## Contents

1 Overview
   - Using Formulas: Explained ................................................................. 1-1

2 Writing Formulas
   - Writing a Fast Formula using Expression Editor: Worked Example ........2-1
   - Writing a Fast Formula Using Formula Text: Worked Example ..............2-2
   - Formula Performance Improvements: Explained ....................................2-3
   - Formula Compilation Errors: Explained ............................................. 2-4
   - Formula Execution Errors: Explained ............................................... 2-5
   - FAQs for Writing Formulas ............................................................... 2-6

3 Formula Components
   - Using Formula Components: Explained ............................................. 3-1
   - Formula Statements: Explained .........................................................3-4
   - Classes of Variables: Explained ....................................................... 3-7
   - Naming Variables: Explained ............................................................ 3-8
   - Database Items: Explained ............................................................... 3-9
   - Formula Operators: Explained .......................................................... 3-10
   - Literals: Explained ........................................................................... 3-11
   - Formula Variable Data Types: How They Are Determined ....................3-13
   - Array Variables: Explained ............................................................... 3-14
   - Formula Contexts: Explained ............................................................ 3-16
   - Working Storage Area: Explained ...................................................... 3-17
   - Calling a Formula from a Formula: Explained ..................................... 3-18

4 Formula Functions
   - Formula Functions: Explained ............................................................4-1
   - Text Formula Functions ................................................................. 4-1
   - Numeric Formula Functions ............................................................ 4-4
   - Date Formula Functions ................................................................. 4-6
   - Data Conversion Formula Functions ..................................................4-7
   - Miscellaneous Formula Functions .................................................... 4-8

5 Formulas for Payroll
   - Proration Formulas: Explained ..........................................................5-1
6 Formulas for Absence Management

Accrual Plan Rules: Points to Consider ................................................................. 6-1
Accrual Start Date for New Hires: How It Is Calculated ...................................... 6-2
Accrual Formulas: Critical Choices ........................................................................ 6-3
Custom Accrual Formulas: Points to Consider ..................................................... 6-5
Turning Accrual Plan Rules Into Formula Statements: Examples ...................... 6-7
Accrual Formula Type ............................................................................................. 6-9
Accrual Carryover Formula Type ........................................................................... 6-10
Accrual Ineligibility Formula Type ......................................................................... 6-11
Accrual Plan Formula Functions ............................................................................ 6-11
Formulas for Absence Benefit Plans ..................................................................... 6-17

7 Formulas for Compensation Plans

Compensation Currency Selection Formula Type .................................................... 7-1
Compensation Default and Override Formula Type ............................................... 7-2
Compensation Hierarchy Determination Formula Type ......................................... 7-4
Compensation Person Selection Formula Type ...................................................... 7-5
Compensation Start Date Formula Type ................................................................. 7-6
Compensation End Date Formula Type .................................................................. 7-7
Total Compensation Item Formula Type ............................................................... 7-9
Preface

This Preface introduces the guides, online help, and other information sources available to help you more effectively use Oracle Fusion Applications.

Oracle Fusion Applications Help

You can access Oracle Fusion Applications Help for the current page, section, activity, or task by clicking the help icon. The following figure depicts the help icon.

You can add custom help files to replace or supplement the provided content. Each release update includes new help content to ensure you have access to the latest information. Patching does not affect your custom help content.

Oracle Fusion Applications Guides

Oracle Fusion Applications guides are a structured collection of the help topics, examples, and FAQs from the help system packaged for easy download and offline reference, and sequenced to facilitate learning. You can access the guides from the Guides menu in the global area at the top of Oracle Fusion Applications Help pages.

Note

The Guides menu also provides access to the business process models on which Oracle Fusion Applications is based.

Guides are designed for specific audiences:

- **User Guides** address the tasks in one or more business processes. They are intended for users who perform these tasks, and managers looking for an overview of the business processes. They are organized by the business process activities and tasks.

- **Implementation Guides** address the tasks required to set up an offering, or selected features of an offering. They are intended for implementors. They are organized to follow the task list sequence of the offerings, as displayed within the Setup and Maintenance work area provided by Oracle Fusion Functional Setup Manager.

- **Concept Guides** explain the key concepts and decisions for a specific area of functionality. They are intended for decision makers, such as chief financial officers, financial analysts, and implementation consultants. They are organized by the logical flow of features and functions.
• **Security Reference Manuals** describe the predefined data that is included in the security reference implementation for one offering. They are intended for implementors, security administrators, and auditors. They are organized by role.

These guides cover specific business processes and offerings. Common areas are addressed in the guides listed in the following table.

<table>
<thead>
<tr>
<th>Guide</th>
<th>Intended Audience</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common User Guide</td>
<td>All users</td>
<td>Explains tasks performed by most users.</td>
</tr>
<tr>
<td>Common Implementation Guide</td>
<td>Implementors</td>
<td>Explains tasks within the Define Common Applications Configuration task list, which is included in all offerings.</td>
</tr>
<tr>
<td>Information Technology</td>
<td>Implementors</td>
<td>Explains how to use Oracle Fusion Functional Setup Manager to plan, manage, and track your implementation projects, migrate setup data, and validate implementations.</td>
</tr>
<tr>
<td>Management, Implement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applications Guide</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technical Guides</td>
<td>System administrators, application developers, and technical members of implementation teams</td>
<td>Explain how to install, patch, administer, and customize Oracle Fusion Applications.</td>
</tr>
</tbody>
</table>

For guides that are not available from the Guides menu, go to Oracle Technology Network at http://www.oracle.com/technetwork/indexes/documentation.

**Other Information Sources**

**My Oracle Support**

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

Use the My Oracle Support Knowledge Browser to find documents for a product area. You can search for release-specific information, such as patches, alerts, white papers, and troubleshooting tips. Other services include health checks, guided lifecycle advice, and direct contact with industry experts through the My Oracle Support Community.

**Oracle Enterprise Repository for Oracle Fusion Applications**

Oracle Enterprise Repository for Oracle Fusion Applications provides visibility into service-oriented architecture assets to help you manage the lifecycle of your software from planning through implementation, testing, production,
and changes. In Oracle Fusion Applications, you can use the Oracle Enterprise Repository for Oracle Fusion Applications at http://fusionappsoer.oracle.com for:

- Technical information about integrating with other applications, including services, operations, composites, events, and integration tables. The classification scheme shows the scenarios in which you use the assets, and includes diagrams, schematics, and links to other technical documentation.
- Publishing other technical information such as reusable components, policies, architecture diagrams, and topology diagrams.

**Documentation Accessibility**

For information about Oracle’s commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/us/corporate/accessibility/index.html.

**Comments and Suggestions**

Your comments are important to us. We encourage you to send us feedback about Oracle Fusion Applications Help and guides. Please send your suggestions to oracle_fusion_applications_help_ww_grp@oracle.com. You can use the **Send Feedback to Oracle** link in the footer of Oracle Fusion Applications Help.
Overview

Using Formulas: Explained

Fast formulas are generic expressions of calculations or comparisons that you want to repeat with different input variables.

You can use fast formulas to:

- Calculate payrolls
- Define the rules for paid time off accrual plans
- Define custom calculations for benefits administration
- Validate element inputs or user-defined tables
- Edit the rules for object group population for elements or people
- Calculate absence duration
- Define custom configuration for compensation

Calculate Payrolls

Write payroll calculations and skip rules for elements that you define to represent earnings and deductions. Associate more than one formula with each element to perform different processing for employee assignments with different statuses. You can define elements and formulas for earnings and deductions with highly complex calculations requiring a number of different calls to the database.

Define the Rules for Paid Time Off Accrual Plans

Edit the delivered accrual type formulas or write your own. Each accrual plan needs two formulas: one to calculate the gross accrual and the other to return information to the PTO carry-over process.

Define Custom Calculations for Benefits Administration

Configure your plan design to the requirements of your enterprise. Formulas provide a flexible alternative to the delivered business rules for such purposes as:
• Date calculations, such as enrollment start and end dates, rate or coverage start and end dates, waiting periods and enrollment periods, or action item due dates

• Calculations of rate and coverage amount, minimum and maximum, or upper and lower limits

• Certification requirements

• Partial month and proration calculations

• Eligibility and participation evaluation

For example, you can write a formula to calculate benefits eligibility for those cases where the provided eligibility criteria does not accommodate your particular requirements.

For more information, see Benefits Fast Formula Reference Guide (1456985.1) on My Oracle Support at https://support.oracle.com.

**Validate Element Inputs or User-Defined Tables**

Validate user entries into element input values using lookups or maximum and minimum values. However, for more complex validations write a formula to check the entry. Also, use a formula to validate entries in user tables.

**Edit the Rules for Populating Work Relationship or Payroll Relationship Groups**

Define criteria to dynamically populate a payroll relationship group or work relationship group. When you create a formula of type Payroll Relationship Group or Work Relationship Group, the Create Fast Formula page provides an expression editor to help you build the selection criteria.

**Calculate Absence Duration**

Calculate the duration of an absence from the start and end dates.

**Define Custom Configuration for Compensation**

Extend the existing flexibility of compensation plan configuration by writing formulas to customize:

- Start and end dates for compensation allocations under individual compensation plans

- Person selection, hierarchy determination, column default values, and currency selection for workforce compensation plans

- The source of items displayed in total compensation statements
Writing a Fast Formula using Expression Editor: Worked Example

This example demonstrates how to create a fast formula that groups executive workers for reporting and processing. All executive workers are in department EXECT_10000. Once the formula is created it will be added as object group parameters so that only those workers in department EXECT_10000 are used in processing.

The following table summarizes key decisions for this scenario:

<table>
<thead>
<tr>
<th>Decisions to Consider</th>
<th>In This Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is the formula for a specific legislative data group?</td>
<td>Yes, InVision</td>
</tr>
<tr>
<td>What is the formula type for this formula?</td>
<td>Payroll Relationship Group</td>
</tr>
</tbody>
</table>

Creating a Fast Formula Using the Expression Editor

1. On the Payroll Calculation Tasks page, click Manage Fast Formulas to open the Manage Fast Formulas page.

2. On the Manage Fast Formula page, click the Create icon to create a new formula.

3. On the Create Fast Formula page, complete the fields as shown in this table.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formula Name</td>
<td>Executive Payroll Relationship Group</td>
</tr>
<tr>
<td>Type</td>
<td>Payroll Relationship Group</td>
</tr>
<tr>
<td>Description</td>
<td>Executive Workers</td>
</tr>
<tr>
<td>Legislative Data Group</td>
<td>Vision LDG</td>
</tr>
<tr>
<td>Effective As-of Date</td>
<td>1-Jan-2010</td>
</tr>
</tbody>
</table>

4. Click Continue.

5. In the Formula Details section, click Add After to add a row to enter the fields in this table.

<table>
<thead>
<tr>
<th>Conjunction</th>
<th>Database Item Name</th>
<th>Data Type</th>
<th>Operand</th>
<th>Literal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>DEPARTMENT</td>
<td>Character</td>
<td>=</td>
<td>'EXECT_10000'</td>
</tr>
<tr>
<td>Field</td>
<td>Value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formula Name</td>
<td>Manager Range of Scheduled Hours</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formula Type</td>
<td>Range of Scheduled Hours</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Manager's Range of Hours</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Effective Start Date</td>
<td>1-Jan-2010</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. Click **Continue**.

5. Enter the following formula details in the Formula Text section:

```sql
/* DATABASE ITEM DEFAULTS BEGIN */
DEFAULT FOR asg_job IS ' '
/* DATABASE ITEM DEFAULTS END */
JOB_1 = ASG_JOB
IF JOB_1 = 'Manager' then

Then SELECT_EMP Character = 'YES'
ELSE SELECT_EMP Character = 'NO'

6. Click **Compile**.

7. Click **Save**.

**Writing a Fast Formula Using Formula Text: Worked Example**

This example demonstrates, using the text editor, how to create a fast formula that returns the range of scheduled hours for managers and a different range for other workers.

The following table summarizes key decisions for this scenario:

<table>
<thead>
<tr>
<th>Decisions to Consider</th>
<th>In This Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is the formula for a specific legislative data group?</td>
<td>No, this is a global formula that can be used by any legislative data group.</td>
</tr>
<tr>
<td>What is the formula type for this formula?</td>
<td>Range of Scheduled Hours</td>
</tr>
<tr>
<td>Are there any contexts used in this formula?</td>
<td>No</td>
</tr>
<tr>
<td>Are there any database item defaults?</td>
<td>Yes, ASG_JOB</td>
</tr>
<tr>
<td>Are there any input value defaults?</td>
<td>No</td>
</tr>
<tr>
<td>What are the return values?</td>
<td>MIN_HOURS, MAX_HOURS, FREQUENCY</td>
</tr>
</tbody>
</table>

**Creating a Fast Formula Using the Text Editor to Determine a Manager's Scheduled Hours**

1. On the Payroll Calculation Tasks page, click Manage Fast Formulas to open the Manage Fast Formulas page.

2. On the Manage Fast Formula page, click the **Create** icon to create a new formula.

3. On the Create Fast Formula page, complete the fields as shown in this table.

4. Click **Continue**.

5. Enter the following formula details in the Formula Text section:

```sql
/* DATABASE ITEM DEFAULTS BEGIN */
DEFAULT FOR asg_job IS ' '
/* DATABASE ITEM DEFAULTS END */
JOB_1 = ASG_JOB
IF JOB_1 = 'Manager' then

Then SELECT_EMP Character = 'YES'
ELSE SELECT_EMP Character = 'NO'

6. Click **Compile**.

7. Click **Save**.
(MIN_HOURS = 25
MAX_HOURS = 40
FREQUENCY = 'H')
else
(MIN_HOURS = 20
MAX_HOURS = 35
FREQUENCY = 'H')
return MIN_HOURS, MAX_HOURS, FREQUENCY

6. Click Compile.
7. Click Save.

Formula Performance Improvements: Explained

When writing formulas there are a number of techniques to follow to ensure your formulas are easy to read, use, understand, and process efficiently.

Variable Names and Aliases

To improve readability, use names that are brief yet meaningful. Use aliases if the names of database items are long. Name length has no effect on performance or memory usage.

Inputs Statements

Use INPUTS statements rather than database items whenever possible. It speeds up the running of your payroll by eliminating the need to access the database for the input variables.

An example of inefficient formula without INPUTS statement is:
SALARY = SALARY_ANNUAL_SALARY / 12
RETURN SALARY

An example of efficient use of INPUTS statements is:
INPUTS ARE ANNUAL_SALARY
SALARY = ANNUAL_SALARY / 12
RETURN SALARY

Database Items

Do not refer to database items until you need them. People sometimes list at the top of a formula all the database items the formula might need, thinking this helps the formula process more quickly. However, this in fact slows processing by causing unnecessary database calls.

An example of an inefficient use of database items is:
S = SALARY
A = AGE
IF S < 20000 THEN
IF A < 20 THEN
TRAINING_ALLOWANCE = 30
ELSE
TRAINING_ALLOWANCE = 0
An example of an efficient use of database items is:
IF SALARY < 20000 THEN
IF AGE < 20 THEN
TRAINING_ALLOWANCE = 30
ELSE
TRAINING_ALLOWANCE = 0
The first example always causes a database fetch for AGE whereas the second only fetches AGE if salary is less than 20000.

Balance Dimensions

Wherever possible, use balance dimensions for single assignments only in formulas. Multiple assignments require more calculation time, leading to
slower processing time. The number of multiple assignments in a payroll is not normally high, and the presence of a small number does not lead to any significant increase in overall processing time. However, there could be a problem if you unnecessarily link balance dimensions for multiple assignments into general formulas.

**Formula Compilation Errors: Explained**

Compilation errors display in the Manage Fast Formulas page when you compile the formula. The formula compiler returns line numbers starting at 1 from the beginning of a formula, and character positions starting at 1 from the beginning of a line in its error messages. The compiler aborts compilation when an error is encountered.

**Common Compilation Errors**

This table lists the type and description of several common formula compilation errors.

<table>
<thead>
<tr>
<th>Formula Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax Error</td>
<td>The formula text violates the grammatical rules for the formula language. An example is using IF1 instead of IF for an IF statement.</td>
</tr>
<tr>
<td>Incorrect Statement Order</td>
<td>ALIAS, DEFAULT, or INPUT statements come after other statements.</td>
</tr>
<tr>
<td>Misuse of ASSIGNMENT Statement</td>
<td>Occurs when any of these conditions occurs:</td>
</tr>
<tr>
<td></td>
<td>• An ASSIGNMENT assigns a value to a database item.</td>
</tr>
<tr>
<td></td>
<td>• A context is assigned a value externally to a CHANGE-CONTEXTS statement.</td>
</tr>
<tr>
<td></td>
<td>• A non-context variable is assigned a value within a CHANGE-CONTEXTS statement.</td>
</tr>
<tr>
<td>Misuse of ALIAS Statement</td>
<td>An ALIAS statement may only be used for a database item.</td>
</tr>
<tr>
<td>Missing DEFAULT Statement</td>
<td>A database item with defaulting specified must have a DEFAULT statement.</td>
</tr>
<tr>
<td>Misuse of DEFAULT Statement</td>
<td>A DEFAULT statement is specified for a variable other than an input or database item.</td>
</tr>
<tr>
<td>Uninitialized Variable</td>
<td>The compiler detects that a variable is uninitialized when used. The compiler cannot do this in all cases. This error often occurs when you want to use a database item, but a database item is not available in the formula.</td>
</tr>
<tr>
<td>Missing Function Call</td>
<td>A function call is not recognized. The combination of return type, function name, and parameter types does not match any available function.</td>
</tr>
<tr>
<td>Incorrect Operator Usage</td>
<td>An instance of a formula operator use does not match the permitted uses of that operator. For example, the + operator has two permitted uses. The operands are both of data type NUMBER, or both of data type TEXT.</td>
</tr>
</tbody>
</table>
### Inconsistent Data Type Usage

A formula variable is being used as if it is of more than one data type. Or a database item or context is being used with the wrong data type.

