Oracle Global Human Resources Cloud
Using Fast Formula
This guide also applies to on-premise implementations

Release 8

April 2014
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 ..............................................2-7

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 ........................................3-11
   Literals: Explained ......................................................3-11
   Generating Flexfield Database Items: Explained .......................3-13
   Formula Variable Data Types: How They Are Determined .................3-14
   Generating Flexfield Database Items: Explained .......................3-15
   Array Variables: Explained ...........................................3-16
   Formula Contexts: Explained ..........................................3-18
   Working Storage Area: Explained ....................................3-19
   Calling a Formula from a Formula: Explained ...........................3-20
   Calling a Formula from a Formula: Examples ...........................3-23

4 Formula Functions
   Formula Functions: Explained ............................................4-1
   Text Formula Functions ..................................................4-1
   Numeric Formula Functions .............................................4-4
   Date Formula Functions ..................................................4-5
   Data Conversion Formula Functions ....................................4-7
   Miscellaneous Formula Functions .....................................4-8

5 Formulas for Payroll
6 Formulas for Absence Management

Qualification Absence Plan Rules: Points to Consider ........................................ 6-1
Accrual Absence Plan Rules: Points to Consider ........................................ 6-2
Formulas for Accrual Plan Rules: Explained .........................................................6-4
Formulas for Absence Qualification Plan Rules: Explained .................................6-5
Formulas for Absence Type Rules: Explained .......................................................6-6

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 ..................................... 7-5
Compensation Person Selection Formula Type ........................................ 7-6
Compensation Start Date Formula Type ......................................................... 7-8
Compensation End Date Formula Type .......................................................... 7-9
Total Compensation Item Formula Type ......................................................... 7-10
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.

Note

If you don't see any help icons on your page, then click the Show Help icon button in the global area. However, not all pages have help icons.

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. To access the guides, go to any page in Oracle Fusion Applications Help and select Documentation Library from the Navigator menu.

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>Functional Setup Manager User Guide</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>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>
<tr>
<td></td>
<td></td>
<td><strong>Note</strong> Limited content applicable to Oracle Cloud implementations.</td>
</tr>
</tbody>
</table>

For other guides, 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 details on 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 Oracle Enterprise Repository at [http://fusionappsoer.oracle.com](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.

- Other technical information such as reusable components, policies, architecture diagrams, and topology diagrams.

**Documentation Accessibility**


**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 **Send Feedback to Oracle** from the Settings and Actions menu in Oracle Fusion Applications Help.
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

Each formula usage corresponds to one or more formula types, requiring specific formula inputs and outputs. These requirements are explained in separate chapters of the Oracle Fusion Fast Formula Guide.

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>Then</td>
<td>SELECT_EMP</td>
<td>Character</td>
<td>=</td>
<td>'YES'</td>
</tr>
<tr>
<td>ELSE</td>
<td>SELECT_EMP</td>
<td>Character</td>
<td>=</td>
<td>'NO'</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formula Name</td>
<td>Manager Range of Scheduled Hours</td>
</tr>
<tr>
<td>Formula Type</td>
<td>Range of Scheduled Hours</td>
</tr>
<tr>
<td>Description</td>
<td>Manager's Range of Hours</td>
</tr>
<tr>
<td>Effective Start Date</td>
<td>1-Jan-2010</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
  (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:

```sql
SALARY = SALARY_ANNUAL_SALARY / 12
RETURN SALARY
```

An example of efficient use of INPUTS statements is:

```sql
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:

```sql
S = SALARY
A = AGE
IF S < 20000 THEN
  IF A < 20 THEN
    TRAINING_ALLOWANCE = 30
```
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 <code>IF1</code> instead of <code>IF</code> for an <code>IF</code> statement.</td>
</tr>
<tr>
<td>Incorrect Statement Order</td>
<td><code>ALIAS</code>, <code>DEFAULT</code>, or <code>INPUT</code> statements come after other statements.</td>
</tr>
<tr>
<td>Misuse of <code>ASSIGNMENT</code> Statement</td>
<td>Occurs when any of these conditions occurs:</td>
</tr>
<tr>
<td></td>
<td>• An <code>ASSIGNMENT</code> assigns a value to a database item.</td>
</tr>
<tr>
<td></td>
<td>• A context is assigned a value externally to a <code>CHANGE-CONTEXTS</code> statement.</td>
</tr>
<tr>
<td></td>
<td>• A non-context variable is assigned a value within a <code>CHANGE-CONTEXTS</code> statement.</td>
</tr>
<tr>
<td>Misuse of <code>ALIAS</code> Statement</td>
<td>An <code>ALIAS</code> statement may only be used for a database item.</td>
</tr>
<tr>
<td>Missing <code>DEFAULT</code> Statement</td>
<td>A database item with defaulting specified must have a <code>DEFAULT</code> statement.</td>
</tr>
<tr>
<td>Misuse of <strong>DEFAULT</strong> Statement</td>
<td>A <strong>DEFAULT</strong> statement is specified for a variable other than an input or database item.</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</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 <strong>NUMBER</strong>, or both of data type <strong>TEXT</strong>.</td>
</tr>
<tr>
<td>Inconsistent Data Type Usage</td>
<td>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 <strong>NUMBER</strong> value at the start of the formula, but a <strong>TEXT</strong> value later in the formula.</td>
</tr>
<tr>
<td><strong>EXIT</strong> Statement Not Within <strong>WHILE</strong> Loop</td>
<td>A condition that eventually becomes false, or an <strong>EXIT</strong> call for exiting the loop does not exist.</td>
</tr>
<tr>
<td>Misuse of Context</td>
<td>A variable is used as a context, or a context is used as a variable. For example, <strong>AREA1</strong> is assigned a value as an ordinary variable, but later in the formula <strong>AREA1</strong> is used as a context in a <strong>GET_CONTEXT</strong> call.</td>
</tr>
</tbody>
</table>

**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>Uninitialized Variable</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>Divide by Zero</td>
<td>Raised when a numeric value is divided by zero.</td>
</tr>
<tr>
<td>Error Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>No Data Found</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.</td>
</tr>
<tr>
<td></td>
<td>This error is also raised from within a formula function. The cause is an error in the formula function code.</td>
</tr>
<tr>
<td>Too Many Rows</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.</td>
</tr>
<tr>
<td></td>
<td>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>NULL Data Found</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>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 items 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>--------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------</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 * 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 \textit{WAGE} value, the formula needs to get the value for the variable \textit{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, \textit{ANNUAL_LEAVE}.

/* Formula: Annual Leave Formula */
IF YEARS_SERVICE \geq 10
THEN
ANNUAL_LEAVE = 25
ELSE
ANNUAL_LEAVE = 20 + \text{FLOOR} \left(\frac{\text{YEARS_SERVICE}}{2}\right)
RETURN ANNUAL_LEAVE

Input Statements

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

INPUTS ARE HOURS_WORKED
WAGE = HOURS_WORKED \times \text{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 \textit{HOURS_WORKED} is numeric. If the input variable is not numeric, you must specify the type. For example,

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 \textit{NUMBER} data type. The format of an expression is:

\text{SUBEXPRESSION} \ [\text{operator} \ \text{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,

