IBM TRIRIGA Application Platform

Version 3 Release 7

Application Building for the IBM TRIRIGA Application Platform 3: Calculations

This edition applies to version 3, release 7 of IBM® TRIRIGA® Application Platform and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About This Guide

This document is part of the *Application Building for the IBM TRIRIGA Application Platform 3* collection of user guides. The collection is intended to provide you with an understanding of the basic tools to build or customize applications running on the IBM TRIRIGA Application Platform.

Conventions

This document uses the following conventions to ensure that it is as easy to read and understand as possible:

The phrase *Application Building* is used as an abbreviation for *Application Building for the IBM TRIRIGA Application Platform 3*.

Note - A Note provides important information that you should know in addition to the standard details. Often, notes are used to make you aware of the results of actions.

Tip - A Tip adds insightful information that may help you use the system better.

Attention - An Attention notice indicates the possibility of damage to a program, device, system, or data.

Intended Audience

This book is intended for those who are building or customizing an application that runs on the IBM TRIRIGA Application Platform. This document assumes familiarity with and competence in the information in the *Application Building for the IBM TRIRIGA Application Platform 3* collection of user guides.

Prerequisites

This document assumes the reader has a basic understanding of Web-based applications.

Support

IBM Software Support provides assistance with product defects, answering FAQs, and performing rediscovery. View the IBM Software Support site at www.ibm.com/support.

1. Extended Formulas

Extended formulas are a more flexible and powerful mechanism for computing values than the formula mechanism used to make simple computations based on fields in the same record that is presented in the “Data Types” chapter of *Application Building*.

Extended formulas can rely on data that is external to the record’s own business object and can perform more advanced operations than just add, subtract, multiply, and divide. The system supports the following extended formula types:

- Single-Hop Query: The query token refers to a query with a `$$RECORDID$$` association filter.
- Single-Hop Association Token: The association token refers to a field on an associated business object.
- Multi-Hop Query: The query token refers to a query with an association filter referencing another query. This can reference queries n-levels deep until reaching a query with a `$$RECORDID$$` association filter.
- Multi-Business Object Query: The query token refers to a multiple business objects where a `$$RECORDID$$` field filter is against one of the query’s associated (non-primary) business objects.
- Multi-Hop Association Token: The association token refers to a field that is in a business object associated n-levels away from the business object in which the extended formula resides.

Extended formulas are available in the Data Modeler for fields. They also are available in the Workflow Builder for object maps and as part of Workflow Condition Builder criteria. At workflow runtime, a workflow field map does not map a value from the source record to the destination record when the target field has a formula defined in its Business Object Field definition. The target field value is calculated and derived from its Data Modeler formula mapping. If a formula field is grayed out on the Object Mapping form, it means that the formula map defined in the Data Modeler is used as the source value for this field. Note that a total field that adds up a sum field in a smart section is also considered a formula field for the purposes of this behavior. To have mapping executed instead of the mapping defined in the Data Modeler, you must remove the formula mapping defined for this field from the Data Modeler; then workflow mapping is applied to the field.

In the Workflow Builder, extended formulas can access fields in the source record of the object map. More information on object mapping can be found in the “Creating Workflows” chapter of *Application Building*. Extended formulas are available for fields of any data type.

Extended formulas can be used in the Workflow Condition Builder as criteria for conditions specifying when a workflow needs to decide which path to take, for example in Start conditions, Switch tasks, and Break tasks. More information on the Workflow Condition Builder can be found in the “Workflow Tasks” section of the “Creating Workflows” chapter of *Application Building*.

An extended formula is always calculated for a particular record. The system determines the active project for the extended formula calculation as follows:

- If the record that the formula is being calculated for is a capital project, the active project for that extended formula calculation is that record itself.
- If the formula is being calculated in a capital project record, the active project for that extended formula calculation is that record itself.
- Whether or not the active project is used in the extended formula calculation is dictated by the data scope on the query definition of the extended formula’s individual query tokens. If the data scope of the query definition is Active Project, the records returned by the query for the calculation will be restricted to the active project as defined above. If the data scope of the query definition is not Active Project, the active project as defined above is of no consequence.

At runtime, if all parameters in an extended formula are not defined, the system continues processing as though the extended formula did not exist and puts an entry in the log.

An extended formula with a query association filter can follow the association or the reverse association path.

The rest of this chapter focuses on how to create extended formulas from within the Data Modeler.

In the Data Modeler, extended formulas can access fields in any record, so long as there is a sequence of associations that connect the record that contains the formula’s field to the record that the formula accesses. Extended formulas are available for fields with data type of Number, Date, or any other data type with a Formula check box in its properties.

Basic Features

The inputs to an extended formula can be fields or queries. This section begins by describing the properties of an extended formula using a field as an input. For additional features available when using a query as an input, see the Query section.

The IBM TRIRIGA Application Platform does not allow supplying an extended formula for a field until after the field is saved for the first time. In the Data Modeler, after a field has been saved for the first time, you can edit the field's properties to add an extended formula. Make sure that the Formula check box is checked and then select the Extended radio button for the Formula Type.

The examples in this section use a Number field for illustration. The same process applies to other field types that contain a Formula check box.

At this point the Field Properties form should look like the following, with an **Enter** link below the Extended radio button in the Formula Type property.

(a) Field Properties

- Section: `General`
- Field Type: `Number`
- Name: cstPerimeterNU
- Label: Perimeter
- Description: _____
- Purpose: _____
- Required: ☐
- Do not Auto Populate: ☐
- Result Column: ☐
- Mobile Field: ☐
- Staging Table Field: ☐
- Staging Table Key: ☐
- Read Only: ☑
- Default Value: _____
- UOM List: `Area`
- Default UOM: `square-feet`
- Validation: Accept `Decimal`
- Use Custom UOM Precision and Mask: ☐
- UOM Source Attribute
- Threshold Source Attribute
- Formula: ☑
- Sum this Field: ☐

- Formula Type: `Extended` **[Enter]**

To create or edit an extended formula, click the **Enter** link in the Formula Type property. Clicking the **Enter** link causes a form for specifying the details of an extended formula to appear in the Data Modeler's List panel.

The Extended Formula form is divided into three sections.

- The Inputs section specifies the fields or queries from which the formula will get values.
- The Outputs section identifies where the formula's result will go.
- The Formula section contains the text of the actual formula.

We will explain how to create formulas using fields as inputs by working through some examples. The first example will be an extended formula to compute the perimeter of a rectangle.

The formula has two inputs, which are the value of fields named Height and Width. Starting from a blank Extended Formula form, to add an input, click the **Add** action on the Inputs section bar. This adds a blank input to the Inputs section.

The Inputs section has a blank input containing the following:

- A radio button to select to if you want to delete the input.
- An empty text field. This is discussed in detail below.
- A drop-down list that has the default value `Field`. The input in this example comes from a field, so do not change the value.
- A **Select** hyperlink. This is used to select the field that will be the source of the input value, as discussed below.

Click the **Select** hyperlink. The Pick Element form pops up that allows selecting a field.

The Pick Element form shows a tree for choosing a field. The elements in the tree are ordered alphabetically. Sections on this field's business object appear first, followed by sections in other business objects associated to this field's business object. The tree can be dragged and resized within the Extended Formula form. Click the ⊕ icon to expand the view. Click the ⊖ icon to close the view. A ▪ icon indicates a field.

Use the formula tree to select the Height field (e.g., `cstHeightNU` under the `RecordInformation` parent) as the first input by clicking its name in the formula tree. After you select the Height field, the Pick Element form disappears. The Extended Formula form now shows a hyperlink in the input.

The hyperlink in the input shows the section and name of the field selected (e.g., `RecordInformation:cstHeightNU`).

If you had selected a field in a different business object, the hyperlink's text would have included the names of all associations used to get to the other business object. The hyperlink's text tells how to find the field selected. It can be long, awkward, and error prone to enter manually. For this reason, the hyperlink's text is not used to identify the field in the formula. Instead, the Inputs text box supplies a shorter name.

Type the name `h` into the text field. This allows us to use the name for this input in the formula section.

Repeat these steps to add the Width field as a second input with the name `w`. The Extended Formula form now shows the following:

- Input `h` for `RecordInformation:cstHeightNU`
- Input `w` for `RecordInformation:cstWidthNU`

The Outputs section identifies the field that will receive the value of the extended formula. The format is `data section:field` name. In this example, the Outputs section (e.g., `Record Information:Perimeter`) does not need to be changed. This is normal.

The extended formula is created with the field it belongs to as one of its outputs. Usually, a formula has no other outputs. If you want the result of the formula to be put in more than one field, you can specify this by adding the other fields that should receive the formula's result to the extended formula's Outputs section. If you add any outputs to an extended formula, you will be allowed to delete the outputs later on. However, you cannot delete the original field that the extended formula belongs to from its Outputs section.

Since no other inputs are needed by the example formula, we are ready to put the formula to be calculated in the Extended Formula form's Formula section. Enter the formula as `h*w*2`.

The Extended Formula form now shows the following:

- Input `h` and Input Type `Field` for `RecordInformation:cstHeightNU`
- Input `w` and Input Type `Field` for `RecordInformation:cstWidthNU`
- Output of `Record Information:Perimeter`
- Formula of `(h+w)*2`

You have finished specifying the extended formula, so click the **Ok** action at the top of the form. It is important to click the **Ok** action. If you do not, the extended formula is not saved and you lose it. You also need to click **Save Field** on the section bar at the top of the Data Modeler.

An extended formula can have any number of inputs. The following example shows an extended formula with four inputs.

- Input `h` and Input Type `Field` for `RecordInformation:cstHeightNU`
- Input `w` and Input Type `Field` for `RecordInformation:cstWidthNU`
- Input `l` and Input Type `Field` for `RecordInformation:cstLengthNU`
- Input `t` and Input Type `Field` for `RecordInformation:cstThicknessNU`
- Output of `Record Information:SampleCalc1`
- Formula of `(h-t-2)*(w-t-2)*(l-t-2)`

The Formula in an extended formula can contain the four arithmetic operators:

- **+** - Addition
- **–** - Subtraction
- ***** - Multiplication
- **/** - Division

The Formula of an extended formula also can contain parentheses to control the order in which the operations are performed. Multiplication and division are normally performed before addition and subtraction. This means that the value of `1+2*4` is `9`. The multiplication is done first; `2*4` is `8`. The addition is done next; `1+8` is `9`.

If you want the addition to be done first, use parentheses. The value of `(1+2)*4` is `12`. Because of the parentheses, the addition is done first; `1+2` is `3`. The multiplication is done next; `3*4` is `12`.

In addition to the arithmetic operators, there are other operations available as predefined formulas that you can use in the Formula text box. The example below shows an extended formula that uses a predefined formula named `sqrt`.

- Input `w` and Input Type `Field` for `RecordInformation:cstWidthNU`
- Input l and Input Type `Field` for `RecordInformation:cstLengthNU`
- Output of `Record Information:SampleCalc1`
- Formula of `sqrt(w*w+l*l)`

There are other predefined formulas provided by the IBM TRIRIGA Application Platform. To see a list of available predefined formulas, click the 🔍 icon to the right of the Formula text box. This causes a form containing a list of the available predefined formulas to pop up.

Some predefined formulas have inputs that are string or text values. The text values can come from fields of records or be the value in the formula. To put a string value directly in a formula, enclose it in double or single quotes like `"THIS"` or `'THAT'`.