For example, Variable A is assigned a **NUMBER** value at the start of the formula, but a **TEXT** value later in the formula.

### EXIT Statement Not Within **WHILE** Loop

A condition that eventually becomes false, or an **EXIT** call for exiting the loop does not exist.

### Misuse of Context

A variable is used as a context, or a context is used as a variable.

For example, **AREA1** is assigned a value as an ordinary variable, but later in the formula **AREA1** used as a context in a **GET_CONTEXT** call.

## Formula Execution Errors: Explained

Fast formula execution errors occur when a problem arises while a formula is running. The usual cause is a data problem, either in the formula or in the application database. These errors contain the formula line number where the error occurs.

### Formula Execution Errors

This table lists the type and description of each formula execution error.

<table>
<thead>
<tr>
<th>Formula Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Uninitialized Variable</strong></td>
<td>Where the formula compiler cannot fully determine if a variable or context is initialized when it is used, it generates code to test if the variable is initialized. When the formula executes and the variable or context is not initialized an error is raised.</td>
</tr>
<tr>
<td><strong>Divide by Zero</strong></td>
<td>Raised when a numeric value is divided by zero.</td>
</tr>
<tr>
<td><strong>No Data Found</strong></td>
<td>Raised when a non-array type database item unexpectedly fails to return any data. If the database item can return no data then it should allow defaulting. This error is also raised from within a formula function. The cause is an error in the formula function code.</td>
</tr>
<tr>
<td><strong>Too Many Rows</strong></td>
<td>Raised when a non-array type database item unexpectedly returns more than a single row of data. The cause is an incorrect assumption made about the data being accessed. This error can also be raised from within a formula function. The cause is an error in the formula function code.</td>
</tr>
<tr>
<td><strong>NULL Data Found</strong></td>
<td>Raised when a database item unexpectedly returns a NULL data value. If the database item can return a NULL value then defaulting is allowed.</td>
</tr>
<tr>
<td>Error Description</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Value Exceeded Allowable Range</td>
<td>Raised for a variety of reasons, such as exceeding the maximum allowable length of a string.</td>
</tr>
<tr>
<td>Invalid Number</td>
<td>Raised when an attempt is made to convert a non-numeric string to a number.</td>
</tr>
<tr>
<td>User Defined Function Error</td>
<td>Raised from within a formula function. The error message text is output as part of the formula error message.</td>
</tr>
<tr>
<td>External Function Call Error</td>
<td>A formula function returned an error, but did not provide any additional information to the formula code. The function might have output error information to the logging destination for the executing code.</td>
</tr>
<tr>
<td>Function Returned NULL Value</td>
<td>A formula function returned a NULL value.</td>
</tr>
<tr>
<td>Too Many Iterations</td>
<td>A single WHILE loop, or a combination of WHILE loops, has exceeded the maximum number of permitted iterations. The error is raised to terminate loops that could never end. This indicates a programming error within the formula.</td>
</tr>
<tr>
<td>Array Data Value Not Set</td>
<td>The formula attempted to access an array index that has no data value. This is an error in the formula code.</td>
</tr>
<tr>
<td>Invalid Type Parameter for WSA_EXISTS</td>
<td>An invalid data type was specified in the WSA_EXISTS call.</td>
</tr>
<tr>
<td>Incorrect Data Type For Stored Item</td>
<td>When retrieving an item using WSA_GET, the item's actual data type does not match that of the stored item. This is an error within the calling formula.</td>
</tr>
<tr>
<td>Called Formula Not Found</td>
<td>The called formula could not be resolved when attempting to call a formula from a formula. This could be due to an error in the calling formula, or because of installation issues.</td>
</tr>
<tr>
<td>Recursive Formula Call</td>
<td>An attempt was made to call a formula from itself. The call could be directly or indirectly via another called formula. Recursive formula calling is not permitted.</td>
</tr>
<tr>
<td>Input Has Different Types In Called and Calling Formulas</td>
<td>When calling a formula from a formula, the actual formula input data type within the called formula does not match the data type specified from the calling formula.</td>
</tr>
<tr>
<td>Output Has Different Types In Called and Calling Formulas</td>
<td>When calling a formula from a formula, the actual formula output data type within the called formula does not match the data type specified from the calling formula.</td>
</tr>
<tr>
<td>Too Many Formula Calls</td>
<td>There are too many formula from formula calls. This is due to a problem with the formulas.</td>
</tr>
</tbody>
</table>

**FAQs for Writing Formulas**

**When do I run the Compile Formula process?**

If you need to compile many fast formulas at the same time, you can run the Compile Formula process on the Submit a Process or Report page. Also, if you...
make any changes to a function after you have compiled a formula that uses it, you need to recompile the formula for the changes to take effect.

**What's the difference between a formula compilation error and an execution error?**

Compilation errors occur in the Manage Fast Formulas page when you compile the formula. An error message explains the nature of the error. Common compilation errors are syntax errors resulting from typing mistakes.

Execution errors occur when a problem arises while a formula is running. The usual cause is a data problem, either in the formula or in the application database.
Using Formula Components: Explained

When developing a formula you must understand formula language, the rules that the application imposes on the formula, and the calculation requirements. Formulas are created using various components.

Formula components include:

- Assignment statements
- Return statements
- Variables
- Input statements
- Expressions
- Conditions
- Comments

Note

There are several other components used in formulas. These include literals, database items, working storage area, calls to other formulas, functions, and operators.

To illustrate how each component is used in a formula, suppose you wanted to calculate the pay value for the element WAGE by multiplying the number of hours an employee works each week by the hourly rate. The formula can be written as follows:

\[
WAGE = HOURS\_WORKED \times HOURLY\_RATE \\
RETURN WAGE
\]

Assignment Statements

The first line is an assignment statement that simply assigns a value to the element WAGE.

Return Statements

The second line is a return statement that passes back the WAGE value to the payroll run. A return statement can be used to stop the formula execution without passing back any values.
Variables

Local variables occur in a single formula only. You can change a local variable within the formula by assigning a value to it using an assignment statement. To calculate the \texttt{WAGE} value, the formula needs to get the value for the variable \texttt{HOURS_WORKED}.

You can use local variables to store data in a formula. You might want to hold data temporarily while you perform some other calculations, or to pass data back to the application. Below is an example showing the use of a variable, \texttt{ANNUAL_LEAVE}.

```plaintext
/* Formula: Annual Leave Formula */
IF YEARS_SERVICE >= 10
   THEN
   ANNUAL_LEAVE = 25
   ELSE
   ANNUAL_LEAVE = 20 + FLOOR (YEARS_SERVICE/2)
RETURN ANNUAL_LEAVE
```

Input Statements

The \texttt{HOURS_WORKED} could be an input value of the element \texttt{WAGE}. To pass the element input values to the formula during processing, define an inputs statement as follows:

```plaintext
INPUTS ARE HOURS_WORKED
WAGE = HOURS_WORKED * HOURLY_RATE
RETURN WAGE
```

Note

This is a payroll application example. The name used in the input statement must be the same as the name of the element input value, and multiple words must be joined by underscores. Other input statements that have nothing to do with elements would have their own rules for formula input variables. In this example, the input variable \texttt{HOURS_WORKED} is numeric. If the input variable is not numeric, you must specify the type. For example,

```plaintext
INPUTS ARE START_DATE (DATE)
```

Expressions

Each function or calculation is one expression, and you can nest expressions to create more complex calculations. Brackets can be used to control the order in which calculations are performed. Expressions within brackets are evaluated first. Within nested brackets, evaluation proceeds from the least inclusive set to the most inclusive set. When brackets are not used the following hierarchal order or execution is implied: multiplication and division then addition and subtraction.

Expressions combine constants and variables with operators (+, -, *, /), array methods, and functions to return a value of a certain data type. For example, the expression \((3 + 2)\) returns a value of 5, and is of \texttt{NUMBER} data type. The format of an expression is:

```plaintext
SUBEXPRESSION [operator SUBEXPRESSION ...]
```
A number of sub-expressions can combine in a single expression. For example, the sub-expressions \((3 + 2)\) and \(\text{MONTHS\_BETWEEN}(\text{start\_date}, \text{end\_date})\) can combine in a single expression as follows:

\[(3 + 2) + \text{MONTHS\_BETWEEN}(\text{start\_date}, \text{end\_date})\]

Expressions can also be used inside functions, such as:

\[\text{salary} = \text{GREATEST}(\text{minimum\_wage}, (\text{hourly\_rate} \times \text{hours\_worked}))\]

Operands in an expression are usually of the same data type which is the data type of the expression as a whole. For example, in the following expression all the operands are numeric and the expression itself is numeric:

\[\text{GREATEST}(\text{MINIMUM\_WAGE}, (\text{HOURLY\_RATE} \times \text{HOURS\_WORKED})) + \text{BONUS}\]

The operands for the above expression are \text{BONUS}, and the return value from \text{GREATEST}. The arguments for \text{GREATEST} are separate expressions.

**Conditions**

Conditions are used to process expressions based on whether a certain condition occurs. For example,

\[
\begin{align*}
\text{TRAINING\_ALLOWANCE} & = 0 \\
\text{IF (AGE < 20) THEN} \\
\text{TRAINING\_ALLOWANCE} & = 30
\end{align*}
\]

This formula checks if the condition \((\text{AGE < 20})\) is true or false. If it is true, the formula processes the statement that follows the word \text{THEN}. If the condition is not true, the formula ignores this statement.

**Comments**

Use comments to explain how all or part of a formula is used. Also, you can change some formula lines into comments until they are ready to be used. Comments are designated by the comment delimiters of /* and */. Anything written inside these delimiters is a comment. You can place comments anywhere within a formula. The beginning of a formula should contain the following comments:

- The formula title and a short purpose statement.
- A description of the formula inputs.
- A list of variables and literals that may require updating.
- An explanation of the formula's calculation.
- The dates of any modifications, the name of the person modifying the formula, and the reason for the change.

An example of a comment is:

\[
/* \text{Use this formula to determine the bonus percentage for staff} */ \\
\text{INPUTS ARE} \text{SALARY\_AMOUNT}, \\
\text{START\_DATE (DATE)}, \\
\text{END\_PERIOD\_DATE (DATE)}, \\
\text{BONUS\_PERCENTAGE} /* \text{Decided at board level.} */
\]

**Note**

Do not put a comment within a comment. This causes a syntax error when the formula is compiled.
Formula Statements: Explained

Statements are instructions that a formula carries out.

When working with statements, it is important to have knowledge of:

- Statement types
- Ordering statement
- Grouping statements

When using statements in a formula you must enter them in the following order: alias statements, default statements, input statements, then other statements.

Statement Types

Use the alias statement to define another name, or alias, for existing variables. Declare aliases for database items and global values. An example of an alias statement is:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
</table>
| ALIAS     | ALIAS name1 AS name2
          | ALIAS OVERTIME_QUALIFYING_LENGTH_OF_AS OT_QLS | Allows a different name to be used instead of a database item name. Database items are named by the system and sometimes these names are too long to conveniently use in a formula. Use the ALIAS statement to shorten the name of a database item. Once the ALIAS is created, use it instead of the database item name. Using an alias is more efficient than assigning the database item to a local variable with a short name. |
| ASSIGNMENT | variable = expression
            | array[index] = expression
            | rate = HOURLy_RATE + 14
<pre><code>        | WAGE = HOURS_WORKED * rate | Assigns an expression value to a variable or an array variable at an index position. A formula evaluates the expression on the right hand side of the statement, and places its result in the variable you name on the left hand side. The left side of an assignment statement must always be a local variable because a formula can only change the value of local variables. Within a CHANGE-CONTEXTS statement only contexts may be assigned values. External to a CHANGE-CONTEXTS statement only input, output, and local variables may be assigned values. |
</code></pre>
<table>
<thead>
<tr>
<th>Formula Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CHANGE-CONTEXTS</strong></td>
<td>Allows one or more contexts to be changed within a formula. The new values are assigned by one or more ASSIGNMENT statements.</td>
</tr>
<tr>
<td>(context1 = expression1 [context2 = expression2] ...)</td>
<td></td>
</tr>
<tr>
<td>CHANGE_CONTEXTS(AREA1 = TAX_REPORTING_UNIT_INCOME_TAX)</td>
<td></td>
</tr>
<tr>
<td>CHANGE_CONTEXTS(DEDUCTION_TYPE = 'SBJ_TO_REGULAR_TAX')</td>
<td></td>
</tr>
<tr>
<td>L_TAXATION_METHOD = 'NONE'</td>
<td></td>
</tr>
<tr>
<td>EXECUTE('TAXABILITY_RULE_EXISTS')</td>
<td></td>
</tr>
<tr>
<td>IF GET_OUTPUT('TR_EXISTS', 'N') = 'Y' THEN</td>
<td></td>
</tr>
<tr>
<td>L_TAXATION_METHOD = 'REGULAR_TAX'</td>
<td></td>
</tr>
<tr>
<td>) /* DEDUCTION_TYPE context change undone here. */</td>
<td></td>
</tr>
<tr>
<td>) /* AREA1 context change undone here. */</td>
<td></td>
</tr>
<tr>
<td><strong>DEFAULT</strong></td>
<td>The DEFAULT FOR statement allows a value to be used for a formula input if a value is not provided. The DEFAULT FOR statement allows a value to be used for a database item if the database item value is not found, or if a non-array database item value is NULL.</td>
</tr>
<tr>
<td>DEFAULT FOR variable IS literal</td>
<td></td>
</tr>
<tr>
<td>DEFAULT_DATA_VALUE FOR variable IS literal</td>
<td></td>
</tr>
<tr>
<td>DEFAULT FOR HOURLY_RATE IS 3.00</td>
<td></td>
</tr>
<tr>
<td>INPUTS ARE HOURLY_RATE X = HOURS_WORKED * HOURLY_RATE</td>
<td></td>
</tr>
<tr>
<td><strong>EXIT</strong></td>
<td>The EXIT statement allows a value to be used for an array database item where individual data values are NULL.</td>
</tr>
<tr>
<td>EXIT</td>
<td></td>
</tr>
<tr>
<td>FOUND = -1 /* -1 is not a valid index for A. */</td>
<td></td>
</tr>
<tr>
<td>I = A.FIRST(-1)</td>
<td></td>
</tr>
<tr>
<td>WHILE (A.EXISTS(I)) LOOP</td>
<td></td>
</tr>
<tr>
<td>/* EXIT-clause for early exit. */</td>
<td></td>
</tr>
<tr>
<td>IF A[I] = KEY THEN</td>
<td></td>
</tr>
<tr>
<td>FOUND = I /* Exit the loop. */</td>
<td></td>
</tr>
<tr>
<td>EXIT;</td>
<td></td>
</tr>
<tr>
<td>I = A.NEXT(I,-1)</td>
<td></td>
</tr>
</tbody>
</table>

Some database items are defined to require a default value because they could return no data or NULL values from the database.

Used to immediately exit from the enclosing WHILE loop. The EXIT statement cannot be used outside of a WHILE loop.
### Formula Calling

**SET_INPUT**

```
SET_INPUT(input [,value])
```

**EXECUTE**

```
EXECUTE(formula)
```

The formula `RATE_FORMULA` is called to get a value for `HOURLY_RATE`. `RATE_FORMULA`.

```
SET_INPUT('UNIT','Hourly')
EXECUTE('RATE_FORMULA')
HOURLY_RATE =
GET_OUTPUT('RATE',0.0)
WAGE = HOURS_WORKED *
HOURLY_RATE
RETURN WAGE
```

Instead of writing large formulas, common calculations can be put into their own smaller formulas. These calculation formulas can then be called from other formulas that need the calculation.

### IF

**IF**

```
IF condition THEN statements
ELSE statements
```

Allows one or more statements to be executed provided a condition evaluates as true. The IF ELSE statement also specifies a set of statements to execute if the condition evaluates to false.

The IF statement is the only statement that can have other statements nested within it, including other IF statements.

```
IF (AGE < 20) THEN
   TRAINING_ALLOWANCE = 30
ELSE
   TRAINING_ALLOWANCE = 40
```

### INPUT

**INPUTS ARE**

```
input1 [, input2] ...
```

```
INPUTS ARE HOURS_WORKED
WAGE = HOURS_WORKED *
   HOURLY_RATE
RETURN WAGE
```

Lists the input variables for the formula. There is only one INPUT statement in a formula.

### RETURN

**RETURN**

```
RETURN [ output1 [,output2] ...]
```

```
INPUTS ARE HOURS_WORKED
IF HOURS_WORKED <= 10 THEN 
    RETURN
    /* This is ignored. */
    BONUS = 10
*/
/* This is executed if HOURS_WORKED > 10. */
BONUS = 50
RETURN BONUS
```

Causes a formula to stop executing immediately. A formula output variable must appear in the RETURN statement that stopped the formula for its value to be returned to the caller. Multiple return statements are allowed in a formula.

### WHILE

**WHILE**

```
WHILE condition LOOP statements
```

```
WHILE condition LOOP statements
```

```
IN this example, 'A' is an array variable with a numerical index.
/ * -1234 is not a valid index for A in this instance, so use as default. */
NI = A.FIRST(-1234)
WHILE A.EXISTS(NI) LOOP
    VA = A[NI] /* Do some processing with element at index NI. */
    NI = A.NEXT(NI,-1234) /* Go to next index. */
```

The WHILE loop executes a number of statements as long as a condition evaluates to true.