```sql
\text{TRAINING\_ALLOWANCE} = 0
\text{IF} \ (\text{AGE} < 20) \ \text{THEN}
\text{TRAINING\_ALLOWANCE} = 30
```

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:

```sql
/* Use this formula to determine the bonus percentage for staff */
\text{INPUTS ARE} \ \text{SALARY\_AMOUNT,}
\text{START\_DATE} \ \text{(DATE),}
\text{END\_PERIOD\_DATE} \ \text{(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>
<tbody>
<tr>
<td>ALIAS</td>
<td>ALIAS name1 AS name2</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>ALIAS OVERTIME_QUALIFYING_LENGTH_OF_AS OT_QLS</td>
<td></td>
</tr>
<tr>
<td>ASSIGNMENT</td>
<td>variable = expression</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>array[index] = expression</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RATE = HOURLY_RATE + 14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WAGE = HOURS_WORKED * RATE</td>
<td></td>
</tr>
</tbody>
</table>
| CHANGE-CONTEXTS | CHANGE CONTEXTS (context1 = expression1 [context2 = expression2] ...  
| Change contexts | (context1 = expression1 [context2 = expression2] ...  
| Allows one or more contexts to be changed within a formula. The new values are assigned by one or more ASSIGNMENT statements.  
| DEFAULT | DEFAULT FOR variable IS literal  
| DEFAULT | DEFAULT DATA VALUE FOR variable IS literal  
| DEFAULT | DEFAULT FOR HOURLY_RATE IS 3.00  
| DEFAULT | INPUTS ARE HOURLY_RATE  
| DEFAULT | X = HOURS_WORKED * HOURLY_RATE  
| DEFAULT | 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 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.  
| DEFAULT | The DEFAULT_DATA_VALUE FOR statement allows a value to be used for an array database item where individual data values are NULL.  
| DEFAULT | Some database items are defined to require a default value because they could return no data or NULL values from the database.  
| EXIT | EXIT  
| EXIT | FOUND = -1 /* -1 is not a valid index for A. */  
| EXIT | I = A.FIRST(-1)  
| EXIT | WHILE (A.EXISTS(I)) LOOP  
| EXIT | /* EXIT-clause for early exit. */  
| EXIT | IF A[I] = KEY THEN  
| EXIT | (FOUND = I  
| EXIT | /* Exit the loop. */  
| EXIT | EXIT;  
| EXIT | )  
| EXIT | I = A.NEXT(I,-1)  
| EXIT | Used to immediately exit from the enclosing WHILE loop. The EXIT statement cannot be used outside of a WHILE loop.  
|
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 (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.

```
INPUT
```

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

```
RETURN
```

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
```

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.

```
FOR
```

The `FOR` loop is used to execute a set of statements a specified number of times. The loop counter is incremented after each iteration until the loop counter exceeds the specified number of times. The `FOR` loop is similar to the `WHILE` loop, but it is more suitable for loops that have a known number of iterations.

```
REPEAT
```

The `REPEAT` loop is similar to the `FOR` loop in that it executes a set of statements a specified number of times. However, the `REPEAT` loop is more flexible and allows for more control over the loop counter. The loop counter is incremented after each iteration until the loop counter exceeds the specified number of times.
WORKING STORAGE

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKING_STORAGE</td>
<td>Use the working storage area statements to store reference data.</td>
</tr>
<tr>
<td>WSA_DELETE([item])</td>
<td>Deletes values from the storage area.</td>
</tr>
<tr>
<td>WSA_EXISTS(item[,type])</td>
<td>Determine if an item exists.</td>
</tr>
<tr>
<td>WSA_GET(item, value)</td>
<td>Fetches values from the storage area.</td>
</tr>
<tr>
<td>WSA_SET(item, value)</td>
<td>Sets values from the storage area.</td>
</tr>
</tbody>
</table>

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

/* 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. */

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:

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

Incorrect example:

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
```
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.

- **Flexfields**
  The Generate Flexfield Database Items process creates database items for the contexts and segments of your registered HCM flexfields.