If you need a text value that contains a single quote, enclose it in double quotes like this: `"O'Clock"`

If you need a text value that contains a double quote, enclose it in single quotes like this: `'He said, "Boo."'`

You cannot directly put a text value that contains both single and double quotes in a formula. However, you can concatenate to combine two or more text values into a single text value. To put the value `It is at "four o'clock"` in a formula you could write it as `'"four ' + "o'clock" + '"'`

The list of predefined formulas has two sections. The upper section shows a list of the formulas that have been predefined by people. The lower section shows formulas that have been predefined by the platform. Formulas predefined by people are described in Predefining Your Own Formulas. Formulas predefined by the platform are described in Formulas Predefined by the Platform.

Query

When the value of an input is to be the result of a query, first define the query in the Report Manager. The process for doing so is described in the *IBM TRIRIGA Application Platform 3 Reporting User Guide*. Once the query exists, it can be used in an extended formula. To do so, in the Inputs section select `Query` from the Input Type drop-down box and click the **Select** hyperlink to identify the query.

Supported Relationships

The definition of an extended formula can be simple or complex. This section describes which relationships the extended formula functionality supports. When a relationship is supported it means that if an extended formula exists and one of the events listed below occurs, the system recalculates the extended formula. Relationships not described in this section are not supported.

The following relationships are supported:

- Same Business Object
- Single-Hop Query
- Single-Hop Association
- Multi-Hop Query
- Multi-Business Object Query
- Multi-Hop Association
- Query Field Filters

Same Business Object

Formulas using fields or queries within its field's business object are supported.

Single-Hop Query

Formulas that rely on the result of a query that has a `$$RECORDID$$` association filter are supported. The association filter can be either a particular association type or the `ALL` association type.

For example:

- Section = `General`
- Field Type = `Number`
- Name = `triGrossAreaNU`
- Label = `Gross Area`
- Input `A` and Input Type `Query` for `Location---triFloor---triFloor – Formula – triBuilding – Total Gross Area---RecordInformation:triGrossAreaNumNU`
- Output of `Record Information:Gross Area`
- Formula of `A`

Single-Hop Association

Formulas that rely on a field's values in associated records are supported.

For example:

- Section = `General`
- Field Type = `Number`
- Name = `triHeadcountNU`

- Label = `Headcount`
- Input `a` and Input Type `Field` for `Parent For:Location:triSpace---RecordInformation:triHeadcountNU`
- Output of `Record Information:Headcount`
- Formula of `a`

Multi-Hop Query

Formulas that rely on a query that references another query in its association filter are supported. This can reference queries n-levels deep. The hopping continues until a referenced query has a `$$RECORDID$$` association filter.

The particular associations used in a multi-hop query are dictated by the query definitions. The association filters can be either a particular association type or the `ALL` association type.

For example:

(a) Building formula referencing a Space query:

- Input `A` and Input Type `Query` for `Location---triSpace---cstQuerySpaceForFloorsForBuilding---RecordInformation:triHeadcountNU`
- Output of `Record Information:Space Headcount`
- Formula of `A`

(b) Association filter for the referenced Space query:

- Association Type = `ALL`
- Module = `Location`
- Business Object = `triFloor`
- Filter Type = `Query`
- Record = cstQueryFloorsForBuilding

(c) Referenced sub-query that is a “hop” away from the Building formula:

- Association Type = `ALL`
- Module = `-Any-`
- Business Object = `All`
- Filter Type = `Record`
- Record = `$$RECORDID$$`

Multi-Business Object Query

Formulas that rely on a multi-business object query where a `$$RECORDID$$` field filter is against one of the query's associated (non-primary) business objects are supported.

For example:

(a) Query Data Source Setup:

- Module = `Location` | BO = `Space` | Association Type = –
- Module = `Location` | BO = `Floor (triFloor)` | Association Type = `ALL`
- Module = `Location` | BO = `Building` (`triBuilding`) | Association Type = `ALL`

(b) Query Field Filter Setup (e.g., System Filter Columns):

- Field = System Record ID (`triRecordIdSY`) where BO = `triBuilding`
- Report Label = `System Record ID`
- Filter Operator = `Equals`
- Value = `$$RECORDID$$`

Multi-Hop Association

Formulas that rely on a field's value that is in a business object associated n-levels away from the business object that the extended formula resides within are supported.

For example:

(a) Pick Element:

- ⊖ cstBuilding
- ⊕ General
- ⊕ About---triPeople---triPeople
- ⊖ Contains---Location---cstFloor
- ⊕ General
- ⊖ Contains---Location---cstSpace
- ⊖ General
- triHeadcountNU 🡐

- triGrossAreaNU

Query Field Filters

A query referenced by an extended formula typically has field filters. If a record changes and the change is such that the results of a query could be affected, the extended formula system recalculates the extended formula.

Relationships between objects can be specified by a query field filter through `$$RECORDID$$` filters and `$$PARENT::[Section Name]$$` filters.

There are limitations to what the queuing and queue processing in the system acts upon:

- Extended formulas are recalculated when a change to a record affects a filter applied to a field in the query's primary business object. If the affected filter is applied to a field in an associated business object, extended formulas are not recalculated.
- For multi-hop queries, extended formulas are recalculated when a change to a record affects a filter on the query directly referenced by the field formula. Extended formulas are not recalculated when a change to a record affects a filter referenced in any of the query's related reports.

Not Supported

The following list of formula building relationships that are not supported is not exhaustive and by its nature cannot include every instance. If a formula building relationship is not explicitly listed earlier in this "Relationships the Extended Formula Supports" section as supported, it is ***not supported***.

The list of non-supported formula building relationships includes:

- Related reports that do not have association filters.
- Query field filters on a non-primary business object.
- In a multi-hop query, filters on queries that are not directly referenced by the formula.

Formulas Predefined by the Platform

A number of formulas are predefined by the platform. They are System Formulas.

Note – Data Modeler extended formulas can be used only with Number and Date fields. The system formulas with strings do not apply.

Note - All Date, Date and Time, and Time fields are stored in the system in milliseconds. When a formula declaration calls for time in milliseconds, you can use a Date, a Date and Time, or a Time field as input.

Attention - Keywords and variables are case sensitive.

Predefined Formula	Description
abs(number)	Returns the absolute value of the number specified.
AddDay(timeInMillis,days)	Adds the number of days specified to the date, date and time, or time specified and returns the result in milliseconds. The system is leap year aware. If you add 365 days to a leap year you will get a different result than if you add 365 days to a non-leap year.
AddMonth(timeInMillis,months)	Adds the number of months specified to the date, date and time, or time specified and returns the result in milliseconds.
AddYear(timeInMillis,years)	Adds the number of years specified to the date, date and time, or time specified and returns the result in milliseconds.
applyRruleToEventUi(rRule,eventUiRecordid)	Applies a certain rule to a UI event for the specified record ID.
createQueryEndDateFromEvent(eventRecordId)	Creates a query to extract the end date from an event related to a certain record ID.
createQueryStartDateFromEvent(eventRecordId)	Creates a query to extract the start date from an event related to a certain record ID.
createRecurrenceExDate(contextRecordId,associationName,dateTimeFieldName)	Creates a query to determine the recurring execution date given the original date and time field, the context record ID and the association name.
createRelativeDateTime(inputDateTime,inputTimeZoneName)	Creates a query to determine the relative date and time given the original date and time and the specified time zone.
CurrentTime()	Returns the current time of the Application Server as the number of milliseconds since midnight, January 1, 1970 UTC.

Predefined Formula	Description
DateFromDateTime(timeInMillis)	Truncates a date, date and time, or time field to the date and returns the result in milliseconds. A time field always returns zero.
DateFromDateTimeTZ(date,timezone)	Converts a Date Time value to a Date value based on the given time zone. Example: DateFromDateTimeTZ(10/10/2012 12:00 AM EST,(GMT -5) Eastern Time (US, Canada) [US/Eastern]) returns 10/10/2012 Server Time Zone
datestring(date,format)	Returns a string from the date, date and time, or time specified in the format specified. The following replacements are made: ▪ yyyy = year ▪ M = month (1-12) ▪ MM = month (01-12) ▪ MMM = English 3 character abbreviation of month (Jan-Dec) ▪ MMMM = English full spelling of month (January-December) ▪ dd = day of month (01-31) ▪ h = hour (1-12) ▪ hh = hour (01-12) ▪ HH = hour of day (00-23) ▪ mm = minute (00-59) ▪ ss = second (00-59) ▪ SSS = millisecond (000-999) ▪ zzz = English 3 character abbreviation of time zone (e.g., GMT, PST) ▪ a = AM or PM
DateTimeFromDateTZ(date,timezone)	Converts a Date value to a DateTime value with respect to a time zone. When this function is applied to a date, it returns a date time value that has the time value zeroed to 12 midnight. Example: DateFromDateTimeTZ(10/8/2012,UTC-8) returns 10/8/2012 12:00 AM UTC-8
datetimestring(date,format,timezone)	Returns a string from the date, date and time, or time specified in the format and time zone specified.

Predefined Formula	Description
DayOfMonth(timeInMillis)	Returns the numeric value of the day of the month (1-31) from the date, date and time, or time specified. A time field always returns zero.
DayOfWeek(timeInMillis)	Returns the numeric value of the day of the week (1-7) from the date, date and time, or time specified. 1=Sunday, 2=Monday, …, 7=Saturday. A time field always returns zero.
DigitCount(string)	Returns the number of numeric digit characters in `string`.
div(a,b)	Returns the quotient of the number specified as `a` divided by the number specified as `b`. Any fractional part is discarded.
e()	Returns the mathematical constant e.
endsWith(string,endString)	Both inputs should be text values. Returns TRUE if the first string specified ends with the text specified in `endString`, otherwise returns FALSE.
excelMod(a,b)	Returns the module (rest) of the number specified as `a` divided by the number specified as `b`. Any fractional part is discarded.
getBoNameForRecord(recordId)	Returns the business object name of the record ID specified.
GetDays(startDate,endDate)	Returns the number of days between the two date, date and time, or time values specified.
getEventsForResource(resourceId,availabilityStart,availabilityEnd,calendarSetName)	Returns the number of events between the availability start date and end date, for the record ID and calendar specified.
getFirstDateTimeMatchingRrule(startDateTime,rRule,rRuleTimeZone)	Returns the first date and time that match the rule specified. Values for rRuleTimeZone must match values in the Time Zone Classification.
getModuleNameForRecord(recordId)	Returns the name of the module for the record ID specified.
getNextEventForResource(resourceId,searchFromDate,calendarSetName)	Returns the next event, counting from a certain date (searchFromDate), for the resource ID and calendar specified.
getRecordFromId(recordId)	Returns the record content for the record ID specified.

Predefined Formula	Description
getRecordsFromId(recordIds)	Returns more than one record contents for the record IDs specified.
getRuleFromEventUi(eventUiRecordId)	Returns the rule applied to a certain UI event for the record ID specified.
getUomType(recordId,fieldName)	Returns the type of unit of measurement (UoM) for the record ID and field name specified.
getUserFormattedDateTime(myProfileRecordId,dateTimeValue)	Returns the user's date format.
HasValue(field)	Used in workflow conditions to test if a value has been entered in a field. Supported field types are Number, Date, DateTime, Financial Rollup, and Classification Rollup. The function is true if a value has been entered and false if no value exists. This is not the same as a blank string value, HasValue tests if the value for the field has ever been set.
Hour(timeInMillis)	Returns the numeric value of the hour (0-11) from the date, date and time, or time specified.
HourOfDay(timeInMillis)	Returns the numeric value of the hour (0-23) from the date, date and time, or time specified.
if/If/IF/iF(condition, trueClauseExp, falseClauseExp)	Returns the result of the `trueClauseExp` expression if the `condition` expression evaluates to true; otherwise returns the result of the `falseClauseExp` expression. For more information, see If Expressions below.
indexof(string,substring)	Both inputs should be text values. Returns the number of characters in the first value that precede the first occurrence of the second value in the first value. If the second value does not occur in the first value, returns -1.