To prevent endless looping, an error occurs if the WHILE statement loop performs an excessive number of iterations.
<table>
<thead>
<tr>
<th>WORKING STORAGE</th>
<th>WSA_DELETE([item]) - Deletes values from the storage area.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WSA_EXISTS(item[,type]) - Determine if an item exists.</td>
</tr>
<tr>
<td></td>
<td>WSA_GET(item, value) - Fetches values from the storage area.</td>
</tr>
<tr>
<td></td>
<td>WSA_SET(item, value) - Sets values from the storage area.</td>
</tr>
</tbody>
</table>

In the following example, a number of rates are set up:

```c
/* Formula: RATE_SETTER */
WSA_SET('RATE:HOURLY1',3.5)
WSA_SET('RATE:HOURLY2',4.0)
WSA_SET('RATE:HOURLY3',4.5)
WSA_SET('RATE_FLAG','Y') /* Flag to say that the rates have been set. */
```

Use the working storage area statements to store reference data.

---

**Ordering Statements**

Statements need to be placed in a formula in the following order:

1. ALIAS statements, if any
2. DEFAULT statements, if any
3. INPUT statements, if any
4. Other statements

**Grouping Statements**

If you want to group more than one statement, under IF - THEN statement, ELSE clauses, WHILE-loop, or CHANGE_CONTEXTS, enclose the group of statements within brackets. In the absence of brackets, the preceding statement only applies to the first statement.

**Correct example:**

```c
I = A.FIRST
WHILE (A.EXISTS(I)) LOOP
  
  A[I] = I
  I = A.NEXT(I,-1)

```

**Incorrect example:**

```c
I = A.FIRST
WHILE (A.EXISTS(I)) LOOP
  A[I] = I
  I = A.NEXT(I,-1) /* This is not executed as part of the loop. */
```

---

**Classes of Variables: Explained**

Formula variables can have frequently changing values. The variable's data type determines the type of information it holds. You do not have to specify
what type you want a variable to be. The formula works out the type from how you use the variable. For example, if you set a variable to 'J. Smith', this is interpreted as a TEXT variable. The system also warns you if you try to perform any inconsistent operations, such as trying to add a number to a text string.

There are three classes of variables:

- Input variables appear in INPUTS statements and bring values into a formula.
- Output variables appear in RETURN statements and return values from a formula. A variable can be both an input and output.
- Local variables are only used within one formula.

Variable values can be changed using an assignment statement and referenced within expressions. However, if a variable has not been given a value when referenced, the formula will raise an error.

Naming Variables: Explained

There are two naming schemes for variables. In both cases, the maximum size of a variable name is 255 characters.

In the first naming scheme, variables have names comprising one or more words, joined by underscores. The words must each start with a letter and can be followed by a combination of letters, and digits.

In the second naming scheme, variable names begin and end with double quotes ("). Between the quotes, any printable characters can be used such as "This is a quoted variable name!". Note that any word consisting of only digits could be mistaken for numbers.

Formulas are not case sensitive. For example, the variable named EMPLOYEE_NAME is the same as the variable employee_name.

The following reserved words cannot be used as the names of variables:

```
ALIAS
AND
ARE
AS
CHANGE_CONTEXTS
DEFAULT
DEFAULT_DATA_VALUE
DEFAULTED
ELSE
EMPTY_DATE_NUMBER
EMPTY_NUMBER_NUMBER
EMPTY_TEXT_NUMBER
EMPTY_DATE_TEXT
EMPTY_NUMBER_TEXT
EMPTY_TEXT_TEXT
EXIT
FOR
IF
INPUTS
IS
LIKE
LOOP
NEED_CONTEXT
NOT
```
Database Items: Explained

Database items exist in the application database and have associated code that the system uses to find the data. All database items are read-only variables. An attempt to write to a database item causes a compiler error.

Database item values cannot be changed within a formula. Database items exist in the application database and have a label, hidden from users, which the system uses to find the data. Database items are specific to the context in which you use them.

There are two types of database items:

- Static
- Dynamic

**Static Database Items**

Static database items are predefined. They include standard types of information, such as the sex, birth date, and work location of an employee, or the start and end dates of a payroll period.
Dynamic Database Items

Dynamic database items are generated from your definitions of:

- **Elements**
  
  The element name is the database item name prefix.

- **Balances**
  
  The balance name followed by balance dimension name is the database item name.

- **Formula global values**
  
  The global value name is the database item name.

- **Input values**
  
  The element and input value names are the database item name prefix.

Array Database Items

There are also array database items. Array database items have an index type of NUMBER with indexes starting at 1 and increasing by 1 without gaps.

```java
/* 1 is the starting index for an array database item. */
I = 1
WHILE DBI.EXISTS(I) LOOP
  V = DBI[I] /* Do some processing with element at index I. */
  I = I + 1 /* Array database items indexes go up in steps of 1. */
END LOOP;
```

The default data value for a statement is used to set a default value in the case where an array database item could return a NULL value for an element. This is an extension of standard database item behavior. There can only be one default data value for a statement for each array database item and it must appear at the start of the formula.

An example of a default data value for a statement:

```
DEFAULT_DATA_VALUE FOR A IS 0
INPUTS ARE B, C
```

An example of an array database item usage error case:

```java
/* Array database item A. */
A[1] = 1
A = B
A.DELETE(1)
A.DELETE
```

Formula Operators: Explained

An expression may contain arithmetic operators, which determine how variables and literals are manipulated. For example, the operator + indicates that two items are added together. It is also used for string concatenation.
Types of Operators

The operator types are described in the following table.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>A = B + 1</td>
</tr>
<tr>
<td>+</td>
<td>String concatenation</td>
<td>A = 'Hello' + 'World'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B = 'Hello'</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>A = B - 1</td>
</tr>
<tr>
<td>-</td>
<td>Unary minus</td>
<td>A = -B</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>A = B * C</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>A = B / C</td>
</tr>
</tbody>
</table>

Using Operators

The arithmetic operators, subtraction, multiplication, and division, can only be used with numeric operands. The addition operator can be used with numeric or text operands. The operands can be variables, literals, or sub-expressions. A formula error occurs if:

- The second operand of a division equals zero
- The result of multiplication is too large

What is too large is determined by the normal limits in the database. For string concatenation, if the result is longer than 255 characters, a formula error is raised.

Expressions are evaluated in order from left to right. The unary minus has precedence over the other operators because it applies directly to a single sub-expression. The multiplication and division operators take precedence over addition and subtraction. For example, the expression 1 + 2 * 3 evaluates to 7 rather than 9. Brackets can be used to change precedence. For example, (1 + 2) * 3 evaluates to 9.

Literals: Explained

Every piece of information that you can manipulate or use in a fast formula is a literal.

There are four types of literals:

- Numeric
- Text
- Date
- Array
Numeric Literals

Enter numeric literals without quotes. Precede negative numbers with a minus sign (-). Numbers may have a decimal component after a decimal point. Do not use exponents and floating point scientific notations. Do not use commas or spaces in a number.

Examples of numeric literals are:

- 63
- 3.55
- -2.3
- -.033
- -.2
- 10000

Text Literals

Enclose text literals in single quotes. They may contain spaces. You can represent the single quote character in a text constant by writing it twice ("'). Note that this is not the same as the double quote (").

Examples of text literals are:

- 'J. Smith'
- 'P O''Donnell'
- '1234'
- 'Manager'
- '12 Union Road'
- 'The Bonus this year is 23%'

Date Literals

Enclose dates in single quotes and follow immediately with the word date, in brackets. Use the format YYYY-MM-DD"T" HH:MI:SS.FFF"Z", YYYY-MM-DD HH24:MI:SS, or DD-MON-YYYY. It is recommended that you use the first two formats if you want to compile the formula under different language settings.

Examples of date literals are:

- '2010-11-04T00:00:00.000Z' (DATE)
- '1989-03-12 00:00:00' (DATE)
- '12-MAR-1989' (DATE)

Array Literals

An array holds multiple values that can be accessed using the corresponding index values. Arrays literals are defined only for an empty array of each type.
The array types are:

- Array of date values indexed by a numeric index (EMPTY_DATE_NUMBER)
- Array of number values indexed by a numeric index (EMPTY_NUMBER_NUMBER)
- Array of text values indexed by a numeric index (EMPTY_TEXT_NUMBER)
- Array of date values indexed by a text index (EMPTY_DATE_TEXT)
- Array of numeric values indexed by a text index (EMPTY_NUMBER_TEXT)
- Array of text values indexed by a text index (EMPTY_TEXT_TEXT)

Examples of array literals are:

- A1('One') = 1
- SECONDINDEX = A1.NEXT(FIRSTINDEX, '')
- FIRSTINDEX = A1.FIRST('"

**Formula Variable Data Types: How They Are Determined**

The data type of a variable can be numeric, text or date. The data type determines the type of information the variable holds. You do not have to specify the variable type. Formulas determine the type from how you use the variable. For example, if you set a variable to 'J. Smith', this is interpreted as a text variable. The system also warns you if you try to perform any inconsistent operations, such as trying to add a number to a text string.

**Settings That Affect Variable Data Types**

A formula will fail to compile where variables are used inconsistently or incorrectly. Examples of such errors are:

- The formula attempts to alter a database item value.
- A variable is initially determined to be of one data type, but is later used as another data type. For example, the assignment C = 1 shows that C is a number variable. This is inconsistent when later in the formula C = 'Hello' is entered. In this case, C is being used as a text variable.
- The compiler can determine that a variable has not been assigned a value when it is used. Note: If the compiler thinks there might be a problem, it generates code to raise an error if the variable is not initialized at the time of use.
- A variable is determined to be a formula context but then is used as an ordinary local variable.
- A variable is determined to be a local variable, but is then used as a context within a context handling statement.
How Variable Data Types Are Determined

The rules that determine the variable data type in a formula are processed in the following order:

1. The variable can be an input you name in the input statement. For example:

   INPUTS ARE SALARY_AMOUNT,
   START_DATE (DATE),
   FREQUENCY (TEXT)

   If you do not specify the variable data type in the statement, the formula assumes it has data type number.

   The variable data type can be determined from a DEFAULT FOR statement such as:

   DEFAULT FOR A IS EMPTY_NUMBER_NUMBER /* A is a NUMBER_NUMBER array variable. */

   The type can also be determined from a DEFAULT_DATA_VALUE statement:

   DEFAULT_DATA_VALUE FOR B IS 0 /* B is a NUMBER_NUMBER database item. */

   The DEFAULT_DATA_VALUE statement applies to array database items. Array database items have a NUMBER index type and the type of default data value determines the array’s value type.

2. The formula searches the list of database items. If it is in the list, the data type is known.

3. If the variable appears in a context handling statement then the formula searches the list of contexts. If it is in the list, then the formula knows the data type, otherwise an error is raised.

4. If the variable is not a database item or a context, then it is treated as a local variable. The data type is determined by the way in which the variable is used. For example:

   A = 'abc' /* A is a TEXT variable. */

Array Variables: Explained

Formulas have array variables holding date, number, or text values. If the data type cannot be determined then the data type is defaulted to number.

Arrays are similar to PL/SQL index-by tables. Do not specify the array size limit. Arrays are provided for convenience and excessive use and large arrays will result in excessive memory consumption.

The index types are either text or number. Text indexes are upper case unique. Gaps in index value sequences are permitted. Number indexes are truncated.
to remove any fractional part. An array may be iterated in index forwards or backwards. Methods are provided to get the first and last indexes and to get the next or prior index given an index. A method is also provided to test the existence of an index.

Array types are specified as `<DATA_TYPE>_<INDEX_TYPE>` giving: NUMBER_NUMBER, NUMBER_TEXT, DATE_NUMBER, DATE_TEXT, TEXT_NUMBER, and TEXT_TEXT. Arrays can be used for input, output, and local formula variables. Contexts cannot be array types. Formula functions cannot return arrays nor take array parameters. Methods for returning first, last, next, prior indexes take a default value to be used if the required indexes do not exist. These methods return the index data type. An attempt to delete a value at a nonexistent index does not cause an error. An attempt to reference an array value at a nonexistent index causes an error to be raised. The array method syntax does not work directly with the array literal values. For example, it is not possible to use a construct such as EMPTY_DATE_NUMBER.COUNT.

**Array Methods**

Examples of array method syntax:

<table>
<thead>
<tr>
<th>Array Method</th>
<th>Description</th>
<th>Usage Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;name&gt;[&lt;index value&gt;]</code></td>
<td>Get the value for an index.</td>
<td>V = A[1]</td>
</tr>
<tr>
<td><code>&lt;name&gt;.FIRST(&lt;default value&gt;)</code></td>
<td>Get the first index for an array. The default value is returned if the array is empty.</td>
<td>I = A.FIRST(-1)</td>
</tr>
<tr>
<td><code>&lt;name&gt;.LAST(&lt;default value&gt;)</code></td>
<td>Get the last index for an array.</td>
<td>L = B.LAST(' ')</td>
</tr>
<tr>
<td><code>&lt;name&gt;.EXISTS(&lt;index value&gt;)</code></td>
<td>Conditional checking if a value exists at an index. The default value is returned if the array is empty.</td>
<td>IF A.EXISTS(1) THEN</td>
</tr>
<tr>
<td><code>&lt;name&gt;.NEXT(&lt;index value&gt;, &lt;default index value&gt;)</code></td>
<td>Get the next index given an index position. The default value is returned if there is no next index.</td>
<td>N = A.NEXT(1)</td>
</tr>
<tr>
<td><code>&lt;name&gt;.PRIOR(&lt;index value&gt;, &lt;default index value&gt;)</code></td>
<td>Get the prior index given the index position. The default value is returned if there is no prior index.</td>
<td>P = B.PRIOR('Two')</td>
</tr>
<tr>
<td><code>&lt;name&gt;.COUNT</code></td>
<td>Numeric method to count the array elements.</td>
<td>C = A.COUNT</td>
</tr>
<tr>
<td><code>&lt;name.DELETE(&lt;index value&gt;)</code></td>
<td>Delete the element at an index position.</td>
<td>B.DELETE('three')</td>
</tr>
<tr>
<td><code>&lt;name.DELETE()</code></td>
<td>Delete all elements.</td>
<td>B.DELETE()</td>
</tr>
</tbody>
</table>

**Iterating Through an Array**

In the following example, A is an array variable with a NUMBER index. -1234 is known to be an invalid index for A so it is used as a default value when the FIRST and NEXT calls cannot find an index.

```/* -1234 is not a valid index for A in this instance, so use as default. */```
NI = A.FIRST(-1234)
WHILE A.EXISTS(NI) LOOP
  VA = A[NI] /* Do some processing with element at index NI. */
  NI = A.NEXT(NI,-1234) /* Go to next index. */
END LOOP;

The following example does the same thing for array variable B with a TEXT index.

/* 'No Index' is not a valid index for A in this instance, so use as
default. */
TI = B.FIRST('No Index')
WHILE B.EXISTS(TI) LOOP
  VB = B[TI] /* Do some processing with element at index TI. */
  TI = B.NEXT(TI, 'No Index') /* Go to next index. */
END LOOP;

The following example iterates backwards from through an array C with a
NUMBER index.
/* -1234 is not a valid index for C in this instance, so use as default. */
NI = C.LAST(-1234)
WHILE C.EXISTS(NI) LOOP
  VC = C[NI] /* Do some processing with element at index NI. */
  NI = C.PRIOR(NI,-1234) /* Go to prior index. */
END LOOP;

**Formula Contexts: Explained**

A formula executes within an application-specific execution context. Formula
context variables specify the formula execution context.

Examples of contexts are:

- **EFFECTIVE_DATE** for the effective date the formula is executing
- **PAYROLL_ID** for the running payroll
- **PERSON_ID** identifying the person for whom the formula is executing

Context values act as SQL bind values when database item values are fetched
from the database. They can also be passed into formula function calls.

**Context Value Setting**

The application code calling a formula usually sets all the context values. For
some complex applications, such as the payroll run, the code only sets the
contexts necessary to meet general processing requirements. A payroll run sets
contexts for the legislative data group, date earned, the payroll being processed,
the payroll relationship, payroll actions, and the person being processed. Payroll
formulas also use additional contexts whose setting is country-specific. For
example, the jurisdiction area and tax code context values are localization-
specific and are set within formulas using a variety of mechanisms.

**Formula Context-Handling Statements**

If a variable appears in a context handling statement the formula searches the list
of contexts. The variable must appear in the contexts list, otherwise the formula
raises an error. The data type is held with the context list entry.
Formula context handling statements are described below.

- **CHANGE_CONTEXTS(assignment [...])** - Context values may be changed within a formula using a context changing block, but after leaving the context changing block any changed context values are restored to their previous values. Inside the context changing block, formula function calls, database items, and called formulas use the new context values. Context changing blocks may be nested where context changes need to be applied in stages.

  For example:

  ```
  /*
  * Nested Context changes: DBI1 depends upon SOURCE_ID and SOURCE_TEXT. */
  CHANGE_CONTEXTS(SOURCE_TEXT = 'A')
  { /* SOURCE_TEXT = 'A' */
    X = DBI1
    /* Nesting used to change Contexts in stages. */
    CHANGE_CONTEXT(SOURCE_ID = 2)
    { /* SOURCE_TEXT = 'A', SOURCE_ID = 2 */
      Y = DBI1
      /* Overriding a Context change. */
      CHANGE_CONTEXTS(SOURCE_TEXT = 'B', SOURCE_ID = 3)
      { /* SOURCE_TEXT = 'B', SOURCE_ID = 3 */
        Z = DBI1
      }
    }
  }
  ```

- **CONTEXT_IS_SET(context)** - Tests whether or not a context value is set.

  For example, the following code tests whether or not the AREA3 context is set.