**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
ADELETE(1)
```
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>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:

- '2010-11-04T00:00:00Z' (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)

**Generating Flexfield Database Items: Explained**

You configure registered HCM flexfields to add contexts and segments for your business requirements. After you deploy the flexfield, you can generate database items for the flexfield for use in your formulas and extracts by submitting the Generate Flexfield Database Items process from the Payroll Checklist or Payroll Calculation work areas.

You can generate DBIs for the following flexfields:

• Descriptive flexfields
• Extensible flexfields for single and multiple row routes
• Key flexfields

The process generates DBIs at the enterprise level only. As a best practice, when you submit the process, skip the legislative data group parameter so that the process generates DBIs for use by any legislative data group.

You can determine which DBIs to generate by specifying or skipping the flexfield and context parameters.

<table>
<thead>
<tr>
<th>Flexfield Parameter</th>
<th>Context Parameter</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify parameter</td>
<td>Skip parameter</td>
<td>Generate DBIs for all the contexts and related segments for a specified flexfield</td>
</tr>
<tr>
<td>Skip parameter</td>
<td>Skip parameter</td>
<td>Generate DBIs for all registered flexfields and their contexts.</td>
</tr>
</tbody>
</table>
The process creates database item names with this following structure:

\[ \text{<FLEXFIELD_CODE><CONTEXT_CODE><SEGMENT_CODE>} \]

When you include the database item in a formula or extract, the application returns a value for the database item, based on the flexfield context, for the segments column in the underlying flexfield table. After you generate DBIs, compile any formulas using these DBIs.

Periodically, you may need to update a flexfield structure, for example to add a segment to capture additional data. If you previously generated DBIs for a flexfield, submitting the process deletes and regenerates its DBIs. After the process regenerates the DBIs, be sure to compile any formulas using them.

**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:

   \[ \text{INPUTS ARE SALARY\_AMOUNT,} \]
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. */

Generating Flexfield Database Items: Explained

You configure registered HCM flexfields to add contexts and segments for your business requirements. After you deploy the flexfield, you can generate database items for the flexfield for use in your formulas and extracts by submitting the Generate Flexfield Database Items process from the Payroll Checklist or Payroll Calculation work areas.

You can generate DBIs for the following flexfields:

- Descriptive flexfields
- Extensible flexfields for single and multiple row routes
- Key flexfields

The process generates DBIs at the enterprise level only. As a best practice, when you submit the process, skip the legislative data group parameter so that the process generates DBIs for use by any legislative data group.

You can determine which DBIs to generate by specifying or skipping the flexfield and context parameters.
<table>
<thead>
<tr>
<th>Flexfield Parameter</th>
<th>Context Parameter</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify parameter</td>
<td>Skip parameter</td>
<td>Generate DBIs for all the contexts and related segments for a specified flexfield</td>
</tr>
<tr>
<td>Skip parameter</td>
<td>Skip parameter</td>
<td>Generate DBIs for all registered flexfields and their contexts.</td>
</tr>
</tbody>
</table>

The process creates database item names with this following structure:

<FLEXFIELD_CODE><CONTEXT_CODE><SEGMENT_CODE>

When you include the database item in a formula or extract, the application returns a value for the database item, based on the flexfield context, for the segments column in the underlying flexfield table. After you generate DBIs, compile any formulas using these DBIs.

Periodically, you may need to update a flexfield structure, for example to add a segment to capture additional data. If you previously generated DBIs for a flexfield, submitting the process deletes and regenerates its DBIs. After the process regenerates the DBIs, be sure to compile any formulas using them.

**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>&lt;name&gt;[&lt;index value&gt;]</td>
<td>Get the value for an index.</td>
<td>V = A[1]</td>
</tr>
<tr>
<td>&lt;name&gt;.FIRST(&lt;default value&gt;)</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>&lt;name&gt;.LAST(&lt;default value&gt;)</td>
<td>Get the last index for an array.</td>
<td>L = B.LAST(&quot;&quot;)</td>
</tr>
<tr>
<td>&lt;name&gt;.EXISTS(&lt;index value&gt;)</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>&lt;name&gt;.NEXT(&lt;index value&gt;, &lt;default index value&gt;)</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>&lt;name&gt;.PRIOR(&lt;index value&gt;, &lt;default index value&gt;)</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>&lt;name&gt;, COUNT</td>
<td>Numeric method to count the array elements.</td>
<td>C = A.COUNT</td>
</tr>
<tr>
<td>&lt;name&gt;, DELETE(&lt;index value&gt;)</td>
<td>Delete the element at an index position.</td>
<td>B.DELETE('three')</td>
</tr>
<tr>
<td>&lt;name&gt;, DELETE()</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.NEXXT(TI, 'No Index') /* Go to next index. */
END LOOP
```

The following example iterates backwards 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. */
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 = DBII
/* Overriding a Context change. */
CHANGE_CONTEXTS({SOURCE_TEXT = 'B', SOURCE_ID = 3})
{ /* SOURCE_TEXT = 'B', SOURCE_ID = 3 */
  Z = DBII
}
}

• 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

• WSA_SET

• WSA_GET

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>
</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. There are two alternative approached to calling a formula: using a single call, or separate calls.

Consider the following aspects:

- **Validation of the Called Formula**
- **Passing Contexts**
- **Alternative Methods to Call a Formula**
  - Using Separate Calls
  - Using a Single Self-Contained Call
  - Use Cases to Compare Methods

### Validation of the Called Formula

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. You can use the `IS_EXECUTABLE` call to determine whether an executable formula with a specified name exists. The formula must be compiled and available for the specified legislative data group. Also, it must be valid as of the effective date of calling formula execution. Payroll code imposes extra restrictions based on formula type combinations.

### Passing Contexts

Context values are inherited from the calling formula. You can also set or unset the context values explicitly in the nested formula call.

### Alternative Methods to Call a Formula

There are two ways to call a formula from a formula:

- Using a series of separate calls
- Using a single self-contained call

### Using Separate Calls

If you are using separate calls, there are three stages in calling a formula:

<table>
<thead>
<tr>
<th>Formula Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WSA_GET(item, default-value)</code></td>
<td>Gets a value for the item called <code>item</code>. If there is no item called <code>item</code> then <code>default-value</code> is returned. The data type of <code>default-value</code> is the expected data type for <code>item</code>.</td>
</tr>
</tbody>
</table>
1. Set the Inputs

Use a `SET_INPUT` call for each formula input and context that you need to explicitly set for the formula call. You don’t need to specify all formula inputs and contexts. To explicitly unset a context values, use the `SET_INPUT` call without passing the optional value parameter. Any extra inputs specified in `SET_INPUT` calls are ignored.

2. Call the Formula

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

3. 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.

The following table summarizes the methods for calling a formula using separate calls.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SET_INPUT(input [,value])</code></td>
<td>The value parameter is optional. If it’s provided, the specified input is set to this value. If it’s not provided, the If a value is provided, the input is passed as unset to the formula. The data type of the value is the expected data type for the input.</td>
</tr>
<tr>
<td><code>EXECUTE(formula)</code></td>
<td>Executes the called formula.</td>
</tr>
<tr>
<td><code>GET_OUTPUT(output, default-value)</code></td>
<td>Gets the value of the output parameter after calling a formula. If there’s no formula output called ‘output’ or it’s not set, then the value specified in the default value parameter is returned. The data type of default value is the expected data type for output.</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.

**Using a Single-Self Contained Call**

The end result with this approach is the same as using separate calls except that:

- Input values are cleared at the start so that prior `SET_INPUT` call values are not used.

