non-contract INCLUDE statements.

The following are used to identify the rate form to export (same rules as import):

Parameter	Description
-rCODE	CODE is rate form code.
-oCODE	CODE is Operating Company code.
-jCODE	CODE is Jurisdiction code.
-v###	### is version number - 1, 2, ...The default is 0.
-dsmm/dd/yyyy	Start date for rate form version. The default is 0.

Dump Mode

Dump Mode overrides all other modes except Resave.

Parameter	Description
-dump	Export all rate forms. They are put in the USER directory by default, one rate form per file (unless -w used). Each file name is OPCOCODE_JURISCODE_RSCODE_nn, where nn depends on the version number and start date. The values for r, o, j, v, and ds (see **Export Mode**) are coded in the first line in each file.
-w	Files put in this directory (must exist).
-ssbiq	Small comment, blank lines, and indentation. q is quick; other s options are ignored, the text is written as-is.
-x	Expand non-contract INCLUDE statements (ignored if -sq).

Global Compile/Resave Mode

This mode overrides all other modes.

Parameter	Description
-lcomp	Reads each rate form and compiles it, without INCLUDE statements (see the *Oracle Utilities Rules Language User's Guide* for more information). This is needed to verify the rate form using the latest Rules Language syntax.
-lincl	Reads each rate form and compiles it, with INCLUDE statements (see the *Oracle Utilities Rules Language User's Guide* for more information). This is needed to verify the rate form using the latest Rules Language syntax.
-lsave	Reads each rate form, compiles it, and saves it back with new formatting. This is needed for correct formatting in Single Step and to verify the rate form using the latest Rules Language syntax. Use one (-lcomp *or* -lsave) but not both.

Importing Groups of Rate Forms

The Import Mode of RFIMPEXP is used to import rate forms one at a time. You can also import groups of rate forms. To import all the rate form files from a specific directory, run the following command from the DOS command line:

```
for %%1 in (<path>*.prg) do rfimpexp -i%%1 ...
```

(... is userid, password, and connect string).

Where:

- <path> indicates the path to the directory where the exported rate form files (*.prg files) are stored.

You must run this command from the "C:\LODESTAR\Bin" directory. If not, you must specify the path to the RFIMPEXP program in the command line. For example, if your rate forms were stored in the C:\LODESTAR\RATES directory, and the RFIMPEXP program was in the C:\LODESTAR\APPS directory, the command line would be:

```
for %%1 in (C:\LODESTAR\RATES\*.prg) do C:\LODESTAR\APPS\rfimpexp -i%%1 ...
```

Index

Symbols

$EFFECTIVE_REVENUE 4-4

Numerics

1 4-24, 4-27

A

account
  definition 1-3
Account Rate Code History Record 1-6
Assigned a name to a charge 1-7
Automatically finding correct values for billing periods 1-7

B

basic building blocks of The Rules Language 1-4
Bill Calculations 1-3
Bill calculations 1-5
Bill History Record
  What it contains 1-3
Bill History records 1-3
Bill History Table
  stores account usage data 1-3
  stores bill determinant values 4-5
Bill History Value Table
  stores account usage data 1-3
  stores bill determinant values 4-5
Bill History Value table 1-3
BILL_TYPE 4-11
BILLDETERMINANT Lookup Table
  defining bill determinant identifiers 4-5
Billing Mode flag A-2
billing modes 3-4
Bills
  Outputting in an electronic format 1-2
Block statements
  Forms of 1-2

C

CANCEL/REBILL rider
  Definition of 1-5
CIS Billing Options
  Billing Component 4-3
Class of customers 1-5
Constants 1-4, 4-23
Contract A-2
  Creating 2-2
  Definition of 1-5
contracts 1-5
creating a rate form 1-3
Current 1-5, A-1
Customer Bill charges 1-3
Customer demand 1-3

D

Definition of an account 1-3

F

Factor Name 1-7
Factor Name table 1-7
Factor Table
  factor names 1-7
Factor Value Table
  stores associated values 1-7
Factors 1-7
Factors and Overrides 1-7
Full Day Bill A-2
Functions 1-3
functions 1-4

G

Graphical user interface 1-3

H

Historical 1-5, A-1
Historical determinants 4-5
How to
  add a statement to the rate form script 2-4
  add rate codes A-4
  copy lines from another rate form script. 2-5
  create a new rate form version record via Data Manager A-5
  create a new rate form version record via the Browser A-5
  create a rate form record A-2
  delete a statement from the rate form script. 2-4
  insert a statement between existing lines. 2-5
  move a statement within the rate form script. 2-5
  open the Rules Language Editor for an existing rate form

version 2-3
open the Rules Language Elements Editor 2-9
save your rate form version script. 2-10
select a parameter
2-9
how to
Compute a customer bill 1-3

I

Identifiers
Override 4-8
If-then-else statement 1-2
Interval data 1-3
Using stem.component to set header data 4-15

L

Language statements 1-2
Computational ability of 1-2
LODESTAR Rules Language
Features of 1-2

M

Meter Value Table
stores bill determinant values 4-5
Metered Billing Determinant Values
Where stored 1-3
middle list box 2-3

O

Operator rules
In arithmetic expressions 4-27
Other Tools for Writing Rate Forms 2-11
Overrides 1-7
Overriding standard charges 1-2

P

Parameters
Selecting with the Rules Language Elements Editor 2-9
Pre-defined identifiers 1-2
Print Detail A-2
Prorate flag 4-7

R

Rate Code Records 1-6
Rate Form 2-3
Rate form 1-4
Capabilities of 1-2
Rate Form record 1-5, A-1
Rate Form records 2-2
Rate Forms
Creating new 2-2
Editing an existing 2-3
Saving 2-10
Types, definition of 1-5
Rate Schedule
Creating 2-2
Definition of 1-5
rate schedules 1-5
Rate Tariff 1-5
Rate Wizard 2-13
Record identifiers 4-24
Revenue Analyses 1-3
Revenue identifiers 4-4
Rider
Creating 2-2
Definition of 1-5
riders 1-5
Rules Language Editor
How to use 2-1
Rules Language Elements Editor 6-1
Rules Language Elements editor 2-6
Rules Language Text Editor 2-11
Running a Rate Form 1-6

S

Sample Rate Form 1-6
Saving
Results of calculations back to the LODESTAR Databases 1-2
Selection statement 1-2
SFUNCALL 6-1
simple identifiers 4-2
Special functions of language statements 1-2
Special identifiers 1-2
Statements
Adding to a rate form 2-4
Copying from rate form into another 2-5
Deleting from a rate form 2-4
Inserting in a rate form 2-5
statements 1-4
Stem.components
Using to set cut header values 4-15
Storing
Values associated with overrides 1-7
Straight-line meter rate 1-6
Sub-form
Definition of 1-5
Symbol Table 3-2
Symbols
/* */ 1-6

T

templates 1-3
The Rules Language 1-4
Time of Use handles 4-6
Time Series Data 1-3
Trial 1-5, A-1
Trial revenues 1-2
types of statements 1-4

U

Usage data
Where it is stored 1-3
Using the Rules Language Editor 2-2

V

versions
where lists are stored 2-3

W

What Customer Usage Data is Used to Calculate Charges? 1-3
What is the LODESTAR Rules Language? 1-2
Writing and Editing a Rate Form 1-6