Predefined Formula	Description
IPMT(rate,period,numberOfPeriods,presentValue,futureValue,dueType)	**IPMT** or **Interest Payment** formula calculates the interest payment for a given period of an investment based on periodic, constant payments and a constant interest rate. The first input is the interest **rate** per period, which must be in a fraction form (already divided by 100). The second input is the **period** for which you want to find the interest and must be in the range 1 to **numberOfPeriods**. The next input is the total number of payment periods in an annuity. The fourth input is the **present value**, or the lump-sum amount that a series of future payments is worth right now. Then you have the **future value**, or the cash balance one wants to attain after the last payment is made. Enter 0 for the default future value which means that annuity based on present value is attained or paid off after the last payment is made. The **due type** indicates when payments are due. A warning is logged if the type is not supported and, in this case, the platform defaults to 0: ▪ **0** if payments are due at the end of the period ▪ **1** if payments are due at the beginning of the period For the **presentValue** and **futureValue** parameters, the cash you pay out (such as deposits to savings) is represented by negative numbers; the cash you receive (such as loan) is represented by positive numbers.
IPMTX(rate,period,numberOfPeriods,presentValue,payment,dueType)	This is a slight variation of the IPMT formula, where payment is specified instead of the future value. This enhanced formula allows to pre-calculate periodic payment (**PMT**) before calculating IPMT inside an iteration of periods. Using this formula instead of the regular IPMT yields better performance as it does not have to re-calculate the payment for every iteration. When pre-calculating PMT, make sure to specify the same rate, numberOfPeriods, presentValue, futureValue and dueType parameter values as you would specify for the regular **IPMT** formula to achieve the correct interest payment result.

Predefined Formula	Description
IRR(periods,payment,initial, hint)	Calculates the Rate of Return (IRR). IRR is related to Net Present Value (NPV). The value of IRR is the rate that will cause the NPV to result in zero. All parameters are Number field type. The hint parameter is a guess at the correct IRR. With a good hint the calculation can be performed much faster. If a hint is not available, 0 can be passed in and the function calculates a reasonable starting point. For example: Given an initial investment of $100,000 and a return of $175,000 for 3 periods, the IRR would be approximately 166%. In the IRR function you would get that using one of the following 2 calls: ▪ (using a hint) irr = IRR(3, 175000.00, 100000.00, 1.60) ▪ (without a hint) irr = IRR(3, 175000.00, 100000.00, 0) When checking results remember that calculating an IRR gives an approximate answer so it is possible that the result from IBM TRIRIGA's IRR function and from Excel might be slightly different. However, given a good hint they should be within a fraction of a percent (+- 0.001).
IRRX(initial,payments,hint)	This enhanced formula supports an array of payment values to the IRR calculation. The IRRX function requires 3 parameters and returns an approximate IRR value that is accurate to 12 decimal places. The parameters required are as follows: An initial payment - This is a positive number representing the undiscounted cost of an investment. An array of payments - This is an array of positive numbers representing payments that will be received by the investor against the initial investment. Rate hint - This is a number (positive or negative) that is a best guess or an approximation of the actual IRR. If a good hint is supplied it will make the calculation faster. If no hint is known use the value 0 and a hint will be calculated internally.
IsAMPM(timeInMillis)	Returns 0 to indicate the date, date and time, or time specified is A.M. and returns 1 to indicate P.M.
isResourceAvailable(resourceId,availabilityStart,availabilityEnd,calendarSetName)	Returns 0 to indicate the resource ID, within the specified availability start and end, and for a certain calendar, is not available and returns 1 to indicate it is available.

Predefined Formula	Description
isResourceAvailableForEventChange(resourceId,eventId,availabilityStart,availabilityEnd,calendarSetName)	Returns 0 to indicate the resource ID, within the specified availability start and end, and for a certain calendar, is not available for a certain event change and returns 1 to indicate it is available.
ln(number)	The input should be a positive number. Returns the natural (Napierian) logarithm of the number specified.
log(number)	The input should be a positive number. Returns the logarithm (base 10) of the number specified.
logn(number,base)	Both inputs should be positive numbers. Returns the logarithm of the number specified in the base specified.
LowercaseCount(string)	Returns the number of lowercase characters in string.
MilliSecondOfSecond(timeInMillis)	Returns the numeric value of the millisecond of the second (0-999) from the date, date and time, or time specified.
MilliSecondsFromDuration(duration)	Returns the milliseconds for the given duration. For example, IF(DURATION>0, MilliSecondsFromDuration(DURATION)/3600000, 0)
MinuteOfHour(timeInMillis)	Returns the numeric value of the minute of the hour (0-59) from the date, date and time, or time specified.
mod(a,b)	Returns the remainder of the number specified as `a` divided by the number specified as `b`. The sign of the returned value is the same as the sign of `b`.
Month(timeInMillis)	Returns the numeric value of the month (1-12) from date, date and time, or time specified. 1=January, 2=February, …, 12=December.
normsinv(probability)	Returns the inverse normal cumulative distribution of the number specified.
NPV(periods,payment,rate,initial)	All inputs should be numbers. Returns the net present value. The first input is the number of periods you expect to hold the investment. The second is the projected net cash flow per period. The third is the discount rate, which must be in a fraction form (already divided by 100). The fourth is the capital outlay to initiate the investment.

Predefined Formula	Description
NPVX(payments,rate,initial)	This enhanced formula supports an array of payment values to the NPV calculation. The period will then be determined by the number of payment values passed to the NPVX formula and the order of those values will be followed. All inputs should be numbers. The first input is the result of a query token, which is an array of number values. The second is the **rate** of discount over the length of one period. The third is the **initial** cost of investment which is to be deducted from the NPV result. The enhanced query token can be set to **Total** or **Values**: ▪ **Values** – assigns all the number values returned by the query as an array of numbers. The query token passed to the **NPVX** formula should have the type set to **Values** so that the formula can understand the array of number values. ▪ **Total** – assigns the total of all the number values returned by the query.
pi()	Returns the mathematical constant pi.
PMT(rate,numberOfPeriods,presentValue,futureValue,dueType)	The **PMT** or **Payment** formula calculates the periodic payment of an annuity based on constant payments and a constant interest rate. It contains the principal payment and interest of an annuity. The function can be used in **IPMTX** and **PPMTX** formulas to improve the performance when calculating these formulas inside an iteration of periods. The first input is the interest **rate** per period, which must be in a fraction form (already divided by 100). The second input is the total **number of payment periods** in an annuity. The third input is the **present value**, or the lump-sum amount that a series of future payments is worth right now. Then you have the **future value**, or the cash balance one wants to attain after the last payment is made. Enter 0 for the default future value which means that annuity based on present value is attained or paid off after the last payment is made. The **due type** indicates when payments are due. A warning is logged if the type is not supported and, in this case, the platform defaults to 0: ▪ **0** if payments are due at the end of the period ▪ **1** if payments are due at the beginning of the period For the **presentValue** and **futureValue** parameters, the cash you pay out (such as deposits to savings) is represented by negative numbers; the cash you receive (such as loan) is represented by positive numbers.

Predefined Formula	Description
power(number,exponent)	Exponent should be either a non-negative real number or a negative numeric value. Returns the result of the `number` specified raised to the `exponent` specified.
PPMT(rate,period,numberOfPeriods,presentValue,futureValue,dueType)	The **PPMT** or **Principal Payment** formula calculates the principal payment for a given period of an investment based on periodic, constant payments and a constant interest rate. The first input is the interest **rate** per period, which must be in a fraction form (already divided by 100). The second input is the **period** for which you want to find the principal payment and must be in the range 1 to **numberOfPeriods**. The next input is the total number of payment periods in an annuity. The fourth input is the **present value**, or the lump-sum amount that a series of future payments is worth right now. Then you have the **future value**, or the cash balance one wants to attain after the last payment is made. Enter 0 for the default future value which means that annuity based on present value is attained or paid off after the last payment is made. The **due type** indicates when payments are due. A warning is logged if the type is not supported and, in this case, the platform defaults to 0: ▪ **0** if payments are due at the end of the period ▪ **1** if payments are due at the beginning of the period For the **presentValue** and **futureValue** parameters, the cash you pay out (such as deposits to savings) is represented by negative numbers; the cash you receive (such as loan) is represented by positive numbers.
PPMTX(rate,period,numberOfPeriods,presentValue,payment,dueType)	This is a slight variation of the PPMT formula, where payment is specified instead of the future value. This formula allows to pre-calculate periodic payment (**PMT**) before calculating the PPMT inside an iteration of periods. Using this formula instead of the regular PPMT yields better performance as it does not have to re-calculate the payment in every iteration.
RandomNumber()	Returns a pseudo random number greater than or equal to 0 and less than 1. This function does not take any arguments.

Predefined Formula	Description

RandomString(minLen,maxLen,style)	Generates a random string value that is at least `minLen` characters long and no longer than `maxLen`. The string is generated using English characters (a-Z) and digits (0-9) based on rules defined by the `style` parameter.
	
	`minLen` and `maxLen`
	
	Define the minimum and maximum length of the generated string. The generated string will be at least `minLen` characters long and will be no longer than `maxLen` characters long. If `minLen` is less than 1 it will default to 1. If `maxLen` is less than `minLen` the minimum length will be used.
	
	`style`

RandomString(minLen,maxLen,style) *[continued]*	The `style` parameter is a string that is used to control certain aspects of the format of the generated string. Except for the special characters ‘0’, ‘1’, ‘i’, ‘I’, ‘l’, ‘L’, ‘o’, and ‘O’ (see below) the characters used in the `style` parameter do not matter. ▪ Allow ‘0’: If style contains a ‘0’ (zero) the string can contain the digit ‘0’; otherwise ‘0’ is not used. ▪ Allow ‘1’: If style contains a ‘1’ (one) the string can contain the digit ‘1’; otherwise ‘1’ is not used. ▪ Allow ‘i’/’I’: If style contains an ‘i’ (case does not matter for this rule) the string can contain ‘i’/’I’ characters; otherwise ‘i’/’I’ is not used. ▪ Allow ‘l’/’L’: If style contains an ‘l’ (case does not matter for this rule) the string can contain ‘l’/’L’ characters; otherwise ‘l’/’L’ is not used. ▪ Allow ‘o’/’O’: If style contains an ‘o’ (case does not matter for this rule) the string can contain ‘o’/’O’ characters; otherwise ‘o’/’O’ is not used. ▪ Uppercase only: If style contains only uppercase characters the string will be completely uppercase. ▪ Lowercase only: If style contains only lowercase characters the string will be completely lowercase. ▪ Digits only: If style contains only digit characters the string will be completely digits. ▪ Uppercase and Digits: If style contains only uppercase characters and digits the string will contain only uppercase characters and digits.