- Outputs are discarded at the end so that subsequent `GET_OUTPUT` calls just return the default values.

Use the `CALL_FORMULA` method as follows:
CALL_FORMULA(formula, [set statement, get statement])

A SET statement is a SET_INPUT call. A GET statement assigns a GET_OUTPUT call result to a variable in the calling formula. The execution order is:

1. SET_INPUT calls
2. EXECUTE call
3. GET_OUTPUT assignments

The compiler generates code to execute in this order even if SET and GET statements are interspersed.

**Use Cases to Compare Methods**

The following table describes complex expressions used for called formula validation names, setting input values, and providing default output values.

**Note**

SET_INPUT or > statements have no effect if the calling formula has no formula input or context of the same name.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Using Separate Calls</th>
<th>Using a Self-Contained Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute a formula where the formula GET_RATES is executed</td>
<td>EXECUTE('GET_RATES')</td>
<td>Use within a CALL_FORMULA statement 'GET_RATES'</td>
</tr>
<tr>
<td>Set an input value in the called formula where you round up EXTRA_HOURS to 2 decimal places and set the input OVERTIME in the called formula. The called formula should contain the statement: INPUTS ARE OVERTIME</td>
<td>SET_INPUT('OVERTIME',ROUNDUP(EXTRA_HOURS,2))</td>
<td>Use within a CALL_FORMULA statement ROUNDUP(EXTRA_HOURS,2) &gt;'OVERTIME'</td>
</tr>
<tr>
<td>Leave a formula input value unset inside the called formula, where RATE is not a formula context.</td>
<td>A SET_INPUTS statement is not required, but the following can be used: SET_INPUT('RATE')</td>
<td>A SET statement is not required, but the following can be used: &gt; 'RATE'</td>
</tr>
<tr>
<td>Inherit a context value from the called formula. For example, both the calling and called formula support the AREA1 context. We inherit the AREA1 context value from the calling formula in the calling formula.</td>
<td>No statements are required to do this.</td>
<td>No statements are required to do this.</td>
</tr>
<tr>
<td>Set a context value inside a called formula, where the called formula supports the AREA1 context and AREA1 has to be set to 'London' in the called formula.</td>
<td>SET_INPUT('AREA1','London')</td>
<td>'London' &gt; 'AREA1'</td>
</tr>
</tbody>
</table>
Call a formula with an unset context value, where the called formula supports the AREA1 context and AREA1 has to be unset in the called formula.

| Set_INPUT('AREA1') | 'AREA1' |

Get a formula output from the called formula.

Get BONUS_RATE output value into the RATE variable using the default value 0.0 if the BONUS_RATE output does not exist or was not set.

| RATE = GET_OUTPUT('BONUS_RATE', 0.0) | RATE <'BONUS_RATE' DEFAULT 0.0 |

Get a formula output from a called formula into an array.

Get the BONUS_RATE output value into the RATES array variable at index position 'BONUS'. The default value 0.0 is used if the BONUS_RATE output does not exist or was not set.

| RATES['BONUS'] = GET_OUTPUT('BONUS_RATE', 0.0) | RATES['BONUS'] <'BONUS_RATE' DEFAULT 0.0 |

### Calling a Formula from a Formula: Examples

These examples illustrate a formula called from another formula as a series of separate calls or as a single self-contained call. Aspects in both examples are:

- The formula RATE_FORMULA is called to get a value for HOURLY_RATE.
- The RATE_FORMULA has a text input of UNITS.
- The UNIT input is set to 'Hourly' in the formula call.
- The RATE_FORMULA returns the rate in the output variable of RATE.
- The GET_OUTPUT call returns 0.0 if the RATE_FORMULA does not return RATE.

The following examples show modified versions of the wage formula.

#### Separate Calls

This example illustrates a formula call from separate calls.

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

#### Self-Contained Call

This example illustrates a formula called from a self-contained call.
CALL_FORMULA
('RATE_FORMULA','Hourly' > 'UNIT'
/* SET_INPUT('UNIT', 'Hourly') */
,HOURLY_RATE < 'RATE' DEFAULT 0.0
/* HOURLY_RATE = GET_OUTPUT('RATE',0.0) */
)
WAGE = HOURS_WORKED*HOURLY_RATE
RETURN RATE
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:

\[ \text{NAME-OF-FUNCTION}(\text{operand}, \text{operand}, \ldots) \]

Parameters can be optional or mandatory. They may be repeated any number of times, such as with the \text{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

\[ \text{Note} \]

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 n-th character for the m-th 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 n-th 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 'helloXYXYX' */
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','de')

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

SUBSTR 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:
FLOOR(35.455)
It returns 35.

GREATEST(n, n [, n] ...) or GREATEST_OF(n, n [, n] ...)
Compares all the operands and returns the largest value.

LEAST(n, n [, n] ...) or LEAST_OF(n, n [, n] ...)
Compares all the operands and returns the smallest value.

MOD(m, n)
Returns the remainder from dividing m by n.

POWER(m, n)
Returns m raised to the nth power.

ROUND(m [,n])
Rounds m to n decimal places. The default number of decimal places is 0.
Examples:
ROUND(2.3401, 2)
It returns 2.34.
ROUND (2.3461, 2)
It returns 2.35.

ROUNDUP(m [,n]) or ROUND_UP(m [,n])
Rounds m up to n decimal places. The default number of places is 0.
Examples:
ROUND_UP(2.3401, 2)
It returns 2.35.
ROUND_UP(2.3400, 2)
It returns 2.34.