  ```
  IF CONTEXT_IS_SET(AREA3) THEN
  ```

- **GET_CONTEXT(context, default value)** - Returns a context’s value if the context is set, otherwise it returns the default value specified in its second argument.

  For example:

  ```
  /* AREA1 is a context of type TEXT. */
  AREA1_VALUE = GET_CONTEXT(AREA1, ' ')
  ```

**Working Storage Area: Explained**

The working storage area is a mechanism for storing global values across formulas. The values are accessed by name. The names are case-independent.

There are four working storage area call methods:

- **WSA_EXISTS**
- **WSA_DELETE**
The working storage area methods are described in the below table.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSA_EXISTS(item [, type])</td>
<td>Test whether or not the item called item exists in the storage area. If type is specified then the item must be of the same type. The valid values for type are one of the strings: DATE, DATE_NUMBER, DATE_TEXT, NUMBER, NUMBER_NUMBER, NUMBER_TEXT, TEXT, TEXT_NUMBER, or TEXT_TEXT.</td>
</tr>
<tr>
<td>WSA_DELETE([item])</td>
<td>Delete the item called item. If a name is not specified then all storage area data is deleted.</td>
</tr>
<tr>
<td>WSA_SET(item, value)</td>
<td>Set the value for the item called item. Any existing item of the same name is overwritten.</td>
</tr>
<tr>
<td>WSA_GET(item, default-value)</td>
<td>Gets a value for the item called item. If there is no item called item then default-value is returned. The data type of default-value is the expected data type for item.</td>
</tr>
</tbody>
</table>

### Calling a Formula from a Formula: Explained

A formula can be called from another formula. This enables some modularity in formula organization. The called formula name, and any formula input or output names are specified as TEXT values. The names are case-independent.

When the formula runs, checks are performed to ensure the called formula can be executed, and whether the specified input and output data types are correct. The IS_EXECUTABLE call determines whether an executable formula with a specified name exists. The formula must be compiled and visible according to data partitioning requirements. Also, it must be valid as of the effective date of calling formula execution. Payroll code imposes extra restrictions based on formula type combinations.

This table describes the methods used when calling a formula from a formula.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXECUTE(formula)</td>
<td>Execute the called formula.</td>
</tr>
<tr>
<td>GET_OUTPUT(output, default-value)</td>
<td>Get the value for the called output after calling a formula. If there is no formula output called output or it is not set then a default value is returned. The data type of default value is the expected data type for output.</td>
</tr>
<tr>
<td>IS_EXECUTABLE(formula)</td>
<td>Test whether the called formula is executable.</td>
</tr>
<tr>
<td>SET_INPUT(input [,value])</td>
<td>If a value is provided, set the value for the called input before calling a formula to value. The data type of the value is the expected data type for input. If the value is not provided then the input is passed as unset to the formula.</td>
</tr>
</tbody>
</table>
Note

Formula inputs set using `SET_INPUT` persist as long as no `EXECUTE` or `GET_OUTPUT` calls are made. Output values from a called formula persist as long as no `SET_INPUT` or new `EXECUTE` calls are made. Any saved input or output values are also removed when the calling formula exits.

There are three stages in calling a formula:

- Set the inputs
- Call the formula
- Get the formula outputs

Set the Inputs

Use a `SET_INPUT` call for each formula input and context whose value needs to be explicitly set for the formula call. It is not necessary to specify all formula inputs and contexts. If a formula input is not specified, it is unset in the called formula. If not specified, context values are inherited from the calling formula. Context values can be explicitly unset in the called formula by using the single argument `SET_INPUT` call.

Call the Formula

Use an `EXECUTE` call to call a formula. Any extra inputs specified in `SET_INPUT` calls are ignored. Errors are raised if the formula is not executable, if the called formula is already executing, or if an input variable's data type, as specified using `SET_INPUT`, does not match its actual data type within the formula.

Get the Formula Outputs

Use one or more `GET_OUTPUT` calls to fetch outputs from the last formula call. An error is raised if an output variable's data type, as specified using `GET_OUTPUT`, does not match its actual data type within the formula. A `GET_OUTPUT` call has a default value that is returned when the specified output does not exist, or was not set by the called formula.
Formula Functions: Explained

Functions manipulate data in different ways and always return a value. They are restricted to simple data types of date, number, and text. Some functions work on only one type of data, some can work on two, others work on more than one data types.

A function is specified by its name, return data type, parameter data types, and parameter usage behavior. The general form of a function is:

```
NAME-OF-FUNCTION(operand,operand,...)
```

Parameters can be optional or mandatory. They may be repeated any number of times, such as with the `GREATEST` function. The formula compiler resolves functions by matching function calls against function specifications. You can use multiple functions with the same name within a formula provided that they have different return or parameter data types.

Functions can work one or multiple data types:

- Text
- Number
- Date
- Data Conversion
- Miscellaneous

---

Note

There are additional formula functions for specific applications, such as payroll, benefits, or compensation.

---

Text Formula Functions

The following formula functions manipulate text data.
**CHR(n)**

Returns the character having the binary equivalent to a number operand n in the ASCII character set.

**GREATEST(expr, expr [,expr]...)**

Compares the values of all the text string operands. It returns the value of the last string in alphabetic order.

**INITCAP(expr)**

Returns the expression expr with the first letter of each word in uppercase, all other letters in lowercase. Words are delimited by white space or characters that are not alphanumeric.

**INSTR(expr1, expr2 [,n [,m]])**

Searches expr1 beginning with its nth character for the mth occurrence of expr2 and returns the character position in expr1 for the first character of this occurrence. If n is negative, INSTR counts and searches backward from the end of expr1. The value of m must be positive. The default values of both n and m are 1, meaning INSTR begins searching at the first character of expr1 for the first occurrence of expr2. The return value is relative to the beginning of expr1, regardless of the value of n, and is expressed in characters. If the search is unsuccessful (expr1 does not appear m times after the nth character of expr1) the return value is 0.

**INSTRB(expr1, expr2 [,n [,m]])**

The same as INSTR, except that n and the return value are expressed in bytes, rather than in characters. For a single-byte character set, INSTRB is equivalent to INSTR.

**LEAST(expr, expr [,expr]...)**

Compares the values of all the text string operands. Returns the first string in alphabetic order from among its operands.

**LENGTH(expr)**

Returns the number of characters in the text string operand expr.

**LENGTHB(expr)**

Returns the length of expr in units of bytes.

**LOWER(expr)**

Converts a text string to lower case.

**LPAD(expr, n [,pad])**

Returns the text string operand expr left-padded to length n with the sequence of characters in pad. The default for pad is a blank. If expr is longer than n, then LPAD returns the portion of expr that fits in n.
Examples:

/* A is set to 'XYXYXhello' */
A = LPAD ('hello', 10, 'XY')
/* A is set to 'hell' */
A = LPAD ('hello', 4 )

**LTRIM(expr [,set])**

Returns the text string operand expr with all the left-most characters that appear in set removed. The default for set is a blank. If none of the left-most characters of expr appear in set then expr is returned.

Examples:

/* A is set to 'def' */
A = LTRIM ('abcdef','abc')
/* A is set to 'abcdef' */
A = LTRIM ('abcdef','bc')

**REPLACE(expr, search [,replacement])**

Returns the text string operand expr with every occurrence of search replaced with replacement. If replacement is omitted, all occurrences of search are removed. Use REPLACE to substitute one string for another as well as to remove character strings.

Example:

/* Set A to 'BLACK and BLUE'. */
A = REPLACE('JACK and JUE', 'J', BL')

**RPAD(expr, n [,pad])**

Returns the text string operand expr right-padded to length n with the sequence of characters in pad. The default for pad is a blank. If expr is longer than n, then RPAD returns the portion of expr that fits in n.

Examples:

/* A is set to 'helloXYXY' */
A = RPAD ('hello', 10, 'XY')
/* A is set to 'hell' */
A = RPAD ('hello', 4 )

**RTRIM(expr [,set])**

Returns the text string operand expr with all the right-most characters that appear in set removed. The default for set is a blank. If none of the right-most characters of expr appear in set then expr is returned.

Examples:

/* A is set to 'abc' */
A = RTRIM ('abcdef','def')
/* A is set to 'abcdef' */
A = RTRIM ('abcdef','bc')

**SUBSTR(expr, m [,n]) or SUBSTRING(expr, m [,n])**

SUBSTRING returns a substring of the text string operand expr of length n characters beginning at the mth character. If n is negative, SUBSTR counts
backwards from the end of expr. If you omit the n, the substring starts from m and finishes at the end of expr.

Example:
/* Check that the tax code starts with GG */
IF length(Tax_code) <= 2
THEN
  (message = 'Tax code is too short'
RETURN message
)
IF substr( Tax_code, 1, 2) = 'GG' THEN ...

SUBSTRB((expr, m [,n])
The same as SUBSTR, except that the arguments m and n are expressed in bytes, rather than in characters. For a single-byte database character set, SUBSTRB is equivalent to SUBSTR.

TRANSLATE(expr, from, to)
Returns the text string operand expr with all occurrences of each character in from replaced by its corresponding character in to. Characters in expr that are not in from are not replaced. The argument from can contain more characters than to. In this case, the extra characters at the end of from have no corresponding characters in to. If these extra characters appear in expr, they are removed from the return value.

TRIM(expr)
Trims leading and trailing spaces from a character string.

UPPER(expr)
Converts a text string to upper case.

Numeric Formula Functions
The following formula functions manipulate numeric data.

ABS(n)
Returns the magnitude of a numeric operand n as a positive numeric value. If the value of the operand is positive, its value returns unchanged. If the operand is negative then the value's sign inverts, and the value returns as a positive number.

Example:
ABS (-17)
It returns 17.

FLOOR(n)
Returns the integer part of a numeric operand n. If the value of the operand contains information after the decimal point, FLOOR discards that information and returns a whole number.
Example:

\[ \text{FLOOR}(35.455) \]

It returns 35.

\[ \text{GREATEST}(n, n [, n] ...) \text{ or } \text{GREATEST_OF}(n, n [, n] ...) \]

Compares all the operands and returns the largest value.

\[ \text{LEAST}(n, n [, n] ...) \text{ or } \text{LEAST_OF}(n, n [, n] ...) \]

Compares all the operands and returns the smallest value.

\[ \text{MOD}(m, n) \]

Returns the remainder from dividing \( m \) by \( n \).

\[ \text{POWER}(m, n) \]

Returns \( m \) raised to the \( n \)th power.

\[ \text{ROUND}(m [,n]) \]

Rounds \( m \) to \( n \) decimal places. The default number of decimal places is 0.

Examples:

\[ \text{ROUND}(2.3401, 2) \]

It returns 2.34.

\[ \text{ROUND} (2.3461, 2) \]

It returns 2.35.

\[ \text{ROUNDUP}(m [,n]) \text{ or } \text{ROUND_UP}(m [,n]) \]

Rounds \( m \) up to \( n \) decimal places. The default number of places is 0.

Examples:

\[ \text{ROUND_UP}(2.3401, 2) \]

It returns 2.35.

\[ \text{ROUND_UP}(2.3400, 2) \]

It returns 2.34.

\[ \text{TRUNC}(n [,m]) \text{ or } \text{TRUNCATE}(n [,m]) \]

Truncates \( m \) down to \( n \) decimal places. The default number of places is 0.

Examples:

\[ \text{TRUNC}(2.3401, 2) \]

It returns 2.34.
Date Formula Functions

The following formula functions manipulate date data.

**ADD_DAYS(date, n)**

Adds $n$ whole days to `date`.

Example:

```
ADD_DAYS('30-DEC-1990' (date), 6)
```

It returns `5 JAN 1991`.

**ADD_MONTHS(date, n)**

 Adds $n$ whole months to `date`.

**ADD_YEARS(date, n)**

Adds $n$ whole years to `date`.

**DAYS_BETWEEN(date1, date2)**

Returns the number of days between `date1` and `date2`. If `date1` is later than `date2` then the result is a positive number. If `date1` is earlier than `date2` then the result is a negative number.

Example:

```
DAYS_BETWEEN('1995/06/27 00:00:00' (date), '1995/07/03 00:00:00' (date))
```

It returns `-5`.

**GREATEST(date, date [, date] ...)**

Compares its operands and returns the latest date.

**LAST_DAY(date)**

Returns the last day of the month containing `date`.

**LEAST(date, date [, date] ...)**

Compares the operands and returns the earliest `date`.

**MONTHS_BETWEEN(date1, date2)**

Returns the number of months between `date1` and `date2`. If `date1` is later than `date2`, the result is a positive number. If `date1` is earlier than `date2`, the result is a negative number. The return value has a numeric data type that can contain a fraction if the dates do not differ by a whole number of months.

**NEW_TIME(date, zone1, zone2)**

Returns the date and time in zone `zone2` when the date and time in zone `zone1` are `date`. 
The arguments `zone1` and `zone2` can be any one of the standard text strings such as:

<table>
<thead>
<tr>
<th>Time Zone</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST or ADT</td>
<td>Atlantic Standard or Daylight Time</td>
</tr>
<tr>
<td>BST or BDT</td>
<td>Bering Standard or Daylight Time</td>
</tr>
<tr>
<td>CST or CDT</td>
<td>Central Standard or Daylight Time</td>
</tr>
<tr>
<td>EST or EDT</td>
<td>Eastern Standard or Daylight Time</td>
</tr>
<tr>
<td>GMT</td>
<td>Greenwich Mean Time</td>
</tr>
<tr>
<td>HST or HDT</td>
<td>Alaska-Hawaii Standard Time or Daylight Time</td>
</tr>
<tr>
<td>MST or MDT</td>
<td>Mountain Standard or Daylight Time</td>
</tr>
<tr>
<td>NST</td>
<td>Newfoundland Standard Time</td>
</tr>
<tr>
<td>PST or PDT</td>
<td>Pacific Standard or Daylight Time</td>
</tr>
<tr>
<td>YST or YDT</td>
<td>Yukon Standard or Daylight Time</td>
</tr>
</tbody>
</table>

**NEXT_DAY(d, expr)**

Returns the first date following `d` of the weekday named by `expr`.

**ROUND(date [,format])**

Returns the result of rounding `date` according to `format`. The default format is `DDD`, which represents the nearest day.

**TRUNC(date [,format])**

Returns the result of truncating `date` according to `format`. The default format is `DDD`, which represents a whole day.

## Data Conversion Formula Functions

The following formula functions perform data conversions.

**DATE_TO_TEXT(date [,format]), TO_CHAR(date [,format]), and TO_TEXT(date [,format])**

Converts `date` to a character string with format specified by `format`. The default format is the application canonical format.

**NUM_TO_CHAR(n, format)**

Converts the number `n` to a character string in the specified format. This function is equivalent to the SQL TO_CHAR function.

**TO_CHAR(n) and TO_TEXT(n)**

Converts the number `n` to a character string in canonical number format.

**TO_DATE (expr [, format])**

Converts the character string `expr` in the specified format to a date. If no format is specified then `expr` must be in canonical format.
**TO_NUMBER(expr) and TO_NUM(expr)**

Converts the character string `expr` to a number. The character string must be in canonical number format. A period is used for the decimal point, such as 1.234. Negative numbers are preceded with a minus, such as -1.234.

### Miscellaneous Formula Functions

The following formula functions manipulate messaging data.

- **GET_MESG(appname, msgname [, token1, value1] [, token2, value2] [, token3, value3] [, token4, value4] [, token5, value5] ) and

  Returns an expanded version of the application message specified using `appname`, `msgname` and up to five pairs of message tokens and their corresponding values.

- **HR_TRACE(expr)**

  Outputs a trace message.

---

**Note**

It is more efficient to use an application-specific logging function than `HR_TRACE`.  

---
Proration Formulas: Explained

When the payroll run encounters an event (such as a grade change) that you have defined as a proration event for the element being processed, it creates two run results for the element, one for the payroll period up to the day before the event, and one from the date of the event to the end of the period. You must define a formula to handle this proration processing for the element. There are two ways to create the formula:

- Edit the formula of type Oracle Payroll for the element so that it can handle proration

- Create an additional formula to run after the Oracle Payroll formula only in periods when a proration event is encountered. You select this formula in the Proration Formula field on the Proration section of the Manage Elements page.

Using a separate proration formula has the advantage that proration takes place even when you enter a pay value directly on the element entry. Embedding the proration calculation in the Oracle Payroll formula avoids the processing time of calling the second formula in periods when proration events occur.

If you want to write a proration formula, you must follow these rules:

- Select the formula type Payroll Run Proration.
- Add the formula inputs:
  - Any of the element input values
  - Prorate_start (DATE)
  - Prorate_end (DATE)
- Add the formula outputs of any of the element input values.

There may be predefined example formulas for your localization that you can use.
Accrual Plan Rules: Points to Consider

You can specify the following accrual plan rules on the Accrual Plan page in accordance with the leave policy of your enterprise:

- Accrual start rule
- Accrual term and frequency
- Ineligibility period
- Gross accrual maintenance
- Accrual bands
- Net accrual calculation rules

Note

If you do not specify values for any of the above plan rules, the accrual plan uses the default values in the associated accrual formula.

Accrual Start Rule

You use an accrual start rule to determine when newly enrolled employees start to accrue time. For example, some accrual plans allow new hires to accrue time from the date of their hire. If the predefined start rules that are available on the Accrual Plan page do not meet your requirements, you can add your own rule directly in the accrual formula.

Accrual Term and Frequency

You can specify the type of the accrual term and its length during which employees accrue time. For example, you can define an accrual term of one calendar year that restarts on January 1, or on the employee’s annual hire date. You define the frequency at which employees accrue time during an accrual period, for example, two days every pay period, or two days every month.

Ineligibility Period

You can define a period, such as six months from the date of hire in which newly hired employees accrue time, but not use it until the end of the period. Although you can define the ineligibility period in the Accrual Plan page, if you have more complex rules to incorporate, you can include those rules in the accrual formula.
Gross Accrual Maintenance

Oracle Fusion Global Payroll users can choose to store gross accruals in a payroll balance. The advantage is that the gross accruals are calculated since the last payroll run, and not for the entire accrual term, thus reducing the number of calculations.

Accrual Bands

Use accrual bands to define accrual benefits for employees. You can create as many bands as you require.

Net Entitlement Rules

The accrual plan generates the net accrual calculation for enrolled employees. By default, the absence types that you associate with the accrual plan appear as deductions in the calculation. However, you can include other elements to customize the calculation.

Accrual Start Date for New Hires: How It Is Calculated

By default, the payroll accrual formulas (Accrual Payroll Calculation formula and Accrual Payroll Balance Calculation formula) start the accrual from January 1, and the simple accrual formulas (Accrual Simple Multiplier and Accrual Simple Balance Multiplier), from June 1.

Settings That Affect Accrual Start Date

The formula uses the start rule you specified for the accrual plan in the Accrual Plan page.

How the Start Rules Are Interpreted

On the basis of the start rule, the formula calculates the start date from the employee's hire date and compares it with the plan enrollment date. Accrual begins on whichever of these two dates is later.

The following table describes how the formula interprets each start rule.

<table>
<thead>
<tr>
<th>Start Rule</th>
<th>How the Formula Interprets It</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hire Date</td>
<td>Accruals begin from the first full pay period following the hire date. If the hire date is on the first day of the pay period, the participant starts to accrue time as of that date. For example, if the hire date of a participant on a monthly payroll falls on January 10, 2011, and the pay period starts on the first of every month, accruals start on February 1, 2011. If the hire date falls on January 1, 2011, then the participant starts to accrue time on the same day.</td>
</tr>
<tr>
<td>Beginning of calendar year</td>
<td>Accruals begin from the start of the year following the year of hire. For example, a participant with a hire date of January 1, 2011 starts to accrue time from January 1, 2012.</td>
</tr>
</tbody>
</table>
### Changing Start Rules

There are two ways to change the start rules:

- You can enter additional start rules as values for the ACCRUAL_START_TYPE lookup type, which provides a list of values for the Accrual Start field. You must add a statement to your copy of the predefined formula to calculate the accrual start date using your new start rule.

- You can ignore the Accrual Start field and calculate the start date entirely within the formula. To do this, copy the predefined formula and replace the section that calculates the accrual start date with your own formula statements.

### Accrual Formulas: Critical Choices

The accrual formula calculates the gross accrual on the basis of time that employees accrue in each accrual period. The type of accrual formula that you want to associate with your accrual plan depends on how you want to implement the following plan rules:

- Type of accrual period
- Gross accrual maintenance

Use the following table to decide which formula to select for your accrual plan. For example, if you want your employees to accrue time per payroll period and you want to maintain gross accruals using a payroll balance, you must base your formula on the Accrual Payroll Balance Calculation formula.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accruals per payroll period</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Accruals begin from the first full pay period after the six-month anniversary of the hire date. If the six-month anniversary falls on the first day of the pay period, the participant starts to accrue time as of that date.

For example, a participant on a semi-monthly payroll who is hired on 9 February, 2011 completes six months service on 9 August, 2011. The participant starts to accrue time on the first day of the second pay period in August. If the hire date was on 1 February, 2011, the participant starts to accrue time on the first day of the first pay period in August.

The period of ineligibility does not apply to accrual plans using this start rule.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of accrual term</td>
<td>One year.</td>
<td>One year.</td>
<td>Yes</td>
</tr>
<tr>
<td>Accrual term start date</td>
<td>January 1.</td>
<td>June 1.</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Accrual calculations restart at the beginning of each calendar year.</td>
<td>Accrual calculations restart at the beginning of each June.</td>
<td></td>
</tr>
<tr>
<td>Accrual frequency</td>
<td>On the basis of the enrolled employee’s pay periods.</td>
<td>Monthly.</td>
<td>Yes, if you select Simple as the accrual frequency type.</td>
</tr>
<tr>
<td></td>
<td>For example, employees on a monthly payroll accrue time on the last day of their pay period, independently of payroll runs.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accrual amount</td>
<td>2 days per pay period.</td>
<td>2 days per month.</td>
<td>Yes</td>
</tr>
<tr>
<td>Ceiling</td>
<td>20 days.</td>
<td>20 days.</td>
<td>Yes</td>
</tr>
<tr>
<td>Maximum carryover</td>
<td>20 days.</td>
<td>20 days.</td>
<td>Yes</td>
</tr>
<tr>
<td>Length of service calculation (for accrual bands and ineligibility period)</td>
<td>Uses continuous service date (if present) or hire date.</td>
<td>Uses continuous service date (if present) or hire date.</td>
<td>No, but you can enter the continuous service date in the accrual plan element using the Manage Element Entries page.</td>
</tr>
<tr>
<td>Accrual start date for new hires</td>
<td>Based on start rules that you can select when you create the accrual plan.</td>
<td>Based on start rules that you can select when you create the accrual plan.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Note**

The Accrual Payroll Calculation formula and the Accrual Simple Multiplier formula incorporate the same rules as the accrual formulas that use balances, except that they cannot be called from the payroll run to maintain a balance of gross accruals.

**Accrual Formulas That Support Payroll Balances**

Use the following table to compare the differences in the default calculation methods of the Accrual Payroll Balance Calculation formula and the Accrual Simple Balance Multiplier formula. You can also use the table to check if you can change a particular plan rule on the Accrual Plan page.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Example</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period of ineligibility</td>
<td>Based on the period that you can define in the Accrual Plan page. Accrued time is not credited to the employee until the end of the ineligibility period.</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Effective dates of carried-over time</td>
<td>Sets the effective start date to December 31 of the accrual term year that was processed by the Calculate Carry-Over process. Carried-over time expires a year later. For example, the carried-over time with the effective start date December 31, 2010 expires on December 31, 2011.</td>
<td>Sets the effective start date to May 31 of the accrual term year that was processed by the Calculate Carry-Over process. Carried-over time expires a year later. For example, the carried-over time with the effective start date May 31, 2010 expires on May 31, 2011.</td>
<td>Yes. The Calculate Carry-Over process calculates the effective dates according to the carryover expiry duration that you specify.</td>
</tr>
<tr>
<td>Calculation of gross accruals</td>
<td>Sums accruals in all full pay periods that end on or before the calculation date in the current accrual term. The formula also considers in its calculation any employee termination date, and changes in assignment status.</td>
<td>Sums accruals in all full months that end on or before the calculation date in the current accrual term. The formula also considers in its calculation the employee termination date (if present).</td>
<td>No</td>
</tr>
<tr>
<td>Calculation of gross accruals for suspended assignments</td>
<td>Calculates the number of active working days of the assignment in the payroll period, multiplies the normal accrual rate by the number of active days, and divides it by the number of total working days, to prorate the accrual.</td>
<td>Does not process changes in assignment status</td>
<td>No</td>
</tr>
</tbody>
</table>

**Custom Accrual Formulas: Points to Consider**

If you decide to customize the predefined accrual plan formulas, you can incorporate your own plan rules, using functions and database items to access extra inputs. However, there are constraints on what you can change in the formulas.

**Using Predefined Formulas**

You must take a copy of any predefined formula that you want to use and associate the copy with your accrual plan. This approach enables you to refer
to the predefined formula if the changes you make to your own formula do not work to your expectations.

**Using Extra Inputs**

Use the predefined accrual formula functions and database items to access extra inputs for calculations in your custom accrual plan formulas. You can define and register any additional functions you require to incorporate your plan rules. However, your formula must use the same input statements and return statements that exist in the predefined formula.

**Incorporating Accrual Calculations**

The predefined accrual formulas contain calculations to ensure that the employee is entitled to accrue time. Although you can include your own calculations in the custom formula, your formula must include the following calculations:

<table>
<thead>
<tr>
<th>Calculation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Termination date</td>
<td>Check whether there is a termination date for the assignment. If the termination date is before the calculation date, calculate the accrual as of the termination date. If your formula does not handle partial accrual periods, check whether the termination date is before the end of the first accrual period; if yes, set gross accrual to zero.</td>
</tr>
<tr>
<td>Enrollment end date</td>
<td>Check whether an end date exists for the assignment's enrollment in the plan. If the end date is before the calculation date, calculate the accrual as of the end date. If your formula does not handle partial accrual periods, check whether the enrollment end date is before the end of the first accrual period; if yes, set gross accrual to zero.</td>
</tr>
<tr>
<td>Calculation date</td>
<td>Check whether the calculation date is before the end of the first accrual period; if yes, set gross accrual to zero (unless your formula handles partial accrual periods).</td>
</tr>
<tr>
<td>Hire date</td>
<td>Check the employee's hire date or continuous service date. If your formula handles partial accrual periods, check that this date is before the calculation date, and if not, set the gross accrual to zero. If your formula does not handle partial periods, check that this date is before the start of the last full accrual period used in the current calculation. If the employee has not worked for a full accrual period before the calculation date, set the gross accrual to zero.</td>
</tr>
<tr>
<td>Start date for newly enrolled employees</td>
<td>Check when the employee must start to accrue time. This is typically the date of enrollment in the plan or, if your formula does not handle partial accrual periods, the first period starting on or after the date of enrollment in the plan. If this date (or period) is after the calculation date (or period), set the gross accrual to zero.</td>
</tr>
<tr>
<td>Ineligibility period</td>
<td>Check if an ineligibility period exists. Set the gross accrual to zero if the ineligibility period is still in force on either of these dates:</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>• Calculation date</td>
</tr>
<tr>
<td></td>
<td>• End date of the last accrual period used in the calculation (if your formula does not handle partial accrual periods)</td>
</tr>
</tbody>
</table>

| Inactive assignments | Check whether the employee’s assignment is active throughout the period for which you want to calculate accruals. Depending on your plan rules, your employees might not accrue time when their assignments are inactive, or they might accrue time at a reduced rate during this period. You can use the GET_ASG_INACTIVE_DAYS function to check the assignment status on each day from the start date to the end date of the period and return the number of inactive working days. |

**Turning Accrual Plan Rules Into Formula Statements: Examples**

Although you can incorporate accrual plan rules from the Manage Accrual Plans page, you can incorporate more complex rules in the accrual formula directly. Use these examples to incorporate your plan rules into the Accrual Simple Multiplier formula.

---

**Caution**

Before modifying a predefined formula, you must copy it first and then modify the copy.

**Defining Additional Start Rules for New Hires**

If you want to use an accrual start rule other than the predefined ones, you must first define the new rule as a value for the PER_ACCRUAL_START_RULE lookup type. Then modify the formula section that determines the accrual start rule on the basis of the following sample statements:

```sql
ELSE IF (ACP_START_RULE = <your new lookup value>) THEN
{
    First_Eligible_To_Accrue_Date = <your new calculation to determine the start date>
}
```

**Defining a Different Rule to Determine Accrual Rate**

To determine the accrual rate, the predefined formula by default considers the amount of time that an employee must accrue in the accrual term and the number of accrual periods, both of which you can define on the Accrual Plan page. You can override the default calculation. In the following example, part-time employees accrue 3 hours for every 75 hours worked:

```
Accrual_Rate = ACP_HOURS_WORKED / 75
```
In the above statement, ACP_HOURS_WORKED is a database item that you must add to the accrual plan element input value that contains the number of hours worked by the enrolled employee.

**Defining Accrual Amounts in Advance**

The accrual amount that you define on the **Accrual Plan** page enables enrolled employees to accrue time each period. If, for example, you want employees to accrue their full entitlement of 20 days at the start of every calendar year, use the following basic formula:

INPUTS ARE
- Calculation_Date (date)

Accrued_amt = 20

Effective_start_date = to_date('0101'||to_char(calculation_date, 'YYYY'),'DDMMYYYY')
Effective_end_date = to_date('3112'||to_char(calculation_date, 'YYYY'),'DDMMYYYY')
Accrual_end_date = to_date('0101'||to_char(calculation_date, 'YYYY'),'DDMMYYYY')

RETURN Accrued_amt,
     Effective_start_date,
     Effective_end_date,
     Accrual_end_date

**Note**

The above formula does not contain ineligibility rules or start rules, and does not calculate the accrual for part years (for example, for employees joining the plan midway through a year).

**Using a Different Database Item for Continuous Service Date**

The predefined formula uses the continuous service date (if it was entered for the enrolled employee) to determine when a newly hired employee begins to accrue time. The formula uses the ACP_CONTINUOUS_SERVICE_DATE database item for this purpose. If you are using a different database item for the continuous service date, then replace the database item used in the following formula lines with the new database item:

IF (ACP_CONTINUOUS_SERVICE_DATE WAS DEFAULTED) THEN
  (Continuous_Service_Date = ACP_HIRE_DATE)
ELSE IF(ACP_CONTINUOUS_SERVICE_DATE > Calculation_Period_SD) THEN
  (Total_Accrued_Amt = 0)
  Continuous_Service_Date = ACP_CONTINUOUS_SERVICE_DATE
ELSE
  (Continuous_Service_Date = ACP_CONTINUOUS_SERVICE_DATE)
.
.
IF Continuous_Service_date = ACP_CONTINUOUS_SERVICE_DATE THEN
  (Actual_Accrual_Start_Date = Continuous_service_Date)
Adding Rules for Suspended Assignments

If you want employees to accrue no time (or accrue time differently) while on certain types of leave, such as maternity leave or study leave, use the GET_ASG_INACTIVE_DAYS function to check the status of the assignment, and include the appropriate rules. In the following example, employees do not accrue time while their assignment is inactive:

\[
\text{Assignment\_Inactive\_Days} = \text{GET\_ASG\_INACTIVE\_DAYS}(\text{Period\_SD}, \text{Period\_ED}) \\
\text{IF Assignment\_Inactive\_Days} <> 0 \text{ THEN} \\
\hspace{1cm} \{ \\
\hspace{2cm} \text{Working\_Days} = \text{GET\_WORKING\_DAYS}(\text{period\_SD}, \text{period\_ED}) \\
\hspace{2cm} \text{IF Working\_Days} = \text{Assignment\_Inactive\_Days} \text{ THEN} \\
\hspace{3cm} \{ \\
\hspace{4cm} \text{Multiplier} = 0 \\
\hspace{3cm} \} \\
\hspace{2cm} \} \\
\text{ELSE} \\
\hspace{1cm} \{ \\
\hspace{2cm} \text{Multiplier} = 1 - (\text{Assignment\_Inactive\_Days} / \text{Working\_Days}) \\
\hspace{1cm} \} \\
\]

Accrual Formula Type

If you plan to write your own accrual formula for use with accrual plans, you must ensure that the formula uses the same input and return statements that exist in the predefined formula.

You can use functions and database items to access additional inputs. Contexts are available to the formulas automatically.

**Contexts**

Without using inputs or database items, you can directly access the following values for use with functions in formulas of type Accrual:

- PAYROLL_ASSIGNMENT_ID
- ACCRUAL_PLAN_ID
- LEGISLATIVE_DATA_GROUP_ID
- PAYROLL_ID

**Input Variables**

The following table lists the input variables that you must use in your accrual formula.

<table>
<thead>
<tr>
<th>Input Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculation_Date</td>
<td>Date through which you want to calculate the gross accrual</td>
</tr>
<tr>
<td>Accrual_Start_Date (only if you are using a payroll balance)</td>
<td>Date when accrual calculations must start. If this value is null, accrual calculations start from the beginning of the accrual term.</td>
</tr>
<tr>
<td>Accrual_Latest_Balance (only if you are using a payroll balance)</td>
<td>Latest accrual balance for the accrual term up to the day before the start of the accrual calculations (Accrual_Start_Date). A payroll balance stores the latest balance.</td>
</tr>
</tbody>
</table>
**Return Values**

The following table lists the return values that you must use in your accrual formula.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enrollment_Start_Date</td>
<td>Date when the employee has enrolled in the accrual plan</td>
</tr>
<tr>
<td>Enrollment_End_Date</td>
<td>Date when the accrual plan enrollment expires</td>
</tr>
<tr>
<td>Continuous_Service_Date</td>
<td>The service date that was entered using an input value on the accrual plan element</td>
</tr>
<tr>
<td>Accrual_Start_Date</td>
<td>Start date of the accrual term. Depending on plan rules, this date may be the plan enrollment date, hire date, adjusted service date, or other date.</td>
</tr>
<tr>
<td>Accrual_End_Date</td>
<td>Date through which you want to calculate the gross accrued time. However, if the employee was terminated, this return value contains the termination date. If the employee has left the plan, this value contains the end date of the plan's element entry.</td>
</tr>
<tr>
<td>Accrual_Rate</td>
<td>Amount of time that the employee accrues per year. The value that is returned depends on accrual bands.</td>
</tr>
<tr>
<td>Ceiling_Amt</td>
<td>The maximum time that the employee can accrue. The value that is returned depends on accrual bands.</td>
</tr>
<tr>
<td>Max_Carryover</td>
<td>Maximum time that the employee can carry over to the next accrual term. The value that is returned depends on accrual bands.</td>
</tr>
<tr>
<td>Accrued_Amt</td>
<td>Gross accrued time for the current accrual term</td>
</tr>
<tr>
<td>Entitled_Amt</td>
<td>Net accrual for the current accrual term</td>
</tr>
<tr>
<td>Pay_Rate_Per_Unit</td>
<td>The monetary value of one unit of time that the employee has accrued</td>
</tr>
<tr>
<td>Currency_Code</td>
<td>The currency code used to express the monetary value of one unit of accrued time.</td>
</tr>
</tbody>
</table>

**Accrual Carryover Formula Type**

If you plan to write your own carryover formula for use with accrual plans, you must ensure that the formula uses the same input and return statements that exist in the predefined formula.

You can use functions and database items to access additional inputs. Contexts are available to the formulas automatically.

**Input Variables**

The following table lists the input variables that you must use in your carryover formula.

<table>
<thead>
<tr>
<th>Input Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calculation_date</td>
<td>Any date within an accrual term</td>
</tr>
</tbody>
</table>
accrual_term is a method to calculate the time to carry over. Specify PREVIOUS as the value for this input if you want the formula to calculate carryover for the previous accrual term (before the calculation date). Specify CURRENT if you want the formula to calculate carryover for the accrual term that the calculation date spans.

**Return Values**

The following table lists the return values that you must use in your carryover formula.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_carryover</td>
<td>Amount of time the employee can carry over to the next accrual term.</td>
</tr>
<tr>
<td>effective_date</td>
<td>Last day of the accrual term for which the carryover is calculated.</td>
</tr>
<tr>
<td>expiry_date (optional)</td>
<td>Date when the carryover expires if employees do not use it.</td>
</tr>
</tbody>
</table>

**Accrual Ineligibility Formula Type**

If you plan to write your own ineligibility formula for use with accrual plans, you must ensure that the formulas use the same input and return statements that exist in the predefined formula.

**Input Variables**

You must specify as input the calculation date that indicates the effective date of the enrolled employee's accrual plan element entry.

**Return Values**

You must specify the assignment_eligible return value. The value will be set to Y (eligible) or N (ineligible).

**Accrual Plan Formula Functions**

You can use the predefined accrual formula functions for calculations in your custom accrual plan formulas.

You can create and register any additional functions you require to incorporate your plan rules.

**CALCULATE_PAYROLL_PERIODS ( )**

The function uses the payroll id context. It calculates the number of payroll periods in one year for that payroll, and sets the global variable PAYROLL_YEAR_NUMBER_OF_PERIODS to that value.

For example, the function sets the global variable to 12 for a calendar month payroll.
**GET_ABSENCE (calculation date, start date)**

Returns the total number of whole absences (having start and end dates) that an accrual plan covers between a date interval.

For example, the following formula statement returns the total number of absences that were taken between January 1, 2010 and 31 December, 2010.

\[
\text{TOTAL_ABSENCE} = \text{GET_ABSENCE}('01-JAN-2010'(date), '31-DEC-2010'(date))
\]

**Parameters**

**Start Date**

Date when you want to start searching for absences.

**End Date**

Date up to which you want to search for absences.

**GET_CARRY_OVER (calculation date, start date)**

Returns the number of days or hours recorded on the carryover element entry with an effective date on or between the two input dates. If more than one element entry is effective between these dates, the function sums the hours or days.

Carryover element entries may also have an expiry date, after which any unused carryover time is lost. If the calculation date is after the expiry date, the function calculates the absences between the start date and the calculation date. If the total absence duration equals or exceeds the carryover, the function returns the total carryover because all of the time was used before it expired. If the total duration is less than the total carryover, the function returns the total duration of the absences taken before the carried over time expired. The rest of the carried over time is lost.

For example, if an employee carried over ten days, and takes six days of leave up to the expiry date, the function returns six. The employee loses the four days of carried over time remaining after the expiry date.

**GET_NET_ACCRUAL (calculation date, plan id, accrual start date, accrual latest balance)**

Returns the net accrual at the calculation date.

**Parameters**

**Calculation Date**

Date through which you want to calculate the net accrual.

**Plan ID**

Identifies the accrual plan to use to calculate the net accrual.

**Accrual Start Date**

Start date of the accrual term.

**Accrual Latest Balance**

Latest gross accrual stored in a payroll balance.
**RESET_ACCRUALS()**

Returns a value (Y or N) that indicates whether the PTO_RESET_ACCRUALS pay action parameter was set.

**GET_OTHER_NET_CONTRIBUTION (calculation date, start date)**

Returns between two dates the total amount of time stored in elements that you added to the net calculation on the Accrual Plan page. This calculation does not consider absence entries stored in the absence element or carried-over time stored in the carryover element.

**GET_PAYROLL_PERIOD (date)**

Determines the payroll period that spans the specified date and assigns values to the following global variables:

- PAYROLL_PERIOD_START_DATE
- PAYROLL_PERIOD_END_DATE
- PAYROLL_PERIOD_NUMBER

Use the GET_DATE and GET_NUMBER functions to retrieve the values from the global variables.

**GET_ACCRUAL_BAND (years_of_service)**

Determines the appropriate accrual band for the specified length of service of the employee. The function assigns values to the following global variables:

- ANNUAL_RATE: Amount of time to accrue for the current accrual term at this band
- UPPER_LIMIT: Length of service that the employee must complete for the next band to apply.
- CEILING

The following statements show how to use this function then use the GET_NUMBER function to retrieve the values it sets in the global variables:

```lua
IF (GET_ACCRUAL_BAND(Years_Service) = 0 THEN
    Annual_Rate = GET_NUMBER('ANNUAL_RATE')
    Upper_Limit = GET_NUMBER('UPPER_LIMIT')
    Ceiling = GET_NUMBER('CEILING')
ELSE
    ( <your statements to include error processing> )
)
```

**GET_ASSIGNMENT_STATUS (date)**

Determines the status of an assignment at a given date. The function assigns values to the following global variables:

- ASSIGNMENT_EFFECTIVE_SD (effective start date of the assignment)
• ASSIGNMENT_EFFECTIVE_ED (effective end date of the assignment)
• ASSIGNMENT_SYSTEM_STATUS (status of the assignment)

For example, the following statement returns the status of the assignment on 01 January, 2011:

`ERROR = GET_ASSIGNMENT_STATUS ('01-JAN-2011' (date))`

**GET_ASG_INACTIVE_DAYS (period start date, period end date)**

Returns the number of working days between the input dates when the assignment status was inactive.

**GET_EARLIEST_ASGCHANGE_DATE (p_legislative_data_group_id, p_pay_assignment_id, p_event_group, p_start_date, p_end_date, p_recalc_date)**

Returns the earliest date when changes to the assignment were first detected between two dates.

*Parameters*

- **p_legislative_data_group_id**
  The legislative data group ID of the assignment.

- **p_pay_assignment_id**
  The assignment's pay assignment ID

- **p_event_group**
  The name of the event group that contains the events you want to track

- **p_start_date**
  Date when you want to start scanning for retrospective changes to the assignment

- **p_end_date**
  Date through which you want to scan for retrospective changes to the assignment

- **p_recalc_date**
  Date, normally today's date or the effective date you selected, on which you want to scan for retrospective events that may have occurred during the input dates.

**GET_PERIOD_DATES (base date, UOM, calculation date, number)**

Returns the start and end dates of the accrual period that spans the input date. The function assigns the start date to the PERIOD_START_DATE global variable, and the end date, to the PERIOD_END_DATE global variable.

For example, assume that an accrual plan was set up to allow employees to accrue time once in two months in a calendar year. The following usage of the GET_PERIOD_DATES function populates the PERIOD_START_DATE global variable with 1-MAR-2010, and the PERIOD_END_DATE global variable, with 30-APR-2010.

`GET_PERIOD_DATES('1-JAN-2010', 'M', '15-APR-2010', 2)`
If the calculation date is earlier than the base date, then the function calculates the accrual periods backwards from the base date.

**Parameters**

**Base Date**

Date to start calculating the accrual period dates from. Usually, this is the start date of the accrual term.

**Calculation Date**

Date for which you want to determine the accrual period that spans it.

**UOM**

Type of division that determines accrual periods. Valid units are D (days), M (months), W (weeks).

**Number**

The duration of the accrual period.

**GET_START_DATE (accrual start date, start of accrual term)**

Returns the date when the accrual formula must start calculating accruals.

If you are using a payroll balance and there are retrospective absence entries that have not already been used in an accrual calculation, the function returns the earliest start date of these entries.

However, in the predefined formula, if any unprocessed retrospective element entries are found, the formula always calculates accruals from the beginning of the accrual term.

If you are not using a payroll balance, the function returns the start date of the accrual term.

**GET_WORKING_DAYS (start date, end date)**

Returns the number of working days between the input dates

**CALCULATE_HOURS_WORKED (p_std_hours, p_std_freq, p_range_start, p_range_end)**

Returns the number of working hours between two dates.

For example, assuming that the assignment works 40 hours per week, the following statement returns the number of working hours between 01 January, 2010 and 31 January, 2010:

\[
E = \text{CALCULATE_HOURS_WORKED} (40, "Weekly", '01-Jan-2010', '31-Jan-2010')
\]

The values that you can specify for the p_std_freq parameter are:

- Weekly
- Monthly
- Yearly

**PUT_MESSAGE (expression)**

Adds a message for debugging purposes.
For example, use this function to generate a message from the accrual formula if an employee is not eligible to accrue time.

**GET_ELEMENT_ENTRY ( )**

Returns a value that indicates whether an assignment has enrolled in the accrual plan. The value 1 indicates that an element entry exists, and the value 0 indicates that no element entry exists.

**GET_PAYROLL_DETAILS (p_payroll_id, p_date_in_period)**

Returns payroll details, such as the start date of the payroll period, the end date, and the number of the payroll period that spans the input date. The function accepts the following parameters:

- `p_payroll_id`: The ID of the payroll
- `p_date_in_period`: The date for which you want to determine the payroll details

**Parameters**

- `p_payroll_id`: The ID of the payroll
- `p_date_in_period`: The date for which you want to determine the payroll details

**GET_PAYROLL_DTRANGE (p_payroll_id)**

Returns the start and end dates of the specified payroll.

**GET_PAYROLL_ID (p_pay_assignment_id, p_payroll_id, p_date_in_period)**

Returns the effective payroll ID for an assignment on an input date.

**GET_RETRO_ELEMENT ( )**

Retrieves retrospective elements to be tagged as processed.

**GET_TERMINATION_DATE (p_assignment_id)**

Returns the termination date of an assignment and sets the PER_TERMINATION_DATE context with this value.

**SET_ACCRUAL_BALANCE (p_pay_assignment_id, p_element_type, p_element_type_id, p_input_value_id, p_accrual_band_id, p_absence_type_id, p_absence_attendance_id, p_start_date, p_end_date, p_contributed_amt)**

Sets the accrual balance details. The function accepts the following parameters:

**Parameters**

- `p_pay_assignment_id`: The assignment for which you want to set the accrual balance details
- `p_element_type`:
The type of element for which you want to set the details. Valid types are Accrual, Carryover, Absence, or other types of elements you may have added to the net calculation rules when you created the accrual plan.

\[ p\_element\_type\_id \]

The ID of the element for which you want to set the balance details

\[ p\_input\_value\_id \]

The ID of the input value of the element for which you want to set the balance details

\[ p\_accrual\_band\_id \]

The ID of the accrual band that applies to the assignment

\[ p\_absence\_type\_id \]

The name of the absence type. This parameter is applicable when you want to set details for the absence element type.

\[ p\_absence\_attendance\_id \]

The ID of the absence record. This parameter is applicable when you want to set details for the absence element type.

\[ p\_start\_date \]

If you are using this function to set details for the Absence element type, then set the value of this parameter to the start date of the absence. For setting details for the Carryover element type, specify the effective date of the carried over time.

\[ p\_end\_date \]

If you are using this function to set details for the Absence element type, then set the value of this parameter to the end date of the absence. For setting details for the Carryover element type, specify the date when the carried over time expires.

\[ p\_contributed\_amt \]

If you are using this function to set details for the additional elements you added in the net calculation rules, set the value of this parameter to the total amount of time recorded in those elements.

**Formulas for Absence Benefit Plans**

**Participation and Rate Eligibility Formula Type for Absence Entitlement Plans**

The Evaluate Absence Plan Participation process runs the Participation and Rate Eligibility formula to enroll eligible employees in an absence benefit plan. The formula output indicates whether an absence entitlement plan exists for the type of absence recorded.

You must associate the formula with an eligibility profile of an absence benefit plan. The formula must belong to the Participation and Rate Eligibility formula type. Provide a meaningful name for the formula so that you can easily identify it.
**Contexts**

The following contexts are available to this formula.

- BUSINESS_GROUP_ID
- ASSIGNMENT_ID
- DATE_EARNED
- PGM_ID
- PL_ID
- OPT_ID
- LER_ID

**Database Items**

The following database items are available to formulas of this type.

- All columns except attribute columns from the tables:
  - ben_ler_f
  - ben_pgm_f
  - ben_pl_f
  - ben_pl_typ_f
  - ben_opt_f

The database items are based on the employee’s assignment ID.

**Return Values**

The following return value is available to this formula.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELIGIBLE</td>
<td>Char</td>
<td>Yes</td>
<td>The value of this variable is Y if an entitlement plan exists for the type of absence recorded. The value is N if no entitlement plan exists.</td>
</tr>
</tbody>
</table>

**Errors**

If any other return value is passed back to the formula, then you can process errors with BEN_91329_FORMULA_RETURN.

**Sample Formula**

You can either write your own formula, or use the following text to create the formula on the Fast Formula page:
DEFAULT FOR BEN_ABS_ABSENCE_CATEGORY IS '_DEFAULT_
DEFAULT FOR BEN_ABS_ABSENCE_TYPE_ID IS -987123654
DEFAULT FOR BEN_PLN_PL_ID IS -987123654

l_yn = 'N'
l_error = 0
l_absence_type_lookup_code = ','
l_absence_type_list_name = ','
l_truncated_yes_no = ','
l_error_message = ','
l_absence_type_meaning = ','
l_absence_category = ','

l_pl_id = BEN_PLN_PL_ID
l_absence_type_id = BEN_ABS_ABSENCE_TYPE_ID
l_abs_typ_id = to_char(l_absence_type_id)

l_absence_type_meaning = BEN_CHK_ABS_TYPE (l_abs_typ_id, l_yn)

IF NOT ISNULL(l_absence_type_meaning) = 'Y' THEN
  (l_yn = 'Y')

ELIGIBLE = l_yn
RETURN ELIGIBLE

Extra Inputs Formula Type for Absence Benefit Plans

When an employee records a long term absence, the Evaluate Absence Plan Participation process enrolls the employee in an absence benefit plan and runs the Extra Inputs formula to update the payroll element with details, such as the absence type, absence start and end dates, and the ID of the absence benefit plan that the employee was enrolled in.

You must associate the formula with a benefit rate for an absence benefit plan. The formula must belong to the Extra Input formula type. Provide a meaningful name for the formula so that you can easily identify it.

**Contexts**

The following contexts are available to this formula:

- BUSINESS_GROUP_ID
- ASSIGNMENT_ID
- DATE_EARNED
- ORGANIZATION_ID
- JURISDICTION_CODE
- PGM_ID
- PL_ID
- OPT_ID
- LER_ID
• PL_TYP_ID
• ACTY_BASE_RT_ID

**Database Items**

The following database items are available to formulas of this type:

- All columns except attribute columns from tables:
  - ben_ler_f
  - ben_pgm_f
  - ben_pl_f
  - ben_pl_typ_f
  - ben_opt_f
  - ben_acty_base_rt_f

The database items are based on the employee's assignment ID.

**Input Variables**

The following input values are available to this formula.

<table>
<thead>
<tr>
<th>Input Value</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEN_ABS_IV_ABSENCE_ATTENDANCE_ID</td>
<td>Char</td>
<td>Yes</td>
<td>Absence record ID</td>
</tr>
<tr>
<td>BEN_ABS_IV_ABSENCE_ATTENDANCE_TYPE_ID</td>
<td>Char</td>
<td>Yes</td>
<td>Absence type ID</td>
</tr>
<tr>
<td>BEN_ABS_IV_DATE_STA</td>
<td>Char</td>
<td>Yes</td>
<td>Absence start date</td>
</tr>
<tr>
<td>BEN_ABS_IV_DATE_END</td>
<td>Char</td>
<td>Yes</td>
<td>Absence end date</td>
</tr>
<tr>
<td>BEN_ABS_IV_ABSENCE</td>
<td>Char</td>
<td>Yes</td>
<td>Absence duration</td>
</tr>
</tbody>
</table>

**Return Values**

The following return values are available to this formula.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_absence_id</td>
<td>Char</td>
<td>Yes</td>
<td>Absence record ID</td>
</tr>
<tr>
<td>l_plan_id</td>
<td>Char</td>
<td>Yes</td>
<td>Absence benefit plan ID that the employee enrolled in</td>
</tr>
<tr>
<td>l_absence_start_date</td>
<td>Char</td>
<td>Yes</td>
<td>Absence start date</td>
</tr>
<tr>
<td>l_absence_end_date</td>
<td>Char</td>
<td>Yes</td>
<td>Absence end date</td>
</tr>
<tr>
<td>l_absence_type</td>
<td>Char</td>
<td>Yes</td>
<td>Type of absence recorded</td>
</tr>
</tbody>
</table>

**Errors**

If type casting of variables causes errors, then you can process those errors with BEN_92311_FORMULA_VAL_PARAM.
Sample Formula

You can either write your own formula, or use the following text to create the formula in the Fast Formula page:

```plaintext
/*
  Set default values for database items.
*/
DEFAULT FOR BEN_ABS_ABSENCE_TYPE IS '_DEFAULT_'
DEFAULT FOR BEN_PLAN_PL_ID IS -987123654

/* Other database items.
DEFAULT FOR BEN_ABS_ABSENCE_TYPE IS -987123654
DEFAULT FOR BEN_ABS_ABSENCE_CATEGORY IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_ABSENCE_CATEGORY_CODE IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_ABSENCE_CATEGORY_ID IS -987123654
DEFAULT FOR BEN_ABS_REASON IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_REASON_CODE IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_REASON_ID IS -987123654
DEFAULT FOR BEN_ABS_DATE_START IS '1951/01/01 00:00:00'(DATE)
DEFAULT FOR BEN_ABS_DATE_END IS '1951/01/01 00:00:00'(DATE)
DEFAULT FOR BEN_ABS_SICKNESS_START_DATE IS '1951/01/01 00:00:00'(DATE)
DEFAULT FOR BEN_ABS_SICKNESS_END_DATE IS '1951/01/01 00:00:00'(DATE)
DEFAULT FOR BEN_ABS_DATE_NOTIFIED IS '1951/01/01 00:00:00'(DATE)
DEFAULT FOR BEN_SMP_DUE_DATE IS '1951/01/01 00:00:00'(DATE)
DEFAULT FOR BEN_SMP_MPP_START_DATE IS '1951/01/01 00:00:00'(DATE)
DEFAULT FOR BEN_SMP_ACTUAL_BIRTH_DATE IS '1951/01/01 00:00:00'(DATE)
DEFAULT FOR BEN_SMP_LIVE_BIRTH_FLAG IS 'Y'
DEFAULT FOR BEN_SSP_EVIDENCE_DATE IS '1951/01/01 00:00:00'(DATE)
DEFAULT FOR BEN_SSP_EVIDENCE_SOURCE IS '_DEFAULT_'
DEFAULT FOR BEN_SSP_MEDICAL_TYPE IS 'SICKNESS'
DEFAULT FOR BEN_SSP_EVIDENCE_STATUS IS 'ACTIVE'
DEFAULT FOR BEN_SSP_SELF_CERTIFICATE IS 'N'
DEFAULT FOR BEN_ABS_ACCEPT_LATE_NOTIFICATION_FLAG IS 'Y'
DEFAULT FOR BEN_ABS_PREGNANCY_RELATED_ILLNESS IS 'N'
DEFAULT FOR BEN_SMP_NOTIFICATION_OF_BIRTH_DATE IS '1951/01/01 00:00:00'(DATE)
DEFAULT FOR BEN_SSP_EVIDENCE_RECEIVED_DATE IS '1951/01/01 00:00:00'(DATE)
DEFAULT FOR BEN_SSP_ACCEPT_LATE_EVIDENCE_FLAG IS 'Y'

/*
  Set default values for formula inputs.
*/
DEFAULT FOR BEN_ABS_IV_ABSENCE_ATTENDANCE_ID IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_ABSENCE_ATTENDANCE_TYPE_ID IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_DATE_START IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_DATE_END IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_ABSENCE_DAYS IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_ABS_ATTENDANCE_REASON_ID IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_ABSENCE_HOURS IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_DATE_NOTIFICATION IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_DATE_PROJECTED_END IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_DATE_PROJECTED_START IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_SSP1_ISSUED IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_LINKED_ABSENCE_ID IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_SICKNESS_START_DATE IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_SICKNESS_END_DATE IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_PREGNANCY_RELATED_ILLNESS IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_MATERNITY_ID IS '_DEFAULT_'
DEFAULT FOR BEN_PIL_IV_PER_IN_LER_ID IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_ATTRIBUTE_CATEGORY IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_ATTRIBUTE1 IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_ATTRIBUTE2 IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_ATTRIBUTE3 IS '_DEFAULT_'
DEFAULT FOR BEN_ABS_IV_ATTRIBUTE4 IS '_DEFAULT_'
*/```
DEFAULT FOR BEN_ABS_IV_ATTRIBUTE5 IS '_DEFAULT_
DEFAULT FOR BEN_ABS_IV_ATTRIBUTE6 IS '_DEFAULT_
DEFAULT FOR BEN_ABS_IV_ATTRIBUTE7 IS '_DEFAULT_
DEFAULT FOR BEN_ABS_IV_ABS_INFORMATION_CATEGORY IS '_DEFAULT_
DEFAULT FOR BEN_ABS_IV_ABS_INFORMATION1 IS '_DEFAULT_
DEFAULT FOR BEN_ABS_IV_ABS_INFORMATION2 IS '_DEFAULT_
DEFAULT FOR BEN_ABS_IV_ABS_INFORMATION3 IS '_DEFAULT_
DEFAULT FOR BEN_ABS_IV_ABS_INFORMATION4 IS '_DEFAULT_
DEFAULT FOR BEN_ABS_IV_ABS_INFORMATION5 IS '_DEFAULT_
DEFAULT FOR BEN_ABS_IV_ABS_INFORMATION6 IS '_DEFAULT_
DEFAULT FOR BEN_ABS_IV_ABS_INFORMATION7 IS '_DEFAULT_
*/

/*
Declare input values.

Use the following naming convention for the inputs:
BEN_ABS_IV_
*/

INPUTS ARE BEN_ABS_IV_ABSENCE_ATTENDANCE_ID(TEXT)
, BEN_ABS_IV_ABSENCE_ATTENDANCE_TYPE_ID(TEXT)
, BEN_ABS_IV_DATE_START(TEXT)
, BEN_ABS_IV_DATE_END(TEXT)
, BEN_ABS_IV_ABSENCE_DAYS(TEXT)

/*/ Other available inputs
, BEN_ABS_IV_ABSENCE_ATTENDANCE_REASON_ID(TEXT)
, BEN_ABS_IV_ABSENCE_HOURS(TEXT)
, BEN_ABS_IV_DATE_NOTIFICATION(TEXT)
, BEN_ABS_IV_DATE_PROJECTED_END(TEXT)
, BEN_ABS_IV_DATE_PROJECTED_START(TEXT)
, BEN_ABS_IV_SSP1_ISSUED(TEXT)
, BEN_ABS_IV_LINKED_ABSENCE_ID(TEXT)
, BEN_ABS_IV_SICKNESS_START_DATE(TEXT)
, BEN_ABS_IV_SICKNESS_END_DATE(TEXT)
, BEN_ABS_IV_PREGNANCY_RELATED_ILLNESS(TEXT)
, BEN_ABS_IV_MATERNITY_ID(TEXT)
, BEN_PIL_IV_PER_IN_LER_ID(TEXT)
, BEN_ABS_IV_ATTRIBUTE_CATEGORY(TEXT)
, BEN_ABS_IV_ATTRIBUTE1(TEXT)
, BEN_ABS_IV_ATTRIBUTE2(TEXT)
, BEN_ABS_IV_ATTRIBUTE3(TEXT)
, BEN_ABS_IV_ATTRIBUTE4(TEXT)
, BEN_ABS_IV_ATTRIBUTE5(TEXT)
, BEN_ABS_IV_ATTRIBUTE6(TEXT)
, BEN_ABS_IV_ATTRIBUTE7(TEXT)
, BEN_ABS_IV_ABS_INFORMATION_CATEGORY(TEXT)
, BEN_ABS_IV_ABS_INFORMATION1(TEXT)
, BEN_ABS_IV_ABS_INFORMATION2(TEXT)
, BEN_ABS_IV_ABS_INFORMATION3(TEXT)
, BEN_ABS_IV_ABS_INFORMATION4(TEXT)
, BEN_ABS_IV_ABS_INFORMATION5(TEXT)
, BEN_ABS_IV_ABS_INFORMATION6(TEXT)
, BEN_ABS_IV_ABS_INFORMATION7(TEXT)
*/

/*/ Initialise standard default values. */
l_null = RPAD('X',0,'Y')
l_default = '_DEFAULT_'
l_default_date = '1951/01/01 00:00:00' (date)
l_default_canonical_date = '1951/01/01 00:00:00'
l_default_number = -987123654
l_default_canonical_number = '-987123654'
l_absence_id_iv = BEN_ABS_IV_ABSENCE_ATTENDANCE_ID

/* 1. Check that a default value was not used for the absence attendance ID.

If an absence attendance id was not found, the default value is used. This may occur if this formula is used in a plan that does not have an absence "context" available. Ensure that you select Absence as the option type of the associated plan type. Ensure that you select Absence as the type of the associated life event reasons.
*/

IF NOT l_absence_id_iv = l_default THEN

(l_absence_id = TO_NUMBER(l_absence_id_iv))
l_plan_id = BEN_PLN_PL_ID
l_absence_start_date = BEN_ABS_IV_DATE_START
l_absence_type = BEN_ABS_ABSENCE_TYPE
l_absence_end_date = BEN_ABS_IV_DATE_END
}
ELSE

(l_absence_id = l_default_number)
l_plan_id = l_default_number
l_absence_start_date = l_null
l_absence_end_date = l_null
l_absence_type = l_null
}

RETURN l_absence_id
,l_plan_id
,l_absence_start_date
,l_absence_end_date
,l_absence_type

**Rate Value Calculation Formula Type for Absence Benefit Plans**

When an employee records a long term absence, the Evaluate Absence Plan Participation process enrolls the employee in an absence benefit plan and runs the Rate Value Calculation formula to determine the enrolled employee's length of service. The process uses the length of service to determine the entitlement bands that apply during the absence period.

You must create the formula for the standard benefit rate that you want to associate with the absence benefit plan. The formula must belong to the Rate Value Calculation formula type. Provide a meaningful name for the formula so that you can easily identify it.

**Contexts**

The following contexts are available to this formula:

- BUSINESS_GROUP_ID
- ASSIGNMENT_ID
- DATE_EARNED or p_effective_date

**Database Items**

The database items available to this formula are based on the employee's assignment ID.
### Return Values

The following return value is available to this formula.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LENGTH_OF_SERVICE</td>
<td>Number</td>
<td>Yes</td>
<td>A number that indicates the length of service of the enrolled employee.</td>
</tr>
</tbody>
</table>

### Sample Formula

You can either write your own formula, or use the following text to create the formula in the Fast Formula page:

```plaintext
/*
Set default values.
*/
DEFAULT FOR BEN_ABS_DATE_START IS '1951/01/01 00:00:00'(date)
DEFAULT FOR PER_ASG_REL_DATE_START IS '1951/01/01 00:00:00'(date)

/*
Initialize standard default values.
*/
l_null = RPAD('X',0,'Y')
l_default = '_DEFAULT_'
l_default_date = '1951/01/01 00:00:00'(date)
l_default_canonical_date = '1951/01/01 00:00:00'
l_default_number = -987123654
l_default_canonical_number = '-987123654'
l_length_of_service = -987123654

/*
Determine the absence start date and the employee hire date.
*/
l_absence_start_date = BEN_ABS_DATE_START
l_employee_hire_date = PER_ASG_REL_DATE_START

/*
Check that an absence start date is available for processing.
If an absence start date was not found, the default value is used. This may occur if this formula is used in a plan that does not have an absence "context" available. Ensure that you select Absences as the option type of the associated plan type. Ensure that you select Absence as the type of the associated life event reasons.
*/
IF NOT l_absence_start_date = l_default_date THEN
{
/*
Check that an absence start date is available to process.
If an employee hire date was not found, the default value is used. This may occur if the person was not an eligible person type. Check the associated eligibility profile to ensure that only persons belonging to the Employee person type are selected for plan enrollment.
*/
IF NOT l_employee_hire_date = l_default_date THEN
{
/*
Calculate the length of service.
*/
```
l_length_of_service = FLOOR( MONTHS_BETWEEN (l_absence_start_date, l_employee_hire_date))
}

LENGTH_OF_SERVICE = l_length_of_service
RETURN LENGTH_OF_SERVICE
Compensation Currency Selection Formula Type

The Compensation Currency Selection formula determines the currency associated with a workforce compensation component.

You select the formula on the Configure Compensation Components page.

**Contexts**

The following contexts are available to formulas of this type:

- EFFECTIVE_DATE
- HR_ASSIGNMENT_ID

**Database Items**

Database items are not available to formulas of this type.

**Input Variables**

The following input variables are available to formulas of this type.

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP_IV_PLAN_ID</td>
<td>Number</td>
<td>Y</td>
<td>Plan ID</td>
</tr>
<tr>
<td>CMP_IV_ASSIGNMENT_ID</td>
<td>Number</td>
<td>Y</td>
<td>Assignment ID</td>
</tr>
<tr>
<td>CMP_IV_PERIOD_ID</td>
<td>Number</td>
<td>Y</td>
<td>Period ID</td>
</tr>
<tr>
<td>CMP_IV_COMPONENT_ID</td>
<td>Number</td>
<td>Y</td>
<td>Component ID</td>
</tr>
<tr>
<td>CMP_IV_PLAN_START_DATE</td>
<td>Date</td>
<td>Y</td>
<td>Plan Start Date</td>
</tr>
<tr>
<td>CMP_IV_PLAN_END_DATE</td>
<td>Date</td>
<td>Y</td>
<td>Plan End Date</td>
</tr>
</tbody>
</table>

**Return Values**

Use predefined names for return variables. The following return variables are available to formulas of this type.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_CURR_CODE</td>
<td>Char</td>
<td>N</td>
<td>Currency code from the formula</td>
</tr>
</tbody>
</table>
**Sample Formula**

This sample formula determines if a person is selected for a plan based on their assignment_id.

```plaintext
/************************************************************
FORMULA NAME : Compensation Currency Selection Formula
FORMULA TYPE : Compensation Currency Selection
DESCRIPTION : It returns the currency code based on component_id.
***************************************************************************/

/*=========== INPUT VALUES DEFAULTS BEGIN =====================*/
INPUTS ARE CMP_IV_ASSIGNMENT_ID (number), CMP_IV_PLAN_ID (number),
CMP_IV_PERIOD_ID (number), CMP_IV_COMPONENT_ID (number)
/*=========== INPUT VALUES DEFAULTS ENDS======================*/

/*================ FORMULA SECTION BEGIN =======================*/
DEFAULT FOR CMP_IV_COMPONENT_ID IS 0
l_curr_code = 'XXX'
IF (CMP_IV_COMPONENT_ID = 489) THEN
  l_curr_code = 'USD'
ELSE IF (CMP_IV_COMPONENT_ID = 490) THEN
  l_curr_code = 'GBP'
RETURN l_curr_code
/*================ FORMULA SECTION END =======================*/

**Compensation Default and Override Formula Type**

The Compensation Default and Override formula determines the default values populated in a column for a workforce compensation plan.

You select the formula when you configure the worksheet display for a column using the Default Values tab on the Define Column Properties window.

**Contexts**

The following contexts are available to formulas of this type:

- EFFECTIVE_DATE
- HR_ASSIGNMENT_ID
- PAYROLL_ASSIGNMENT_ID
- DATE_EARNED
- JOB_ID
- HR_TERM_ID

**Database Items**

Database items are not available to formulas of this type.
**Input Variables**

The following input variables are available to formulas of this type.

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP_IV_PLAN_ID</td>
<td>Number</td>
<td>Y</td>
<td>Plan ID</td>
</tr>
<tr>
<td>CMP_IV_PERIOD_ID</td>
<td>Number</td>
<td>Y</td>
<td>Period ID</td>
</tr>
<tr>
<td>CMP_IV_COMPONENT_ID</td>
<td>Number</td>
<td>Y</td>
<td>Component ID</td>
</tr>
<tr>
<td>CMP_IV_ITEM_NAME</td>
<td>Char</td>
<td>Y</td>
<td>Item Name</td>
</tr>
<tr>
<td>CMP_IV_PERSON_ID</td>
<td>Number</td>
<td>Y</td>
<td>Person ID</td>
</tr>
<tr>
<td>CMP_IV_PLAN_START_DATE</td>
<td>Date</td>
<td>Y</td>
<td>Plan Start Date</td>
</tr>
<tr>
<td>CMP_IV_PLAN_END_DATE</td>
<td>Date</td>
<td>Y</td>
<td>Plan End Date</td>
</tr>
</tbody>
</table>

**Return Values**

Use predefined names for return variables. The following return variables are available to formulas of this type.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_DEFAULT_VALUE</td>
<td>Number/Char/Date</td>
<td>Y</td>
<td>Default value from the formula</td>
</tr>
<tr>
<td>L_DATA_TYPE</td>
<td>Char</td>
<td>Y</td>
<td>Data type of the column</td>
</tr>
</tbody>
</table>

**Sample Formula**

This sample formula determines if a person is selected for a plan based on their assignment_id.