Predefined Formula	Description
RandomString(minLen,maxLen,style) *[continued]*	▪ Lowercase and Digits: If style contains only lowercase characters and digits the string will contain only lowercase characters and digits. ▪ Mixed case: If the style contains uppercase and lowercase characters the string will be mixed case. ▪ Mixed case and Digits: If the style contains uppercase, lowercase, and digit characters the string will be mixed case with digits. ▪ Start with uppercase: If the style starts with two uppercase characters the string will start with an uppercase character. ▪ Start with lowercase: If the style starts with two lowercase characters the string will start with a lowercase character. ▪ Start with digit: If the style starts with two digit characters the string will start with a digit. If the length of the generated string does not allow all the rules to be satisfied, the rules that specify the starting type take priority. For instance, if the generated string is two characters long and the rules specify to start with a digit and have mixed case; the string will start with a digit but will not have both uppercase and lowercase characters. `Style` Examples: ▪ "aA" - Generate a mixed case string without digits or the special characters 'I'/'l','L'/'l','O'/'o'. ▪ "aO" - Generate a mixed case string without digits allowing 'o' and 'O' to be used. ▪ "a2" - Generate a string with lowercase and digit characters without allowing the special characters. ▪ "AAa2" - Generate a mixed case string that includes digits that starts with an uppercase character. ▪ "01Aa" - Generate a mixed case string that includes digits, starts with a digit, and can contain the special characters '0' and '1'. ▪ "A" - Generate a string with uppercase characters only.
replace(string,oldstring,newstring)	All inputs should be text values. Returns the first string with all occurrences of the second string replaced with the third string.

Predefined Formula	Description
replaceCRLF(string, replaceString)	All inputs should be text values. Returns the first string with all occurrences of the carriage return line break (CRLF) replaced with the second string. Example: ▪ replaceCRLF(A, “<br/>”)
Round(value,precision)	Rounds the fractional portion of `value` to the number of decimal places given by `precision`. The operation looks at the fractional digits to the right of the digit given by `precision` to determine the value of the fractional digit at `precision` and returns the converted value. If the digit to the right is 5 or greater, the digit at `precision` is rounded up. If the digit to the right is 4 or less, the digit at `precision` is rounded down. This operation is sometimes referred to as ‘Round Half Up’. The value of `precision` should be greater than or equal to zero. If `precision` is less than zero it is treated as zero. Examples: ▪ Round(1.2345, 2) = 1.23 ▪ Round(1.235, 2) = 1.24 ▪ Round(1.2347, 3) = 1.235 ▪ Round(-1.12164, 3) = -1.122
RoundDown(value,precision)	Rounds the fractional portion of a `value` to the number of decimal places given by `precision`. The operation effectively truncates the value at the number of decimal places given by `precision`, discarding any fractional value to the right of the digit at `precision`. Examples: ▪ RoundDown(1.2345, 2) = 1.23 ▪ RoundDown(1.328, 2) = 1.32 ▪ RoundDown(-2.125, 2) = -2.12

Predefined Formula	Description
RoundUp(value,precision)	Rounds the fractional portion of a `value` to the number of decimal places given by `precision`. The operation increases the value of the digit at `precision` if there are any digits to the right of the digit at `precision`. Examples: ▪ RoundUp(1.2365, 2) = 1.24 ▪ RoundUp(1.2325, 2) = 1.24 ▪ RoundUp(1.23, 2) = 1.23
RoundX(value,precision, mode)	RoundX provides control of the rounding function via the `mode` parameter. There are seven modes. ▪ ROUND_CEILING (3) Round the fractional digit at precision so the value moves toward positive infinity. ▪ ROUND_UP (2) Same operation as RoundUp(value, precision). ▪ ROUND_HALF_UP (1) Same operation as Round(value, precision). ▪ ROUND_HALF_EVEN (0) If the fractional digit at precision needs to be rounded, round the value up or down as needed to make it even. This is sometimes referred to as a ‘Banker’s Round’. ▪ ROUND_HALF_DOWN (-1) If the fractional digit at precision needs to be rounded, use the digit to its right to determine the direction of the round. If the value is 6 or greater round up, if it is 5 or less round down. ▪ ROUND_DOWN (-2) Same operation as RoundDown(value, precision). ▪ ROUND_FLOOR (-3) If the fractional digit at precision needs to be rounded, round the fractional digit at precision so the value moves toward negative infinity. Examples: ▪ ROUND_CEILING RoundX(1.234, 2, 3) = 1.24 RoundX(1.12, 2, 3) = 1.13 RoundX(-1.234, 2, 3) = -1.23

Predefined Formula	Description
RoundX(value,precision, mode) *[continued]*	▪ ROUND_UP RoundX(1.234, 2, 2) = 1.24 RoundX(1.23, 2, 2) = 1.23 RoundX(-1.234, 2, 2) = -1.24 ▪ ROUND_HALF_UP RoundX(1.234, 2, 1) = 1.23 RoundX(1.236, 2, 1) = 1.24 RoundX(1.235, 2, 1) = 1.24 RoundX(1.23, 2, 1) = 1.23 ▪ ROUND_HALF_EVEN RoundX(1.234, 2, 0) = 1.23 RoundX(1.236, 2, 0) = 1.24 RoundX(1.235, 2, 0) = 1.24 RoundX(1.245, 2, 0) = 1.24 RoundX(1.23, 2, 0) = 1.23 ▪ ROUND_HALF_DOWN RoundX(1.234, 2, -1) = 1.23 RoundX(1.236, 2, -1) = 1.24 RoundX(1.235, 2, -1) = 1.23 RoundX(1.23, 2, -1) = 1.23 ▪ ROUND_DOWN RoundX(1.234, 2, -2) = 1.23 RoundX(1.236, 2, -2) = 1.23 RoundX(1.235, 2, -2) = 1.23 RoundX(1.23, 2, -2) = 1.23 ▪ ROUND_FLOOR RoundX(1.234, 2, -3) = 1.23 RoundX(1.236, 2, -3) = 1.23 RoundX(1.235, 2, -3) = 1.23 RoundX(-1.234, 2, -3) = -1.24
SecondOfMinute(timeInMillis)	Returns the numeric value of the second of the minute (0-59) from the date, date and time, or time specified.
sqrt(number)	The input should be a non-negative number. Returns the square root of the number specified.

Predefined Formula	Description
startsWith(string,endString)	Both inputs should be text values. Returns TRUE if the first string starts with the second string; otherwise returns FALSE.
stringlength(string)	Returns the number of characters in the string specified. The first position is zero. Note: ▪ stringlength(“”) = 0 ▪ stringlength(null) = 0
substring(string,start,end)	Returns the portion of the string specified that starts and ends at the indexes specified. Examples: ▪ substring(“abcdefgh”,3,5) => cde ▪ substring(“weight”,2,6) => eight

Predefined Formula	Description
toDate(dateString)	Returns the date and time that corresponds to the date and time in the string specified. It recognizes a variety of formats in the text, which follow a predefined sequence: ▪ Numeric date in milliseconds since January 1, 1970 ▪ MM/dd/yyyy hh:mm AM/PM ▪ dd/MM/yyyy hh:mm AM/PM ▪ dd-Month-yyyy HH:mm (Month name, Apr or April, and 24 hour time) ▪ dd/MM/yyyy hh:mm AM/PM zone (EST, PST, GMT-08:00, and so on) ▪ MM/dd/yyyy HH:mm:ss (24 hour time) ▪ dd/MM/yyyy HH:mm:ss (24 hour time) ▪ Month dd,yyyy hh:mm AM/PM zone (Month name and zone, for example, EST, PST, GMT-08:00, and so on) ▪ MM/dd/yyyy ▪ dd-Month-yyyy (Month name) ▪ dd/MM/yyyy ▪ Month dd/yyyy ▪ Month dd yyyy h:mmAM/PM Note that the input "10/11/2011" is recognized as October 11th since that will match the MM/dd/yyyy format. There is no way for this to be interpreted as the 10th of November. Where the formulas above say "Month", the month abbreviation (Apr) or the full month name (April) can be used.
tolower(string)	Returns the string specified with all upper case letters replaced with lower case letters.
toupper(string)	Returns the string specified with all lower case letters replaced with upper case letters.
trim(string)	Returns the string specified with all leading and trailing spaces or control characters removed. Note: ▪ trim("") = empty string ▪ trim(null) = null

Predefined Formula	Description
UppercaseCount(string)	Returns the number of uppercase characters in `string`.
WeekDayDate (timeInMillis,day)	Returns a date value which falls in the week specified in the first input. The second input is the day (1-7), where 1=Sunday, 2=Monday, …, 7=Saturday.
Year(timeInMillis)	Returns the numeric value of the year from date, date and time, or time specified.

If Expressions

There is something that looks like a predefined formula that you can use if you need to include a decision in your formula. For example, suppose that the value of a field represents a 10% surcharge on amounts greater than $100. For amounts less than $100, the surcharge is zero. Write this as

```
If(amount>100,amount*0.10,0)
```

The first input for `if` is something that is either true or false. If the first input is true, the value of the `if`'s second input is the value of the `if`. If the first input is false, the value of the `if`'s third input is the value of the `if`.

There are a few things about `if` that make it different. The first argument is that if the `if` expression can contain the four arithmetic operators, it can also contain other operators, like comparison operators.

The following list contains all of the operators that you can use with both extended formulas and workflow expressions:

(	open parenthesis
)	close parenthesis
-	subtraction
+	addition
/	division
*****	multiplication
==	is equal to
!=	is not equal to
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
&&	and
||	or

Another thing different about `if` is when the values of its inputs are computed. For regular predefined formulas, the values of all their inputs are computed and then the value of the predefined formula is computed.

For an `if`, the value of its first input is computed first. If the value of the first input is true, the value of the second input is computed; if the value of the first input is false, the value of the third input is computed. This means that depending on the value of the first input, the value of either the second or third input is not computed.

Each time the value of an `if` is computed, the value of either its second or third input is not computed. For most formulas, this fact is not important. However, there are some formulas for which this is crucial.

According to the rules for arithmetic, you are not allowed to divide by zero. Because of this rule, if a division operator in a formula tries to divide zero, it is not possible to compute a value for the formula. There are input values that can prevent the IBM TRIRIGA Application Platform from computing the result of a predefined formula. If the result of a formula cannot be computed, the IBM TRIRIGA Application Platform just uses zero as its result. If you need a value other than zero, the way that the inputs of an `if` are computed becomes important.

Here is an example of a formula that can be made to work only by taking advantage of the fact that the value of one of an `if`'s inputs is not computed: Suppose that you need to compute a value for a field named `Aspect Ratio`. The value of `Aspect Ratio` is `Height/Width` unless the value of `width` is zero. If the value of `Width` is zero, the result of `Aspect Ratio` is to be `999999999`.

The only way to write a formula that does this computation is to use `if`. Taking advantage of the fact that `if` does not compute the value of one of its inputs avoids a division by zero. The formula must avoid division by zero, otherwise its result is forced to be zero rather than 999999999. Write the formula for `Aspect Ratio` as `if(width#0,Height/Width,999999999)`. You can read this as `If width not equal to zero then Height divided by Width else 999999999`.