TRUNC(n [,m]) or TRUNCATE(n [,m])
Truncates m down to n decimal places. The default number of places is 0.
Examples:
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.

```plaintext
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

A proration formula controls how the payroll run prorates an element entry when it encounters an event, such as a change to an element entry value, that you have defined as a proration event for the element. You can copy and edit a predefined proration formula if you want to customize the calculation, then select the custom formula as the proration formula for your element.

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.

Some predefined legislations supply proration formulas that you can use as the basis for your customized version.

User Table Validation Formula Type

The User Table Validation formula type validates entries in user-defined tables. Select the formula in the Formula field for user-defined columns when you create or edit user-defined tables.

For example, you can use this formula type to ensure that entries are:

- Between a specified range
- Not a negative amount
**Contexts**

The EFFECTIVE_DATE (text) context is used for formulas of this type.

**Input Variables**

There must be one input variable and it must be called ENTRY_VALUE. The data type is text.

**Return Values**

The following return values 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>FORMULA_MESSAGE</td>
<td>Text</td>
<td>N</td>
<td>Returns a text message for either or both statuses. The message is displayed on the Create User-Defined Table: User-Defined Table Values page.</td>
</tr>
<tr>
<td>FORMULA_STATUS</td>
<td>Text</td>
<td>Y</td>
<td>Returns the value S (success) or E (error).</td>
</tr>
</tbody>
</table>

**Sample Formula**

This formula checks that the deduction entered in the Union A column of the Union Dues table is between 10.00 and 20.00:

```sql
/* Formula Name: Union A Dues Validation */
/* Formula Type: User Table Validation */

INPUTS ARE entry_value (text)

IF TO_NUMBER(entry_value) < 10.00 OR TO_NUMBER(entry_value) > 20.00

THEN

  (formulas_status = 'e'
  formulas_message = 'Error: Union A dues must be between $10.00 and $20.00.'

) ELSE

  (formulas_status = 's'
  formulas_message = ''

) RETURN formulas_status, formulas_message
```

**Flow Schedule Formula Type**

The Flow Schedule formula controls when the application submits the current flow and how often it submits future instances of the flow. When you submit a flow, you select a formula to schedule the flow on the Schedule page.
Create a formula on the Manage Fast Formulas page when the predefined formulas do not cover your scheduling requirements. For example, you might create a formula to load time card batches daily, but increase the frequency to every four hours toward the end of a weekly payroll period when workers are more likely to submit their time cards.

Follow these tips when creating or updating a formula to schedule flows.

- Specify a meaningful name for it to help guide the person who selects the formula when submitting the flow.
- Review the formula to ensure it does not contain negative numbers that might produce a condition, such as running a process continually.
- After updating the formula, cancel any flows you scheduled that use the formula. Resubmit the flow to apply the updated definition.

**Contexts**

The SCHEDULED_DATE (scheduled date) context is available to formula of this type.

**Database Items**

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

<table>
<thead>
<tr>
<th>Database Item</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF_ADD_DAYS</td>
<td>Date</td>
<td>Function to add days to a date.</td>
</tr>
<tr>
<td>FF_ADD_MONTHS</td>
<td>Date</td>
<td>Function to add months to a date.</td>
</tr>
<tr>
<td>NEXT_SCHEDULED_DATE</td>
<td>Date</td>
<td>Calculated value for the date to schedule the next flow.</td>
</tr>
<tr>
<td>SCHEDULED_DATE</td>
<td>Date</td>
<td>Date used to schedule the flow.</td>
</tr>
</tbody>
</table>

**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>SCHEDULED_DATE(DATE)</td>
<td>Date</td>
<td>Y</td>
<td>Date on which to schedule the flow. The date is passed to the formula when it calculates the next date to schedule the flow.</td>
</tr>
</tbody>
</table>

**Return Values**

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

<table>
<thead>
<tr>
<th>Return Values</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXT_SCHEDULED_DATE</td>
<td>Date</td>
<td>Y</td>
<td>The date calculated by the formula to schedule the next flow.</td>
</tr>
</tbody>
</table>
**Sample Formula**

This sample formula is a predefined formula that schedules a flow so that the application submits it weekly from the date the person initially submitted it.

```
/
*****************************************************************************
FORMULA NAME: Weekly
FORMULA TYPE: Flow Schedule
DESCRIPTION: Formula to return a date time.
Returns NEXT_SCHEDULED_DATE;
Formula Results:
NEXT_SCHEDULED_DATE This is a date time value with yyyy-MM-dd HH:mm:ss format.
*****************************************************************************

/* Inputs */
INPUTS ARE SUBMISSION_DATE(DATE), SCHEDULED_DATE(DATE)

/* Calculations */
NEXT_SCHEDULED_DATE = ADD_DAYS(SCHEDULED_DATE,7)

/* Returns */
RETURN NEXT_SCHEDULED_DATE
/* End Formula Text */
```

You can calculate units smaller than a day by replacing the calculation portion of the formula text using a decimal or a fraction. This table shows examples of submitting a flow several times a day.

<table>
<thead>
<tr>
<th>Flow Submission</th>
<th>Formula Text for Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Twice a day</td>
<td>NEXT_SCHEDULED_DATE = ADD_DAYS(SCHEDULED_DATE, 0.5)</td>
</tr>
<tr>
<td>Hourly</td>
<td>NEXT_SCHEDULED_DATE = ADD_DAYS(SCHEDULED_DATE, 1/24)</td>
</tr>
</tbody>
</table>

**Creating a Daily Schedule for a Payroll Flow that Skips Weekends: Worked Example**

This example demonstrates how to create a formula that returns the next schedule date for a flow that is submitted daily on weekdays but not at the weekend.

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 for use by any legislative data group.</td>
</tr>
<tr>
<td>What is the formula type for this formula?</td>
<td>Flow Schedule</td>
</tr>
<tr>
<td>Are there any contexts used in this formula?</td>
<td>No</td>
</tr>
</tbody>
</table>
Are there any database item defaults? | No
---|---
Are there any input value defaults? | SUBMISSION_DATE, SCHEDULED_DATE
What are the return values? | NEXT_SCHEDULED_DATE

### Creating a Fast Formula to Submit a Flow Only on Weekdays

1. Open the Payroll Calculation work area, and click Manage Fast Formulas to open the Manage Fast Formulas page.
2. Click the Create icon to create a new formula.
3. 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>Daily Weekday Schedule</td>
</tr>
<tr>
<td>Formula Type</td>
<td>Flow Schedule</td>
</tr>
<tr>
<td>Description</td>
<td>Submits Flow Daily Except Weekends</td>
</tr>
<tr>
<td>Effective Start Date</td>
<td>1-Jan-2010</td>
</tr>
</tbody>
</table>

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