```sql
/*Formula Name: Compensation Default and Override Formula*/
/*Formula Type: Compensation Default and Override*/
/*Description: Defaults the value of a column based on its item_name.*/
DEFAULT FOR CMP_IV_ITEM_NAME IS 'YYYYYYY'
L_DEFAULT_VALUE = to_char(0)
L_DATA_TYPE = 'NONETYPE'
//Valid L_DATA_TYPE values can be -
//NUMBER, CHAR, DATE.
IF (CMP_IV_ITEM_NAME = 'AmountComp1') THEN
  L_DATA_TYPE = 'NUMBER'
  L_DEFAULT_VALUE = to_char(3333)
```
```
ELSE IF (CMP_IV_ITEM_NAME = 'AmountComp2') THEN
    { 
    L_DATA_TYPE = 'NUMBER'
    L_DEFAULT_VALUE = to_char(7777)
    }
ELSE
    { 
    L_DATA_TYPE = 'NONETYPE'
    L_DEFAULT_VALUE = to_char(-999)
    }
RETURN L_DATA_TYPE, L_DEFAULT_VALUE
/*---------------- FORMULA SECTION END ------------------------*/
```

Compensation Hierarchy Determination Formula Type

The Compensation Hierarchy Determination formula determines the hierarchy for an associated workforce compensation plan.

You select the formula on the Configure Plan Details page.

**Contexts**

The following contexts are available to formulas of this type:

- EFFECTIVE_DATE
- HR_ASSIGNMENT_ID

**Database Items**

Database items are not available to formulas of this type.

**Input Variables**

The following input variables are available to formulas of this type.

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP_IV_ASSIGNMENT_ID</td>
<td>Number</td>
<td>Y</td>
<td>Assignment ID</td>
</tr>
<tr>
<td>CMP_IV_PLAN_ID</td>
<td>Number</td>
<td>Y</td>
<td>Plan ID</td>
</tr>
<tr>
<td>CMP_IV_PERIOD_ID</td>
<td>Number</td>
<td>Y</td>
<td>Period ID</td>
</tr>
<tr>
<td>CMP_IV_PLAN_START_DATE</td>
<td>Date</td>
<td>Y</td>
<td>Plan Start Date</td>
</tr>
<tr>
<td>CMP_IV_PLAN_END_DATE</td>
<td>Date</td>
<td>Y</td>
<td>Plan End Date</td>
</tr>
</tbody>
</table>

**Return Values**

Use predefined names for return variables. The following return variables are available to formulas of this type.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_PERSON_ID</td>
<td>Number</td>
<td>Y</td>
<td>Person ID of manager</td>
</tr>
<tr>
<td>L_ASSIGNMENT_ID</td>
<td>Number</td>
<td>Y</td>
<td>Assignment ID of manager</td>
</tr>
</tbody>
</table>
Sample Formula

This sample formula determines the manager of a person when the assignment_id is passed.

```c
/*=========== INPUT VALUES DEFAULTS BEGIN =====================*/
INPUTS ARE CMP_IV_ASSIGNMENT_ID (number), CMP_IV_PLAN_ID (number),
CMP_IV_PERIOD_ID (number)
/*=========== INPUT VALUES DEFAULTS ENDS======================*/

/*================ FORMULA SECTION BEGIN =======================*/
DEFAULT FOR CMP_IV_ASSIGNMENT_ID IS 0
L_PERSON_ID = '0'
L_ASSIGNMENT_ID = '0'
if ( CMP_IV_ASSIGNMENT_ID = 100000008154060 ) THEN
  (L_PERSON_ID = to_char(-999) //-999 indicates top level
  //Manager.
  L_ASSIGNMENT_ID = to_char(-999))
ELSE
  (L_PERSON_ID = to_char(100000008153756)
  L_ASSIGNMENT_ID = to_char(100000008154060))
RETURN L_PERSON_ID , L_ASSIGNMENT_ID
/*================ FORMULA SECTION END =======================*/
```

Compensation Person Selection Formula Type

The Compensation Person Selection formula determines whether a person is selected for an associated workforce compensation plan.

You select the formula when you run the Start Workforce Compensation Cycle process.

**Contexts**

The following contexts are available to formulas of this type:

- EFFECTIVE_DATE
- PAYROLL_ASSIGNMENT_ID
- DATE_EARNED

**Database Items**

Database items are not available to formulas of this type.

**Input Variables**

The following input variables are available to formulas of this type.
## Return Values

Use predefined names for return variables. The following return variables are available to formulas of this type.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_SELECTED</td>
<td>Char</td>
<td>N</td>
<td>Y or N</td>
</tr>
</tbody>
</table>

## Sample Formula

This sample formula determines if a person is selected for a plan based on their assignment_id.

```
/*-------------------- INPUT VALUES DEFAULTS BEGIN ------------------------*/
INPUTS ARE CMP_IV_ASSIGNMENT_ID (number), CMP_IV_PLAN_ID (number)
/*-------------------- INPUT VALUES DEFAULTS ENDS-------------------------*/

DEFAULT FOR CMP_IV_ASSIGNMENT_ID IS 0

l_selected = 'Y'

if (CMP_IV_ASSIGNMENT_ID = 100000008154095) THEN
    l_selected = 'N'
else
    l_selected = 'Y'
RETURN l_selected
/*-------------------- FORMULA SECTION END -------------------------------*/
```

## Compensation Start Date Formula Type

The Compensation Start Date formula can be used to determine the start date of compensation awarded using an individual compensation plan.

When setting up a compensation plan, you can select the defined formula after you select Formula as the Compensation Start Date value.

## Contexts

The following contexts are available to formulas of this type:

- EFFECTIVE_DATE
- HR_ASSIGNMENT_ID
**Database Items**

Database items are not available to formulas of this type.

**Input Variables**

The following input variables are available to formulas of this type:

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP_IV_EFFECTIVE_DATE</td>
<td>Varchar2</td>
<td>N</td>
<td>Input effective date</td>
</tr>
<tr>
<td>CMP_IV_ASSIGNMENT_ID</td>
<td>Number</td>
<td>N</td>
<td>Assignment ID</td>
</tr>
<tr>
<td>CMP_IV_PERSON_ID</td>
<td>Number</td>
<td>N</td>
<td>Person ID</td>
</tr>
<tr>
<td>CMP_IV_PLAN_ID</td>
<td>Number</td>
<td>N</td>
<td>Plan ID</td>
</tr>
<tr>
<td>CMP_IV_OPTION_ID</td>
<td>Number</td>
<td>N</td>
<td>Option ID</td>
</tr>
</tbody>
</table>

**Return Values**

The following return variables are available to formulas of this type:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPENSATION_START_DATE</td>
<td>Varchar2</td>
<td>N</td>
<td>Compensation Start Date</td>
</tr>
</tbody>
</table>

**Errors**

CMP_VC_ALLOC_FORMULA_FAILURE - Error occurred during formula processing.

**Sample Formula**

This sample formula calculates the compensation start date to be 3 months later from the given date.

```plaintext
/******************************/
FORMULA NAME: Compensation Start Date Formula
FORMULA TYPE: Compensation Start Date
DESCRIPTION: Formula that returns Compensation Start Date to be 3 month later of the given date
*****************************/
/*--------------- DATABASE ITEM DEFAULTS BEGIN ----------------*/
N/A
/*--------------- DATABASE ITEM DEFAULTS ENDS ----------------*/
/*--------------- FORMULA SECTION BEGIN ----------------------*/
INPUTS ARE CMP_IV_EFFECTIVE_DATE (text)
DEFAULT FOR CMP_IV_EFFECTIVE_DATE IS '4012/01/01'
l_date = ADD_MONTHS(TO_DATE(CMP_IV_EFFECTIVE_DATE, 'YYYY/MM/DD'),3)
compensation_start_date = TO_CHAR(ldate, 'YYYY/MM/DD')
RETURN compensation_start_date
/*--------------- FORMULA SECTION END ------------------------*/
```

**Compensation End Date Formula Type**

The Compensation End Date formula can be used to determine the end date of compensation awarded using an individual compensation plan.
When setting up a compensation plan, you can select the defined formula after you select Formula as the Compensation End Date value.

**Contexts**

The following contexts are available to formulas of this type:

- EFFECTIVE_DATE
- HR_ASSIGNMENT_ID

**Database Items**

Database items are not available to formulas of this type.

**Input Variables**

The following input variables are available to formulas of this type:

<table>
<thead>
<tr>
<th>Input</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP_IV_EFFECTIVE_DATE</td>
<td>Varchar2</td>
<td>N</td>
<td>Input effective date</td>
</tr>
<tr>
<td>CMP_IV_ASSIGNMENT_ID</td>
<td>Number</td>
<td>N</td>
<td>Assignment ID</td>
</tr>
<tr>
<td>CMP_IV_PERSON_ID</td>
<td>Number</td>
<td>N</td>
<td>Person ID</td>
</tr>
<tr>
<td>CMP_IV_PLAN_ID</td>
<td>Number</td>
<td>N</td>
<td>Plan ID</td>
</tr>
<tr>
<td>CMP_IV_OPTION_ID</td>
<td>Number</td>
<td>N</td>
<td>Option ID</td>
</tr>
</tbody>
</table>

**Return Values**

The following return variables are available to formulas of this type.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPENSATION_END_DATE</td>
<td>Varchar2</td>
<td>N</td>
<td>Compensation End Date</td>
</tr>
</tbody>
</table>

**Errors**

CMP_VC_ALLOC_FORMULA_FAILURE - Error occurred during formula processing.

**Sample Formula**

This sample formula calculates the compensation end date to be 6 months later of the given date.

```sql
/*FORMULA NAME : Compensation End Date Formula
FORMULA TYPE : Compensation End Date
DESCRIPTION : Formula that returns Compensation End Date
****************************************************************************/
/*============= DATABASE ITEM DEFAULTS BEGIN ==============*/
N/A
/*============= DATABASE ITEM DEFAULTS ENDS ===============*/
/*============= FORMULA SECTION BEGIN =====================*/
INPUTS ARE CMP_IV_EFFECTIVE_DATE (text)
DEFAULT FOR CMP_IV_EFFECTIVE_DATE IS '4012/01/01'

l_date = ADD_MONTHS(TO_DATE(CMP_IV_EFFECTIVE_DATE, 'YYYY/MM/DD'),6)
compensation_end_date = TO_CHAR(ldate, 'YYYY/MM/DD')```
Total Compensation Item Formula Type

The Total Compensation Item formula determines compensation information that is not stored in the other predefined item source types.

You select the formula when you manage compensation items on the Create or Edit Compensation Items page.

**Contexts**

The EFFECTIVE_DATE (statement end date) context is available to formula of this type.

**Database Items**

Database items are not available for this formula type.

**Input Variables**

The following input variables are available to formula of this type.

<table>
<thead>
<tr>
<th>Input Value</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP_IV_PERSON_ID</td>
<td>Char</td>
<td>Y</td>
<td>Worker ID</td>
</tr>
<tr>
<td>CMP_IV_PERIOD_START</td>
<td>Date</td>
<td>Y</td>
<td>Statement Period Start Date</td>
</tr>
<tr>
<td>CMP_IV_PERIOD_END</td>
<td>Date</td>
<td>Y</td>
<td>Statement Period End Date</td>
</tr>
</tbody>
</table>

**Return Values**

Use defined names for return variables. The following return variables are available to formula of this type.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPENSATION_DATE</td>
<td>Date</td>
<td>Y</td>
<td>One to 15 transaction dates delimited by semicolon, maximum 250 characters.</td>
</tr>
<tr>
<td>VALUES</td>
<td>Char</td>
<td>Y</td>
<td>One to 15 transaction values delimited by semicolon, maximum 250 characters. Must be the same number of values as dates.</td>
</tr>
<tr>
<td>ASSIGNMENTS</td>
<td>Char</td>
<td>N</td>
<td>One to 15 transaction assignments delimited by semicolon, maximum 250 characters. Must be the same number of assignments as dates. Can return an empty space with a delimiter (; ;).</td>
</tr>
</tbody>
</table>
**LEGALEMPLOYERS**  Char  N  One to 15 legal employer IDs delimited by semicolon, maximum 250 characters. Must be the same number of assignments as dates. Can return an empty space with a delimiter (; ;).

**COMPENSATION_DATE**  Date  Y  Second variable for transaction dates from 16 to 30 if limit of 250 characters is exceeded.

**VALUES1**  Char  Y  Second variable for transaction values from 16 to 30 if limit of 250 characters is exceeded.

**ASSIGNMENTS1**  Char  N  Second variable for transaction assignments from 16 to 30 if limit of 250 characters is exceeded.

**LEGALEMPLOYERS1**  Char  N  Second variable for legal employer IDs from 16 to 30 if limit of 250 characters is exceeded.

**COMPENSATION_DATE**  Date  Y  Transaction dates from 31 to 45.

**VALUES2**  Char  Y  Transaction values from 31 to 45.

**ASSIGNMENTS2**  Char  N  Transaction assignments from 31 to 45.

**LEGALEMPLOYERS2**  Char  N  Legal employers from 31 to 45.

**COMPENSATION_DATE**  Dates  Y  Transaction dates from 46 to 60.

**VALUES3**  Char  Y  Transaction values from 46 to 60.

**ASSIGNMENTS3**  Char  N  Transaction assignments from 46 to 60.

**LEGALEMPLOYERS3**  Char  N  Legal employers from 46 to 60.

**Sample Formula**

This sample formula returns one date and one value based on the worker ID.

```c
/*****************************************************************************
FORMULA NAME : Total Compensation Simple Item Formula
FORMULA TYPE : Total Compensation Item
DESCRIPTION : Returns one date and one value.
****************************************************************************/

/****************************************************************************
*----------- INPUT VALUES DEFAULTS BEGIN ----------------------------------*/
```
This sample formula returns multiple variables.

----------------------------------------------------------------------------------------------------------------------------------
FORMULA NAME : Total Compensation Multi Item Formula  
FORMULA TYPE : Total Compensation Item  
DESCRIPTION : Returns multiple variables.  
----------------------------------------------------------------------------------------------------------------------------------

/*----------- INPUT VALUES DEFAULTS BEGIN -------------------------------*/
INPUTS ARE CMP_IV_PERSON_ID (text), CMP_IV_PERIOD_START_DATE (date), CMP_IV_PERIOD_END_DATE (date)
/*----------- INPUT VALUES DEFAULTS ENDS -------------------------------*/

/*---------------- FORMULA SECTION BEGIN -------------------------------*/

COMPENSATION_DATES = '2009/01/01;2009/02/01;2009/03/01'  
COMPENSATION_DATES1 = '2009/07/01;2009/08/01;2009/09/01'  
COMPENSATION_DATES2 = '2009/10/01;2009/11/01;2009/12/01'  
COMPENSATION_DATES3 = '2009/10/01;2009/11/01;2009/12/01'  
VALUES = '200.00;200.00;300.00'  
VALUES1 = '300.00;500.00;500.00'  
VALUES2 = '500.00;500.00;600.00'  
VALUES3 = '600.00;600.00;700.00'  

/* Returns only first two assignment */  
ASSIGNMENTS = ';1234567890;1234567890'  
ASSIGNMENTS1 = '1234567890;1234567890;1234567890'  
/* Returns last two assignments */  
ASSIGNMENTS2 = ';1234567890;1234567890'  
/* Returns first and last assignments */  
ASSIGNMENTS3 = '1234567890;1234567890;1234567890'  

LEGALEMPLOYERS = '0123456789;0123456789'  
LEGALEMPLOYERS1 = '0123456789;0123456789;0123456789'  
LEGALEMPLOYERS2 = '0123456789;0123456789;0123456789'  
LEGALEMPLOYERS3 = '0123456789;0123456789'  

RETURN  
COMPENSATION_DATES,VALUES,COMPENSATION_DATES1,VALUES1,COMPENSATION_DATES2,VALUES2,COMPENSATION_DATES3,VALUES3,ASSIGNMENTS,ASSIGNMENTS1,ASSIGNMENTS2,ASSIGNMENTS3,LEGALEMPLOYERS,LEGALEMPLOYERS1,LEGALEMPLOYERS2,LEGALEMPLOYERS3  
/*---------------- FORMULA SECTION END -------------------------------*/
**absence entitlement band**

A level that determines the payment that employees must receive for a specific number of days during a long leave of absence based on their length of service.

**absence entitlement plan**

A benefit that entitles employees to receive payment while on long leaves of absence.

**absence type**

A grouping of absences, such as illness or personal business, that are handled together for reporting, accrual, and compensation calculations.

**accrual band**

A range of eligibility criteria that identify how much paid time eligible employees accrue over the course of an accrual term. The criteria may be years of service, grades, hours worked, or some other factor.

**accrual carryover**

Amount of unused time in the previous accrual term that an employee can use in the next accrual term.

**accrual ceiling**

Maximum amount of time that an employee can accrue in an accrual term.

**accrual period**

A time interval, such as one month, in which employees accrue time within an accrual term.

**accrual plan**

A benefit that entitles employees to accrue time for the purpose of taking leave.

**accrual term**

Period of time, often one year, for which accruals are calculated.

**assignment statement**

A statement used to set a value for a local variable in a fast formula.

**balance**

Positive or negative accumulations of values over periods of time normally generated by payroll runs. A balance can sum pay values, time periods, or numbers.
element
Component in the calculation of a person’s pay. An element may represent a compensation or benefit type, such as salary, wages, stock purchase plans, pension contributions, and medical insurance.

fast formula
A simple way to write formulas using English words and basic mathematical functions. Formulas are generic expressions of calculations or comparisons you want to repeat with different input values.

globals
Used to store values that are constant over a period of time and may be referenced in several formulas. For example, the name of a rate, a specific date, or a company term.

gross accrual
Amount of time that an employee has accumulated in an accrual plan.

local variable
A variable used in only one formula. The value of a local variable can change by assigning a value in an assignment statement.

net accrual
Amount of time in an accrual plan that an enrolled employee can use to take leave.

object group
User-defined set of elements or people used to restrict which of these items to include in various processes and reports.

return statement
A statement used to return values in local variables in a fast formula.

user-defined table
Structures of rows and columns that maintain date-tracked lists of values. The values are stored as cells for specific row and column combinations.