Predefining Your Own Formulas

You can create your own predefined formulas with the System page. To access the System page, select the **Tools** menu. Then click the **System Setup** menu item. When the third-level mega-menu appears, click **More.** On the result page, click **System** link. Then click **Formula** on the System page. The Formula result page displays a list of formulas with the following columns:

- Formula Name
- Formula Desc (Description)
- Formula Declaration
- Formula Expression

To add a new predefined formula, click the **Add** action on the Formula section bar. This causes a form to pop up like the following:

(a) General

- Formula Name
- Formula Desc
- Formula Expression

(b) Formula Parameters (with section actions: **Existing Formulas** | **Quick Add** | **Delete**)

- Parameter Position | Parameter Name

To see how to fill in this form, we will work through an example. The example that we will work through will be a formula named `diagonal` that computes the length of a rectangle's diagonal, given the rectangle's width and height.

Having the diagonal of a rectangle as a formula allows computing the diagonal length of a room without having to store the result in a field. We will be able to use it in extended formulas by writing something similar to `diagonal(width,length)`.

If you do not actually need to store a value that is simple to compute, it is usually better to compute the value.

To define the predefined formula `diagonal`, fill in the form as shown below:

(a) General

- Formula Name: `diagonal`
- Formula Desc: Compute the length of a rectangle's diagonal.
- Formula Expression: sqrt(width*width+height*height)

(b) Formula Parameters (with section actions: **Existing Formulas | Quick Add | Delete**)

- Parameter Position | Parameter Name
- 1 | width
- 2 | height

In the General section, type the name of the formula into the *Formula Name* field. Type the description of the formula into the *Formula Desc* field. Type the text of the formula into the *Formula Expression* field. There are two things to notice about the formula text in this example:

- The formula uses `sqrt`, which is a system formula. Predefined formulas also can use other predefined formulas. To see a list of available predefined formulas, click the **Existing Formulas** action on the Formula Parameters section bar.
- The formula uses the names `width` and `height`. These are not inputs. They are something else called *formal parameters*. A formal parameter is a name of a value that is used within a predefined formula. Formal parameters are not directly associated with values. What they are directly associated with are input positions.

The Formula Parameters section associates the formal parameter `width` with the position 1. It also associates the formal parameter `height` with the position 2. This means that if in an extended formula someone writes `diagonal(7,9)`, the predefined formula `diagonal` will be computed with `width` equal to 7 and `height` equal to 9. This is because `width` gets the value in position 1 and `height` gets the value in position 2.

When Extended Formulas Are Evaluated

Extended formulas can be extremely useful. However, to use them properly it is important to understand when the value of an extended formula is computed.

The value of an extended formula is computed when a change is made to a record. When a change is made to a record, if it contains any fields with a value that is determined by an extended formula, the value of each extended formula is recomputed and used to update the value of its field. When a user is editing a record using a form, the record is not modified until the user clicks an action that causes the changes to be stored in the record.

The timing is important. When a user changes any values in a record, only formulas of fields in that same record are recomputed immediately. If the value of a field is determined by a formula that refers to a field in another record, changes to the field in the other record only cause formulas in the other record to be recomputed. The value of the field with a formula that refers to the changed field in another record is updated through a queue, so it may not be reflected immediately.

When fields referenced by query filters change, the extended formula system checks potentially affected formulas and recalculates as needed. This recalculation only occurs for changes to fields in the query's primary business object and, for multi-hop queries, only for changes to filters on the query directly referenced by the field formula. The system will not detect and recalculate for fields in an associated business object or in a string of rs.

Extended Formula Agent

The purpose of the Extended Formula Agent is to maintain the results of complex formulas. Complex formulas are those that contain query and/or association tokens. They receive special processing because the source and target metadata of the formula reside in different business objects and therefore the source and target instance data of the formula reside in different records.

The Extended Formula Agent is managed in the Agent Manager panel of the Administrator Console, as described in the *IBM TRIRIGA Application Platform 3 Administrator Console User Guide*.

The Extended Formula Agent is a background thread that processes the Extended Formula Queue. The Extended Formula Queue is a table in the database containing information about data changes. The Extended Formula Agent reads this queue to determine which formulas in the system need to be recalculated as a result of data change.

Glossary

Term	Definition
Complex Formulas	Formulas that contain query and/or association tokens.
Extended Formula Cache	The cache of target fields and association relationships defined by the complex formula metadata on a system. Note that this is purely metadata so when comments refer to looking in the Extended Formula Cache, this means this is a metadata consideration only.

Extended Formula Components

There are three main components to managing complex Extended Formulas:

Component	Description
Loading Formula Relationships	At server startup, all complex extended formula metadata are evaluated and the relationships are loaded in to a cache. This cache is used at runtime to determine when instance data changes occur whether there are other records that could be affected by this change.
Populating the Queue	Putting records into the Extended Formula Queue (`EF_QUEUE`). This is done at the point that record and association changes occur.
Processing the Queue	The Extended Formula Agent reads the `EF_QUEUE`, determines what, if any, records may be affected, and recalculates the formulas on those affected records.

Loading Formula Relationships

The metadata relationships to describe complex formulas are stored in the Extended Formula Cache in the IBM TRIRIGA database. When an application server starts, it does not have to gather the business object relationships needed to do the extended formula queuing and queue processing. It just reads the Extended Formula Cache. This cache is essentially a map of important relationships with the purpose of facilitating the identification of field/association changes that could have a formula impact on other records in the system. Each token is either a query token or an association token.

- Association Tokens

Association tokens are parsed and put in to the cache.

An example of an association token: `[FIELD][{BO}{Parent For}{Location}{triSpace}][{BO}{Primary Location For}{triPeople}{triPeople}][{SECTION}{Detail}{triOneNU}]`

The key components to cache are the field, `triPeople.triOneNU`, and the association, `triSpace.Primary Location For.triPeople`. If the Extended Formula Activity log is enabled, such an event will be logged with a `com.tririga.architecture.cache.FormulaQueue.STARTUP` entry, which will indicate for example: `AssociationToken on BO, triProperty, relies on BO,triSpace, Field, triHeadcountNU, via association, Parent For.`

- Query Tokens

  Query tokens are parsed to acquire the different components. The query definition is then loaded and query's `$$RECORDID$$` association filter is evaluated for its association string.

  An example of a query token: `[QUERY][{Location}{triSpace}{triSpace - Formula - triFloor - Total Gross Area - Merchandizable Sales Area}][{RecordInformation}{triAreaNU}]`

  Key components acquired through this process are the field `triSpace.triAreaNU` and the association on the query's `$$RECORDID$$` association filter, `triFloor.Is Parent Of. triSpace`.

  If the record the calculation being done for is in a project, the project context comes from that record, but it is only used if the query definition specifies a Data Scope of `Active Project`.

The following events trigger an update to the Extended Formula Cache:

- A business object is published, either initiated by a user in the Data Modeler or by an Object Migration Import.
- A query is created, modified, or imported.
- A user clears the Extended Formula Cache from the Administrator Console.

Populating the Queue

The Extended Formula Queue is represented in the database as `EF_QUEUE`. There are three events that could cause an entry to go in to the Extended Formula Queue:

- When a record-field has changed values and that field has the potential to affect a complex formula value, the record is put in the Extended Formula Queue.
- When an association has been created and that association has the potential to affect a complex formula value, the association is put in the Extended Formula Queue.
- When an association has been deleted and that association has the potential to affect a complex formula value, the association is put in to the Extended Formula Queue.

If the Extended Formula Activity log is enabled, such an event will be logged with a `com.tririga.ExtFormulaAgent.ACTIVITY` entry prefixed by `Change put in queue`.

The system reviews entries before they go into the queue and does not add an entry if it already exists in the queue.

Processing the Queue

The Extended Formula Agent is responsible for processing the Extended Formula Queue. It processes the queue in first in / first out order so that calculations are done in the order of the data changes that necessitated the calculations occur. This ensures there is a priority to the calculations done and that changes made by User 1 are given priority over changes made by User 2 if User 1 made changes before User 2.

The Extended Formula Agent sleeps for 5 seconds between queries of the Extended Formula Queue.

The steps it goes through to process the queue are as follows:

- Pick up at most 100 records from the `EF_QUEUE` table and tag them in the database with an agent id. The `EF_QUEUE.AGENT_ID` column before being tagged is `-1`.
- Bring those tagged records into memory. If the Extended Formula Activity log is enabled, such an event will be logged with a `com.tririga.ExtFormulaAgent.ACTIVITY` entry prefixed by `Handling queued change`.

- For each queue record, determine which records' formulas need to be recalculated:
  - If the record is an association, determine whether the associated record, `EF_QUEUE.ASS_SPEC_ID`, needs to have its formulas recalculated. If the Extended Formula Activity log is enabled, such an event will be logged with a `com.tririga.ExtFormulaAgent.ACTIVITY` entry prefixed by `Queued change necessitates save`.
  - Most records are not associations and, for those, it retrieves the associated records. The association type used is based on the extended formula relationships that the system loaded at startup.

    The query looks something like the following: `select spec_id from ibs_spec_assignments where ass_spec_id = ? and spec_template_id = ? and spec_class_type = ? and ass_type = ?`.

    The extended formula agent determines whether any record(s) returned need to have their formulas recalculated. If the Extended Formula Activity log is enabled, such an event will be logged with a `com.tririga.ExtFormulaAgent.ACTIVITY` entry prefixed by `Queued change necessitates save`.
- Iterate through the records with formulas that need to be recalculated and recalculate the complex formulas on those records. If the Extended Formula Activity log is enabled, such an event will be logged with a `com.tririga.ExtFormulaAgent.ACTIVITY` entry prefixed by `Record saved by agent`.

  When the same query is used across multiple extended formula parameters in the same business object, that query is executed once and the result is reused in the other extended formulas. This does not cause data integrity issues because at the point the query is run, the record save has not yet put its data into the database.
- When all the records that need to be recalculated have been recalculated, delete the `EF_QUEUE` entries that were tagged.

Examples

Association Token

Example: `[FIELD][{BO}{Parent For}{Location}{triSpace}][{BO}{Primary Location For}{triPeople}{triPeople}][{SECTION}{Detail}{triOneNU}]`

1. A change is made to field value `triPeople.triOneNU` on a `triPeople` record. Given the cached token above, it will put the changed `triPeople` record in the Extended Formula Queue. When the record is picked up by the Extended Formula Agent, it will look in the Extended Formula Cache and see that `triSpace` records related to the `triPeople` record via a `Primary Location For` association may be affected by the change to `triPeople.triOneNU`. It will then search for any such related `triSpace` records. If any `triSpace` records are found, the complex formulas on those records are recalculated.

2. A `Primary Location For` association is created between `triSpace` record and a triPeople record. Given the cached token above, it will put the association in the Extended Formula Queue. When the record is picked up by the Extended Formula Agent, it will look in the Extended Formula Cache and see that `triSpace` records related to the `triPeople` record via a `Primary Location For` association may be affected by association just created. It will then search for any such related `triSpace` records. If any `triSpace` records are found, the complex formulas on those records are recalculated.

Query Token

Example: `[QUERY][{Location}{triSpace}{triSpace - Formula - triFloor - Total Gross Area - Merchandizable Sales Area}][{RecordInformation}{triAreaNU}]`

In this example, the query has a `$$RECORDID$$` association of type `Is Parent Of`.

1. A change is made to field value `triSpace.triAreaNU` on a `triSpace` record. Given the cached token above, it will put the `triSpace` record in the Extended Formula Queue. When the record is picked up by the Extended Formula Agent, it will look in the Extended Formula Cache and see that `triFloor` records related to the `triSpace` record via an `Is Parent Of` association may be affected by the change to `triSpace.triAreaNU`. It will then search for any such related `triFloor` records. If any `triFloor` records are found, the complex formulas on those records are recalculated.