```plaintext
/*******************************************************************
FORMULA NAME: Daily Weekday Schedule
FORMULA TYPE: Flow Schedule
DESCRIPTION: Formula to return a date time.
Returns NEXT_SCHEDULED_DATE;
FORMULA RESULTS:
NEXT_SCHEDULED_DATE This is a date time value with yyyy-MM-dd HH:mm:ss format.
********************************************************************/
/* Inputs */
INPUTS ARE SCHEDULED_DATE(DATE)

/* Calculations */
add = 1
day = to_char(SCHEDULED_DATE, 'DAY')
if (day = 'FRIDAY') then add = 3
NEXT_SCHEDULED_DATE = ADD_DAYS(SCHEDULED_DATE, add)

/* Returns */
RETURN NEXT_SCHEDULED_DATE

/* End Formula Text */
```

6. Click Compile.
7. Click Save.
Qualification Absence Plan Rules: Points to Consider

Configure the following rules when you create an absence qualification plan in accordance with the leave policy of your enterprise:

- Plan term
- Plan eligibility
- Enrollment and termination
- Waiting period
- Payments

Plan Term
A plan term, in the context of an absence qualification plan, is an assessment period for which the Evaluate Absence process calculates entitlements for the total absent time recorded in that period. When you create an absence qualification plan, you must select the type of plan term. For example, you can limit the duration of the plan term to the duration of the absence.

Plan Eligibility
Associate an eligibility profile with the qualification plan to determine the set of workers who are eligible to record an absence that belongs to that plan. You create the eligibility profile on the Manage Eligibility Profiles task in the Setup and Maintenance work area. Then, you associate the eligibility profile with the absence plan using the Manage Absence Plans task. If you want all employees to be eligible for the absence plan, then do not add an eligibility profile. If you associate multiple absence plans with an absence type, the worker must be eligible for at least one absence plan to record an absence of that type.

Enrollment and Termination
Decide when to enroll workers in the qualification plan. Typically, you enroll workers in the plan when a worker or an administrator schedules an absence using an absence type that is associated with a qualification plan. Alternatively, you can use a formula if you must consider other aspects or rules that determine when to enroll workers in the plan.
Decide whether ongoing payments under this plan must continue if a worker is terminated. Decide whether ongoing payments under this plan must continue if a worker is not terminated, but loses eligibility for the plan.

**Waiting Period**

Define a waiting period if you want newly hired workers to be eligible for this plan only after a specific amount of time.

**Payments**

Use an entitlement band matrix to determine the payment percentages that apply for specific time periods during an absence. For example, you want workers who have completed between 5 and 10 years of service to receive 75 percent of pay for up to 10 days of absence. However, you want workers who have completed between 11 and 20 years to receive the same pay percentage, but for 20 days of absence.

Decide how you want to calculate the payment rate of a single unit of absence. You can use a rate definition to include the calculation rules, or use a formula.

### Accrual Absence Plan Rules: Points to Consider

Configure the following rules when you create an absence accrual plan in accordance with the leave policy of your enterprise:

- Accrual term and frequency
- Plan eligibility
- Enrollment and termination
- Waiting period and vesting period
- Plan limits
- Payments
- Adjustments

#### Accrual Term and Frequency

An accrual term is a period of time during which workers accrue time. You must specify the type of the accrual term to use for the plan. For example, you can define an accrual term of one calendar year that restarts on January 1, or an accrual term that starts on the worker’s annual hire date and restarts on every anniversary.

Use one of these methods to determine how workers accrue time in an accrual term:

- Award time in increments, also known as accrual periods, throughout an accrual term. Use the worker’s pay periods or define your own repeating periods to determine the number of accrual periods in a term. For example, workers who are paid monthly have 12 accrual periods in a year. The accrual amounts for each accrual period are automatically calculated based on the accrual rate, which is the total amount of time that you want the worker to accrue in an accrual term.
• Award time upfront at the beginning of each accrual term.

Plan Eligibility

Associate an eligibility profile with the accrual plan to determine the set of workers who can enroll in that plan. Use the Manage Eligibility Profiles task in the Absence Administration work area to create the eligibility profile. Then, you associate the eligibility profile with the absence plan using the Manage Absence Plans task. If you want all employees to be eligible for the absence plan, then do not add an eligibility profile. If you associate multiple absence plans with an absence type, the worker must be eligible for at least one absence plan to record an absence of that type.

Enrollment and Termination

Decide when to enroll workers in the accrual plan. You can enroll workers in the plan when the new-hire event or transfer event occurs. Alternatively, you can use a formula if you want to consider other aspects or rules to determine when to enroll workers.

When a worker is terminated, choose when to disenroll the worker from the plan. Choose how you want to deal with negative and positive balances in situations where only plan enrollment ends, or both plan enrollment and employment ends.

Waiting Period and Vesting Period

Define a waiting period if you want newly enrolled workers to start accruing time under that plan only after a specific amount of time elapses after the date of enrollment. Define a vesting period if you want newly enrolled workers to accrue time, but not use it until after a specific amount of time.

Plan Limits

Configure the following plan limits:

• Accrual rate: Determines how much time a worker can accrue in an accrual term.

• Carryover limit: Determines the maximum time that workers can carry over to the next term.

• Ceiling: Determines the maximum leave time that workers can accrue.

Use an accrual band matrix to build criteria using various factors, such as length of service, to determine the set of eligible workers who qualify for specific plan limits. Alternatively, you can use a formula to determine each plan limit.

Payments

Decide how you want to calculate payment of accrual balances for the following scenarios:

• When workers must be paid a different rate during the absence period

• When a part of the accrual balance must be disbursed to workers as cash