2. An `Is Parent Of` association is created between `triFloor` record and a `triSpace` record. Given the cached token above, it will put the association in the Extended Formula Queue. When the record is picked up by the Extended Formula Agent, it will look in the Extended Formula Cache and see that `triFloor` records related to the `triSpace` record via an `Is Parent Of` association may be affected by association just created. It will then search for any such related `triFloor` records. If any `triFloor` records are found, the complex formulas on those records are recalculated.

Extended Formula Logging

Logging adds information about what the Extended Formula Agent is doing. Logging can be enabled on-the-fly with the Platform Logging managed object in the Administrator Console, as described in the *IBM TRIRIGA Application Platform 3 Administrator Console User Guide*.

This includes the capability to:

1. Log the relationships considered by the Extended Formula Agent, enabled by setting the `com.tririga.architecture.cache.FormulaQueue.STARTUP` category to `DEBUG`-level. This is a one-time logging activity when the system starts up and should have no ongoing effect on system performance.

   Note - If you enable the Extended Formula Startup option after the system is already started, you can cause the system to reload and log the STARTUP information. Select the Cache Manager managed object in the Administrator Console and click Extended Formula Cache, which refreshes that cache.

2. Log the Extended Formula Agent activity on a system, enabled by setting the `com.tririga.ExtFormulaAgent.ACTIVITY` category to `DEBUG`-level. This is a runtime log that logs frequently, will grow the log file at a rapid rate, and could affect system performance. This should only be used for short periods of time to troubleshoot Extended Formula Agent behavior.

Both write to a log file named `extFormulaActivity.log`; however, `log4j.xml` can be configured to do it differently.

A variety of events are logged:

`[...FormulaQueue.STARTUP] AssociationToken...`: When an association token is cached at system startup

`[...FormulaQueue.STARTUP] QueryToken...`: When a query token is cached at system startup

`[...ExtFormulaAgent.ACTIVITY] Change put in queue...`: When a record is put in the Extended Formula Queue

`[...ExtFormulaAgent.ACTIVITY] Handling queued change...`: When the Extended Formula Agent picks up a record from the Extended Formula Queue

`[...ExtFormulaAgent.ACTIVITY] Queued change necessitates save. Data to save...`: When it is determined that a record needs to be saved due to a record having been queued. This will always be nested within a `Handling queued change` entry.

`[...ExtFormulaAgent.ACTIVITY] Record saved by agent...`: After it is determined that an entry needs to be saved, this entry will be logged when the record actually is being saved. Saved in this case implies that the complex formula(s) on the record are recalculated.

An entry put in the queue is given a Change ID. This is seen in the logged entry when an item is added to the Extended Formula Queue and is seen in the logged entry when the item is picked up from the queue. This provides a means of correlating queuing activity with agent activity.

This log entry example shows the online thread logging a change with a particular changeId:

```
Change put in queue: [Record:
[ID.Type.Name=10370588.cstSpace.1] Fields: [cstHeadcount]]
changeId=[4]
```

This log entry example shows the agent thread logging with the changeId when that same entry was picked up:

```
Handling queued change (0 changes left): [Record:
[ID.Type.Name=10370588.cstSpace.1] Fields: [cstHeadcount]]
changeId=[4]
```

Entries show how many items are left to process in the current batch so that monitoring the log provides information about how much work is left to do. These sample log entries illustrate that as entries are picked up from the queue, the log entries display how many changes are left to process:

```
Handling queued change (2 changes left): [Record:
[ID.Type.Name=10370588.cstSpace.3] Fields: [cstHeadcount]]
changeId=[4]

Handling queued change (1 change left): [Record:
[ID.Type.Name=10370588.cstSpace.2] Fields: [cstHeadcount]]
changeId=[4]
```

```
Handling queued change (0 changes left): [Record:
[ID.Type.Name=10370588.cstSpace.1] Fields: [cstHeadcount]]
changeId=[4]
```

Logging shows how data is being traversed to find the affected data for a multi-hop scenario. In this example, the log entry shows that a change to `cstSpace.cstHeadcount` traversed through floor data to ultimately find the `cstBuilding.cstSpaceHeadcount` that was affected by the change.

```
Handling queued change (0 changes left): [Record:
[ID.Type.Name=10370588.cstSpace.1] Fields: [cstHeadcount]]
changeId=[4]

Affected intermediate record:
[ID.Type.Name=10370463.cstFloor.1] Caused by formula on
field:
[Module.BO.Field=cstLocation.cstBuilding.cstSpaceHeadcount]

Affected field
[Module.BO.Field=cstLocation.cstBuilding.cstSpaceHeadcount]
on record: [ID.Type.Name=10370448.cstBuilding.1] Caused by
formula on field: [Module.BO.Field=
cstLocation.cstBuilding.cstSpaceHeadcount]
```

The system attempts to recalculate only those fields affected by a particular change. Activity logging shows when a record save only recalculates specific fields. In this example, the log entry shows that only the `cstBuilding.cstSpaceHeadcount` was recalculated when the `cstSpace.cstHeadcount` was changed.

```
Queued change necessitates save. Data to save: [Record:
[ID.Type.Name=10370448.cstBuilding.1] Field:
[cstSpaceHeadcount]] Caused by formula on field:
[Module.BO.Field=cstLocation.cstBuilding.cstSpaceHeadcount]
```

Note that if no field(s) are shown in the ‘Data to save’ log entry, the system recalculated all complex formulas on the record.

FAQs

1. Can Extended Formula Agents be run on multiple servers?

   No. The Extended Formula Agent is designed to run on a single server. Running on multiple servers could cause undesired behavior such as database deadlocks, and is not allowed by the Agent Manager in the Administrator Console.

2. What are the differences in the behaviors on IBM TRIRIGA Application Platform 2.5 vs. 2.6 vs. 2.7 vs. 3.0?

   There should be no functional differences. There were many functionality and performance improvements in 2.7, none of which affect behavior of existing extended formulas. Internally in 2.6, the Extended Formula Agent uses the record-runtime (smart object) code for its record saves, taking advantage of efficiencies that improve the performance of processing the Extended Formula Queue vs. 2.5.

3. What happens if a query token has a query that does not have a `$$RECORDID$$` association filter but actually has another query referenced in its association query? Is this relationship maintained by the Extended Formula Agent?

   Yes. This is what is called a double-hop query. See the discussion of multi-hop queries above.

2. Financial Transactions and Rollups

This chapter discusses two distinct but related features of the IBM TRIRIGA Application Platform. One feature is the ability to roll up numbers from records that contain the most detailed information through a hierarchy into one or more levels of records that contain summary information. The other feature is the ability to report financial and other numeric transactions that occur in applications to users and to external applications.

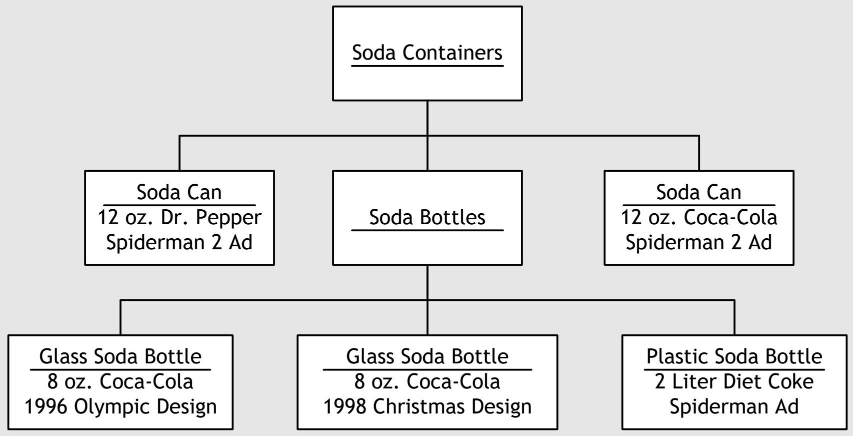
The IBM TRIRIGA Application Platform supports two different mechanisms for rolling up numbers. One is a relatively simple mechanism for rolling up numbers directly through a hierarchy. The other is a more flexible and feature-rich mechanism that uses a hierarchy indirectly to roll up numbers that are part of a transaction.

The first section of this chapter describes the mechanism for rolling numbers up through a hierarchy. This is followed by a description of financial transactions and how they can be used to roll up totals into records that are not organized into a hierarchy. After describing hierarchical rollups and financial transactions, the chapter compares the two different methods of rolling up numbers.

Rolling Up Through a Hierarchy

The “Hierarchies” chapter of *Application Building* describes the IBM TRIRIGA Application Platform’s support for organizing records into a hierarchy. The platform has a feature that allows numbers to be rolled up through a hierarchy. Each record in a hierarchy can contain totals of numbers that come from records that are its immediate children in the hierarchy.

To illustrate, look at a simple example. Assume we want to write an application to manage a collection of soda containers. The different kinds of containers are organized into a hierarchy that looks like the following:



Hierarchy of Soda Containers

All records at the bottom of the hierarchy contain the number of pieces (e.g., cans, bottles) of the type they describe. We want records that have children to contain the total number of pieces under them. The Soda Bottles record should contain the total number of bottles in the collection. The Soda Containers record should contain a total of the number of cans and bottles in the collection.

The platform provides a mechanism to roll these totals up a hierarchy of records. The mechanism is based on a type of field called *Classification Rollup*. Numbers are rolled up into Classification Rollup fields. Numbers also may be rolled up from Classification Rollup fields.

Classification Rollup Fields

A field of type Classification Rollup can be configured either to allow a number to be directly entered into it or to contain the total of numbers from fields in child records. Classification Rollup fields are associated with a classification. The classification determines how data rolls up into or out of the field. The classification associated with a Classification Rollup field is set when the field is defined.

Classification Rollup fields should be used only in business objects that are in a hierarchy module.

The form showing the properties of a Classification Rollup field looks like the following:

- Section: `General`
- Field Type: Classification Rollup

- Name: _____
- Label: _____
- Description: _____
- Purpose: _____
- Required: ☐
- Do not Auto Populate: ☐
- Result Column: ☐
- Mobile Field: ☐
- Read Only: ☑
- Rollup Source: `Roll Up`
- Display Mask: `#.####`
- Storage Precision: `12`
- Rounding Rule: `Half Up`
- Root Classification
- Formula: ☐
- Threshold Source Attribute

Most of the properties in a Classification Rollup field are the same as other kinds of fields and are described in detail in the "Data Types" chapter of *Application Building*. Four properties visible when a Classification Rollup field is first added are specific to Classification Rollup fields. They are as follows:

- The value of the *Root Classification* property determines the classification associated with the field. Click the Search icon 🔎 to identify the classification.
- The value of the *Rollup Source* property determines how data gets into the field. The four choices in the drop-down list are as follows:

Value of Rollup Source Property	Description
Roll Up	If the value of the Rollup Source property is `Roll Up`, the number in the field will be the total of numbers in Classification Rollup fields with the same field name in child records. Numbers roll up from Classification Rollup fields in child records into Classification Rollup fields in a parent record with the Rollup Source property set to `Roll Up` and with the Root Classification property the same as the properties in the child record.

Value of Rollup Source Property	Description
Classification	If the value of the Rollup Source property is `Classification`, the number in the field will be a total computed from child records filtered using the value of their Classification field(s). Only child records that have a Classification field with a value that is the same as or under the classification associated with the parent Classification Rollup field are included in the total. The system ignores child records that contain any Classification Rollup fields. Where in the child records the number to be totaled comes from is determined by a property named *Rollup Type*. The Rollup Type property is visible only if the value of the Rollup Source field is `Classification`. The Rollup Type property shows three radio buttons labeled `Cost`, `Quantity`, and `Field`. Use these radio buttons to specify where in child records the number to total will come from. The `Cost` and `Quantity` values are discussed later in this chapter in "Cost, Quantity". When the Rollup Type is `Field`, it means that the numbers to total come from a Number field in the child records. Selecting `Field` causes the *Rollup Field* property to appear in the Classification Rollup field's properties. The value in Rollup Field identifies the source of the number in the child records. The Rollup Field property is visible only when the Rollup Type is `Field`. The field names in the Rollup Field's drop-down list are the names of Number fields in the same business object. For this to work, the child records must have fields with the selected name. If having the names match is a problem, select `Cost` or `Quantity` for Rollup Type instead of `Field`.
Both	The value of the Rollup Source property should not be set to `Both`. This value exists to support an obsolete feature and will be removed from the IBM TRIRIGA Application Platform.

Value of Rollup Source Property	Description
Data Entry	If the value of the Rollup Source property is `Data Entry`, it means that a number will not be rolled up into the field. Instead, the value of the field is set by data entry, a default value, a formula, or other means. The number that is the value of the field may roll up into a Classification Rollup field in a parent record that has the Rollup Source property set to `Roll Up`. Which, if any, Classification Rollup fields in the parent record the field's value rolls up into is determined by the classification specified in the *Root Classification* property.

These values for Rollup Source allow two different ways to roll totals up through a hierarchy. The `Data Entry` and`Roll Up` values for Rollup Source allow data to be rolled up in a predetermined path, using classifications specified in the Data Modeler to determine which Classification Rollup fields will roll up into a given Classification Rollup field. When Rollup Source is `Classification`, data is filtered and rolled up in a path that is determined by the value of records' Classification fields at the time of the rollup.

- When there is a value in the *Threshold Source Attribute* property, this Classification Rollup field is a scored Classification Rollup field, meaning it will be scored (as described in "Comparison" in *Application Building*). The score is displayed as a red, yellow, or green image to the left of the Classification Rollup field's value.

  The value of the Threshold Source Attribute property must be a locator field that references the Threshold business object. This locator field must be in either a General section or a one-to-one smart section. For a Classification Rollup field that is in a smart section, the linked locator field must reside on the referenced business object. The Threshold record defines the threshold ranges and contains a UOM value that is used to convert the scored value.

  Select the linked locator field in the Threshold Source Attribute drop-down list. The list only shows locator fields in the same business object that references the Threshold business object.

  A scored Classification Rollup field can be used in a form section, non-table smart section, table smart section, or a vertical table section.

  The score for the field is calculated during the rendering of the field and is not stored. The platform compares the UOM of the Classification Rollup field with that of the Threshold record. If a conversion is needed, the platform converts the Classification Rollup value before comparing it with the threshold value to calculate the score.

During runtime, the system uses the threshold record that is currently selected by the linked locator fields. If the locator field does not have a value, the Classification Rollup field displays without a score.

The score is updated when the tab is reloaded. This means that if the value of the Classification Rollup field, or the threshold record, or the locator field that links to the threshold record, is changed, the score will not update (if needed) until the tab is reloaded.

When designing a hierarchy for rolling up data, there is an important restriction to be aware of. Classification Rollup fields with Rollup Source set to `Classification` ignore child records that contain Classification Rollup fields. This means that a record can be a child for one kind of rollup or the other, but not both.

Classification Rollup fields have no unit of measure. This means that it is crucial for all numbers rolled up into a Classification Rollup field to have the same unit of measure. Because Classification Rollup fields have no unit of measure, no conversions are done to resolve inconsistencies between different fields being rolled up.

Cost, Quantity

Sometimes you want numbers to be rolled up from a field in child records into a parent record's Classification Rollup field that has the Rollup Source property set to `Classification`. This is straightforward if all the child records are the same kind of record. You just need to specify the name of the field you want the numbers to come from.

If there is more than one kind of child record, it is still straightforward to specify which field the numbers will come from if the field that the number comes from in each kind of child record has the same name. If there is more than one kind of child record and for each kind of child record numbers should come from fields with different names, things are more complex. The IBM TRIRIGA Application Platform allows you to handle this by indirectly specifying from which field the numbers should come.

The indirect specification of field names is done in the mapping properties of the business object used to create each kind of child record. First designate a number field as the quantity or cost of records created from the business object. Then configure the Classification Rollup field to get its values from the quantity or cost of the child records.

To access a business object's mapping properties, select the business object in the Data Modeler and select BO Mapping from the Data Modeler's Tools menu. You can read more about the Data Modeler, mapping properties, and BO Mapping in *Application Building*. The following shows what the form for editing a business object's mapping properties looks like:

(a) Mapping Properties:

- Name: [Find]
- Cost: [Find][+][Find][+][Find][+][Find]
- Quantity: [Find]
- Image
- Conversion Group
- Exchange Date
- Control Number: ☐ Based on Prefix
- Prefix: [Find][Add][Delete]
- Suffix: [Find][Add][Delete]
- Start With: `1`
- Delimiter

Although the first two properties after *Name* are labeled *Cost* and *Quantity*, they do not have to be used for those purposes.

The value for Quantity is the value of whatever field is selected. The value for Cost is a simple formula that can refer to more than one field.

You can configure a Classification Rollup field to roll up the values specified as Cost or Quantity by specifying `Classification` as the value of its Rollup Source property and selecting `Cost` or `Quantity` as its Rollup Type.

Rollup Actions

Unlike a spreadsheet that updates totals as details are added or changed, the totals in a hierarchy are not updated automatically. A record's Classification Rollup fields are updated from values in child records only when requested.

There are two ways to ask for values to be rolled up from a record's children:

- A user can manually cause a rollup to be done on a record. When using a form to interactively edit a record created from a business object that is in a hierarchy module, the form will have a Roll Up action in its menu bar. Clicking the Roll Up action causes values from the record's children to be rolled up into its Classification Rollup fields.
- A workflow can cause a rollup to be done on a record by performing a Rollup action on the record.

Whichever of these mechanisms is used to cause a rollup, values are rolled up only from a record's immediate children. If there is a lower level to the hierarchy and the values in a record's immediate children do not reflect changes to the lower level, it is these stale values that will be rolled up.

To use this one-level-at-a-time rollup to make all rolled up totals consistent after a detail has changed, do a rollup into the immediate parent of the record that changed, then do a rollup into its parent until you have rolled up the new totals to the hierarchy's root.

If you interactively edit a record that is part of a hierarchy and contains rolled up totals, it is important to do a roll up after you change a number that should be rolled up into the parent record. It is also important to remember that the rollup has to be done to the parent record, not the child record that was modified. The easiest way to ensure that these are not forgotten is to have a workflow associated with the record's Save and Save & Close actions do the rollup on the parent. Expecting users to remember to edit a record's parent to roll up changes made in a child record is not a good idea, because it is too easy to forget.

Consider the soda containers hierarchy shown earlier. After a change is made to numbers in the record named "Glass Soda Bottle 8 oz. Coca-Cola 1998 Christmas Design" you may want to roll up the new numbers in the total to the records above it in the hierarchy. To do this, first perform a rollup on the record labeled Soda Bottles. Then finish by doing a rollup on the record named Soda Containers.

This piecemeal way of rolling up data makes the most sense when changes are infrequent and the shape of the hierarchy is wide and shallow. In this situation, having workflows perform rollups on just the affected records may be faster than a method that would involve recomputing and rolling up all the totals in the hierarchy.

There are situations when recomputing and rolling up all the totals in the hierarchy all at once makes more sense. For example, if changes to the data in the hierarchy are frequent but the rolled up totals are seldom used, it may make more sense to recompute and roll up all the totals in the hierarchy just before they are needed.

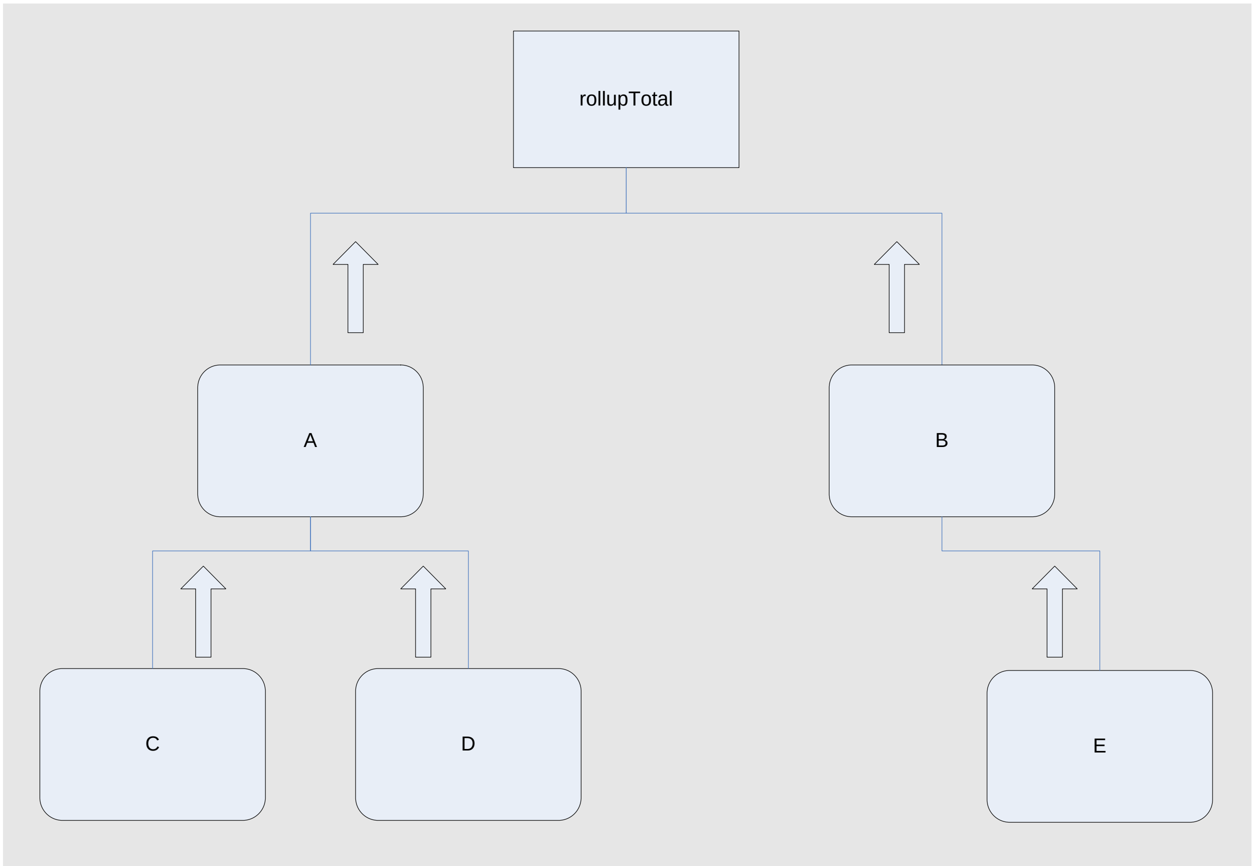
A workflow can cause all the totals in the hierarchy to be recomputed and rolled up all at once. If a workflow performs a Rollup All action on a record, it causes data from all of the record’s children to be rolled up, not just the immediate children. If a workflow performs a Rollup All action on the root of a hierarchy, it causes all the totals in the hierarchy to be recomputed and rolled up.