• When the cost of accrual balance must be calculated to determine employer liability
• When the accrual balance must be paid to workers when their plan participation ends

**Adjustments**

You can enable the following types of adjustments that HR specialists can make during maintenance of absence records and entitlements:

• Accrual transfer from previous employment
• Used time transfer from previous employment
• Discretionary disbursements of accrual balance
• Accrual balance transfers across plans

**Formulas for Accrual Plan Rules: Explained**

Although you can incorporate accrual plan rules in the Manage Absence Plan page, you can write your own formulas if you want to incorporate other special rules to suit your requirement.

**Formulas for Accrual Plan Rules**

The following table lists the aspects of an accrual plan for which you can write a formula and identifies the formula type for each.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Formula Type to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enrollment Start</td>
<td>When eligible workers can be enrolled in the plan.</td>
<td>Global Absence Plan Enrollment Start</td>
</tr>
<tr>
<td>Enrollment End</td>
<td>Date when workers disenroll from the plan.</td>
<td>Global Absence Plan Enrollment End</td>
</tr>
<tr>
<td>Plan Duration Conversion</td>
<td>Method to calculate the absence duration differently. For example, you might have a requirement to consider only whole working days in a vacation absence to update the accrual balance. In such cases, you define logic in a formula to convert the absence duration to a value that excludes partial days.</td>
<td>Global Absence Plan Duration</td>
</tr>
<tr>
<td>Anniversary Event Rule</td>
<td>Method to determine the employment anniversary date on which you want the accrual plan to restart.</td>
<td>Global Absence Plan Period Anniversary Event Date</td>
</tr>
<tr>
<td>Accrual Vesting</td>
<td>A period during which workers accrue time, but cannot use it.</td>
<td>Global Absence Vesting</td>
</tr>
<tr>
<td>Accrual Proration</td>
<td>Calculation method to determine how much time workers accrue if they enroll in the middle of an accrual period.</td>
<td>Global Absence Proration</td>
</tr>
<tr>
<td>Ceiling</td>
<td>The maximum time that a worker can accrue.</td>
<td>Global Absence Ceiling</td>
</tr>
<tr>
<td><strong>Carryover</strong></td>
<td>The maximum unused time that a worker can transfer to the next accrual term.</td>
<td>Global Absence Carryover</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><strong>Accrual Band Formula</strong></td>
<td>Range of eligibility criteria that identify how much paid time eligible workers accrue over the course of an accrual term. The criteria may be years of service, grades, hours worked, or some other factor that you can define.</td>
<td>Global Absence Accrual</td>
</tr>
<tr>
<td><strong>Absence Payment Rate</strong></td>
<td>Method to calculate payment during absence period.</td>
<td>Global Absence Plan Use Base Rate</td>
</tr>
<tr>
<td><strong>Disbursement Rate</strong></td>
<td>Method to calculate payment when paying out part of the accrual balance.</td>
<td>Global Absence Plan Use Cash Out Rate</td>
</tr>
<tr>
<td><strong>Final Balance Payment Rate</strong></td>
<td>Method to calculate payment of accruals when plan participation ends.</td>
<td>Global Absence Plan Use Payout Rate</td>
</tr>
<tr>
<td><strong>Liability Booking Rate</strong></td>
<td>Method to calculate cost of accrual balance to determine employer liability</td>
<td>Global Absence Plan Use Leave Liability Rate</td>
</tr>
</tbody>
</table>

### Formulas for Absence Qualification Plan Rules: Explained

Although you can incorporate qualification plan rules in the Manage Absence Plan pages, you can write your own formulas if you want to incorporate other special rules to suit your requirement.

#### Formulas for Qualification Plan Rules

The following table lists the aspects of a qualification plan for which you can write a formula.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Formula Type to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rolling Backward Start Rule</td>
<td>When the rolling backward plan term starts. A rolling backward term is a specific time period that precedes the absence start date.</td>
<td>Global Absence Roll Backward Start Date</td>
</tr>
<tr>
<td>Enrollment Start</td>
<td>When eligible workers can be enrolled in the plan</td>
<td>Global Absence Plan Enrollment Start</td>
</tr>
<tr>
<td>Enrollment End</td>
<td>When workers are disenrolled from the plan</td>
<td>Global Absence Plan Enrollment End</td>
</tr>
<tr>
<td>Plan Duration Conversion</td>
<td>Method to calculate the absence duration differently. For example, you want to consider only whole working days in a sickness absence in the entitlement calculation. In such cases, you define logic in a formula to convert the absence duration to a value that excludes partial working days.</td>
<td>Global Absence Plan Duration Conversion</td>
</tr>
<tr>
<td>Rule</td>
<td>Description</td>
<td>Formula Type to Use</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>Entitlement Definition</td>
<td>Determines payment percentages to apply during the absence period</td>
<td>Global Absence Entitlement</td>
</tr>
<tr>
<td>Qualification Band</td>
<td>A level that determines the payment that workers receive for a specific number of days during a long leave of absence based on their length of service.</td>
<td>Global Absence Entitlement</td>
</tr>
</tbody>
</table>

**Formulas for Absence Type Rules: Explained**

Use the Manage Absence Types pages to define absence type rules. However, if you want to define other special rules to suit your requirement, you can write your own formulas.

**Formulas for Absence Types**

The following table lists the aspects of an absence type for which you can write a formula and identifies the formula type for each.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
<th>Formula Type to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration Conversion</td>
<td>Method to convert the absence duration to other units of measure. For example, your workers’ work schedules are in work hours, but you want to display the duration in work days.</td>
<td>Global Absence Type Duration</td>
</tr>
<tr>
<td></td>
<td>You can use the formula to convert absence duration values that are in work days or work hours only.</td>
<td></td>
</tr>
<tr>
<td>Validation</td>
<td>Rules in addition to the ones that you can define on the Manage Absence Types pages to check the validity of the absence.</td>
<td>Global Absence Entry Validation</td>
</tr>
</tbody>
</table>
Formulas for Compensation Plans

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 related to Person, Assignment, Salary, and Element Entries are 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.

```
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. When you configure the worksheet display for a column in the Configure Column Properties dialog box, Default Values tab, you can select this formula.

The following predefined formulas are available for the eligible salary column for this formula type.

**Note**

Use these formulas as samples for testing purposes only. Copy and create your own version of a formula for use in your own compensation plans. Modifying the sample formula might provide unexpected results upon upgrade.

<table>
<thead>
<tr>
<th>Formula</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMP_ELIGIBLE_SALARY_PRORATION_DAILY_AVERAGE</td>
<td>Eligible salary calculated by averaging daily salary. Accounts for number of days that a salary is in effect during the workforce compensation cycle evaluation period.</td>
</tr>
<tr>
<td>Formula</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CMP_ELIGIBLE_SALARY_PRORATION_MONTH_END_AVERAGE</td>
<td>Eligible salary calculated by averaging salary on the last day of each month in the workforce compensation cycle evaluation period. Uses salary on the last day of the evaluation period for mid-month evaluation end dates.</td>
</tr>
<tr>
<td>CMP_ELIGIBLE_SALARY_PRORATION_DAILY_AVERAGE_NINETY_DAY_MIN</td>
<td>Eligible salary calculated by averaging daily salary. Accounts for number of days that a salary is in effect during the workforce compensation cycle evaluation period. Returns zero for workers who worked fewer than 90 days.</td>
</tr>
<tr>
<td>CMP_ELIGIBLE_SALARY_PRORATION_DAILY_AVERAGE_USING_FTE</td>
<td>Eligible salary calculated by averaging daily salary adjusted for part-time workers. Accounts for number days that a salary is in effect and FTE during the workforce compensation cycle evaluation period.</td>
</tr>
<tr>
<td>CMP_ELIGIBLE_SALARY_PRORATION_DAILY_AVERAGE_FOR_JOBS</td>
<td>Eligible salary calculated by averaging salary for the number of days worker holds a specific job code on the assignment. Accounts for the number of days that a salary is in effect during the workforce compensation cycle evaluation period.</td>
</tr>
</tbody>
</table>

**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 related to Person, Assignment, Salary, and Element Entries are 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>Unique numeric identifier for the workforce compensation plan</td>
</tr>
<tr>
<td>CMP_IV_PERIOD_ID</td>
<td>Number</td>
<td>Y</td>
<td>Unique numeric identifier for the fiscal calendar period</td>
</tr>
<tr>
<td>CMP_IV_COMPONENT_ID</td>
<td>Number</td>
<td>Y</td>
<td>Unique numeric identifier for the workforce compensation plan component</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. */

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

/* ============= FORMULA SECTION BEGIN ===============*/

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
```

---

7-4 Oracle Global Human Resources Cloud Using Fast Formula
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 related to Person, Assignment, Salary, and Element Entries are 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_D</td>
<td>Date</td>
<td>Y</td>
<td>Plan Start Date</td>
</tr>
<tr>
<td>CMP_IV_PLAN_END_D</td>
<td>Date</td>
<td>Y</td>
<td>Plan End Date</td>
</tr>
</tbody>
</table>

**Return Values**

Use the following predefined names for return variables, which 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>
Table 5-3: Return Value, Data Type, Required, Description

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_PERSON_NUMBER</td>
<td>Number</td>
<td>Y</td>
<td>Person number of manager</td>
</tr>
</tbody>
</table>

If the formula returns an invalid PERSON_NUMBER and the ASSIGNMENT_ID cannot be obtained, the following error is returned:

Formula passed in an invalid person number <15465857>. Assignment ID could not be obtained.

**Sample Formula**

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

```/*-----------------------------------------------*/
FORMULA NAME : Compensation Hierarchy Determination Formula
FORMULA TYPE : Compensation Hierarchy Determination
DESCRIPTION : Hierarchy determination fast formula which is based on assignment_id
*/

/*---------- 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)
END IF

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 related to Person, Assignment, Salary, and Element Entries are 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>
</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_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.

```c
/*******************************************************************
FORMULA NAME : Compensation Selection Formula
FORMULA TYPE : Compensation Person Selection
DESCRIPTION : Assignment_id based selection fast formula
*******************************************************************/

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

/*================ FORMULA SECTION BEGIN =======================*/
DEFAULT FOR CMP_IV_ASSIGNMENT_ID IS 0
l_selected = 'Y'

/* 100000008154095 - Ariel.Aimar@oracle.com - GBI data*/
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 related to Person, Assignment, Salary, and Element Entries are 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_STARTDATE</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.

/*********************************************************
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 '2012/01/01'
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 related to Person, Assignment, Salary, and Element Entries are 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>
</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

l_date = ADD_MONTHS(TO_DATE(CMP_IV_EFFECTIVE_DATE, 'YYYY/MM/DD'),6)
compensation_end_date = TO_CHAR(ldate, 'YYYY/MM/DD')

RETURN compensation_end_date
```

**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 related to Person, Assignment, Salary, and Element Entries are available to formulas of this 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_DATE</td>
<td>Date</td>
<td>Y</td>
<td>Statement Period Start Date</td>
</tr>
<tr>
<td>CMP_IV_PERIOD_END_DATE</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>COMPENSATIONDATES</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>
<tr>
<td>LEGALEMPLOYERS</td>
<td>Char</td>
<td>N</td>
<td>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 (; ;).</td>
</tr>
<tr>
<td>COMPENSATIONDATES1</td>
<td>Date</td>
<td>Y</td>
<td>Second variable for transaction dates from 16 to 30 if limit of 250 characters is exceeded.</td>
</tr>
<tr>
<td>VALUES1</td>
<td>Char</td>
<td>Y</td>
<td>Second variable for transaction values from 16 to 30 if limit of 250 characters is exceeded.</td>
</tr>
</tbody>
</table>
**Sample Formula**

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

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

```plaintext
/*========== INPUT VALUES DEFAULTS BEGIN ===============*/
INPUTS ARE CMP_IV_PERSON_ID (text), CMP_IV_PERIOD_START_DATE (date),
CMP_IV_PERIOD_END_DATE (date)
DEFAULT FOR CMP_IV_PERSON_ID IS '-1'
DEFAULT FOR CMP_IV_PERIOD_START_DATE IS '4712/12/31' (date)
DEFAULT FOR CMP_IV_PERIOD_END_DATE IS '4712/12/31' (date)
/*========== INPUT VALUES DEFAULTS ENDS ================*/

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

COMPENSATION_DATES = '2005/01/01'
VALUES = '500.00'

RETURN COMPENSATION_DATES, VALUES

/*--------------- FORMULA SECTION END ================*/
```

This sample formula returns multiple variables.

```plaintext
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;1234567890'
/* Returns first and last assignments */
ASSIGNMENTS3 = '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 type
A grouping of absences, such as illness or personal business, that are handled together for reporting, accrual, and compensation calculations.

accrual absence plan
A benefit that entitles workers to accrue time for the purpose of taking leave.

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.

database item
An item of information in Fusion HCM that has special programming attached, enabling it to be located and retrieved for use in formulas.

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.

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.

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

A benefit that entitles workers to paid leave time as a result of an event, such as childbirth, illness, or injury.

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.