Storage Precision

Classification Rollup fields use values from many records in a hierarchy to compute values. The platform handles the computation involved in rolling up these values based on the value of the *Storage Precision* and *Rounding Rule* properties. Storage Precision specifies the maximum number of decimal places to use when the field is stored in the database and when performing computations. Rounding Rule identifies how the platform will round raw data to the precision in Storage Precision.

When the platform performs classification rollup computations, it uses intermediate values as part of the computation. No intermediate rounding is explicitly done; however, some rounding may have occurred already based on the Storage Precision and Rounding Rule specified in the records being rolled up. The system uses Storage Precision of `12` and Rounding Rule of `ROUND_HALF_UP` for intermediate calculations. Only the final calculation to that produces the result is rounded based on the Storage Precision and Rounding Rule in the Classification Rollup field.

For example, the diagram below shows an example rollup hierarchy.



Storage Precision Example

In this example, rollupTotal=A+B+C+D+E. Assume the storage precisions for A, B, C, D, and E are 4, 6, 3, 7, 8 respectively, and the storage precision for rollupTotal is 2. A, B, C, D and E will be added to an intermediate value with a precision of 12. Then, that value is stored as rollupTotal using the storage precision of 2 (defined on the business object for rollupTotal) and the appropriate rounding rule.

Hierarchy Rollup Example

This section uses the soda containers hierarchy shown earlier to demonstrate a hierarchy rollup. This hierarchy keeps information about a collection of soda cans and bottles. Assume the purpose of this hierarchy is to keep track of how many cans and bottles are in the collection. This can be accomplished using Classification Rollup fields with their Rollup Source property set to `Roll Up` or `Data Entry`. This kind of rollup is called a *straight rollup*.

For a straight rollup, a hierarchy contains two kinds of records: those containing detail data and those containing rolled up totals. Both kinds of records use a

Classification Rollup field to contain their data. In records that contain detail data, the Rollup **Source** property of their Classification Rollup field is `Data Entry`. In records that contain rolled up totals, the Rollup Source is `Roll Up`.

In the soda containers hierarchy example, records that describe soda cans contain detail data and records that describe glass and plastic soda bottles also contain detail data. The record in the hierarchy that is the parent for all kinds of soda bottles contains a rolled up total, the total number of bottles in the collection. The record in the hierarchy that is the parent for all kinds of soda containers also contains a rolled up total that contains the total number of cans and bottles in the collection.

The example below shows the fields in Glass Soda Bottle records. The fields for Plastic Soda Bottle and Soda Can records are similar. The field involved in the straight rollup is the Classification Rollup field named RollupQuantity (`cstRollupQuantityCR`).

(a) Field List

- Field Name | Field Label | Field Type
- cstBrandCL | Brand | Classification
- cstDescriptionTX | Description | Text
- cstHasTwistOffTopBL | Has Twist-Off Top | Boolean
- cstNameTX | Name | Text
- cstRollupQuantityCR | Quantity | Classification Rollup

The RollupQuantity field is shown below:

(b) Field Properties

- Section: `General`
- Field Type: Classification Rollup
- Name: cstRollupQuantityCR
- Label: `Quantity`
- Description: _____
- Purpose: _____
- Required: ☐
- Do not Auto Populate: ☐
- Result Column: ☐
- Mobile Field: ☐

- Read Only: ☑
- Rollup Source: `Data Entry`
- Display Mask: `#.####`
- Storage Precision: `12`
- Rounding Rule: `Half Up`
- Root Classification: `each`
- Formula: ☐
- Threshold Source Attribute

The Rollup Source property is set to `Data Entry`. This setting allows a user to enter numbers into the field.

The Root Classification field is set to `each`. For straight rollups, there are just two considerations for choosing a classification:

- The classification in the child field must be the same as the classification in the parent field. Only child fields that are supposed to rollup into a particular field in the parent should have the same Root Classification setting as the field in the parent.
- The platform does not care which classification is used for the Root Classification setting. To help document the rollup's purpose, choose a classification that suggests something about its nature.

In our example, the full path of the chosen classification is `Classifications/Unit Type/Quantity/each`. This is a classification, not a unit of measure. However, its name suggests that the rollup has something to do with quantities measured in each, which is exactly what it is. The following shows the fields in the Soda Bottles record.

(a) Field List

- Field Name | Field Label | Field Type
- cstDescriptionTX | Description | Text
- cstNameTX | Name | Text
- cstRollupQuantityCR | Quantity | Classification Rollup

The fields in the Soda Containers record are similar. The field involved in the straight roll up is also named RollupQuantity. This field is shown below:

(b) Field Properties

- Section: `General`
- Field Type: Classification Rollup

- Name: cstRollupQuantityCR
- Label: `Quantity`
- Description: _____
- Purpose: _____
- Required: ☐
- Do not Auto Populate: ☐
- Result Column: ☐
- Mobile Field: ☐
- Read Only: ☑
- Rollup Source: `Data Entry`
- Display Mask: `#.####`
- Storage Precision: `12`
- Rounding Rule: `Half Up`
- Root Classification: `each`
- Formula: ☐
- Threshold Source Attribute

Now suppose that many of the cans and bottles in the collection are brands such as A&W and Diet Coke that are owned by The Coca-Cola Company. In addition to knowing the total number of cans and bottles, we also want to know how many of the cans or bottles are brands that are owned by The Coca-Cola Company.

We have already taken the first step towards doing this. Records created from the Glass Soda Bottle, Plastic Soda Bottle, or the Soda Can business objects have a field named Brand. The Brand field is a Classification field with the root classification of Brands.

Since we already have a way to know the brand of each can and bottle, we just need a way to tell if a particular brand belongs to The Coca-Cola Company. One way is to organize the brands so that those that belong to The Coca-Cola Company have a common parent in the Classification hierarchy, as shown below:

(a) Hierarchy of The Coca-Cola Company Brands

- ⊖ Coca-Cola (Brands)
- A&W (Brands)
- Barq’s (Brands)

- Caffeine-free Barq’s (Brands)
- Caffeine-free Coca-Cola (Brands)
- Caffeine-free Diet Coke (Brands)
- Cherry Coke (Brands)
- Coca-Cola (Brands)
- Coke II (Brands)
- Diet A&W (Brands)
- Diet Barq’s (Brands)
- Diet Cherry Coke (Brands)
- Diet Coke (Brands)

The next steps are as follows:

- Have a field in the Soda Bottles record that contains the total number of bottles in the collection. And another field in the Soda Bottles record that contains the number of bottles in the collection that are a brand owned by The Coca-Cola Company.
- Have a field in the Soda Containers record that contains the total number bottles in the collection. And another field in the Soda Containers record that contains the number of bottles in the collection that are a brand owned by The Coca-Cola Company.

To have a field that contains a total of just cans or bottles that are Coca-Cola brands, we need a Classification Rollup field with the Rollup Source property set to `Classification`. Because a child record can participate in a straight rollup or a classification rollup but not both, we also need a Classification Rollup field with the Rollup Source property set to `Classification` for the totals that reflect all brands.

If a Classification Rollup field’s Rollup Source is `Classification`, it ignores any child records that contain Classification Rollup fields. We need to remove the Classification Rollup field from the Glass Soda Bottle and the Soda Can business objects, and replace the Classification Rollup field named Rollup Quantity with a Number field named Quantity. The example below lists the fields in the Glass Soda Bottle business object after we have done this.

(b) Field List

- Field Name | Field Label | Field Type
- cstBrandCL | Brand | Classification
- cstDescriptionTX | Description | Text

- cstHasTwistOffTopBL | Has Twist-Off Top | Boolean
- cstNameTX | Name | Text
- cstRollupQuantityCR | Quantity | Classification Rollup

We may avoid unnecessary work by having the Classification Rollup fields roll up numbers from the child record's Quantity rather than getting it directly from a field. This is because in the future we might want to add a different kind of record to the hierarchy that does not have the same fields as the records we are using now. By rolling the numbers up as the child's Quantity, we can add additional records later that have different fields and not have to spend time changing anything but the new child records.

To make the numbers roll up through the Quantity property of Glass Soda Bottle records, set it up in the business object's mapping properties, as shown below:

(c) Mapping Properties

- Name: **`Name`** [Add][Delete]
- Cost: [Find][+][Find][+][Find][+][Find]
- Quantity: **`Quantity`**
- Image
- Conversion Group
- Exchange Date
- Control Number: ☐ Based on Prefix
- Prefix: [Find][Add][Delete]
- Suffix: [Find][Add][Delete]
- Start With: `1`
- Delimiter

To roll up the total number of bottles and the number of Coca-Cola bottles to the Soda Bottles record, the Soda Bottles business object needs two Classification Rollup fields. The following shows the fields in the Soda Bottles business object.

(d) Field List

- Field Name | Field Label | Field Type
- cstCocaColaQuantityCR | Coca-Cola Quantity | Classification Rollup
- cstNameTX | Name | Text
- cstRollupQuantityCR | Quantity | Classification Rollup

The two Classification Rollup fields are named RollupQuantity and CocaColaQuantity. The details of the CocaColaQuantity field are shown below:

(e) Field Properties

- Section: `General`
- Field Type: Classification Rollup
- Name: cstCocaColaQuantityCR
- Label: Coca-Cola Quantity
- Description: _____
- Purpose: _____
- Required: ☐
- Do not Auto Populate: ☐
- Result Column: ☐
- Mobile Field: ☐
- Read Only: ☑
- Rollup Source: `Classification`
- Rollup Type: `Quantity`
- Display Mask: `#.####`
- Storage Precision: `12`
- Rounding Rule: `Half Up`
- Root Classification: `Coca-Cola`
- Formula: ☐
- Threshold Source Attribute

The details of the Coca-Cola Quantity field most relevant to the rollup are that its Rollup Source property is `Classification`, its Rollup Type property is `Quantity`, and its Root Classification is `Coca-Cola`. When a rollup happens, it will compute the total of the Quantity of all child records that have a value of or under Coca-Cola in a Classification field. Since the Glass Soda Bottle business object only has the Classification field named Brand, only bottles of a brand owned by The Coca-Cola Company will be included in the total.

The details of the RollupQuantity field are shown below:

(f) Field Properties

- Section: `General`
- Field Type: Classification Rollup
- Name: cstRollupQuantityCR
- Label: `Quantity`
- Description: _____
- Purpose: _____
- Required: ☐
- Do not Auto Populate: ☐
- Result Column: ☐
- Mobile Field: ☐
- Read Only: ☑
- Rollup Source: `Classification`
- Rollup Type: `Quantity`
- Display Mask: `#.####`
- Storage Precision: `12`
- Rounding Rule: `Half Up`
- Root Classification: `Brands`
- Formula: ☐
- Threshold Source Attribute

The important difference between the Rollup Quantity field and the Coca-Cola Quantity field is that Root Classification is set to `Brands`. Since this is the root classification for all brands, the Quantity from all child records will be totaled into this field.

The last part of the hierarchy to examine is the Soda Containers record. We want to roll up two sets of numbers into this record: The all-brands total and the Coca-Cola total for soda cans. We also want to roll up similar totals for soda bottles.

Use classification to roll up these totals for cans into the Soda Containers record. This is almost the same thing we did to roll up totals for glass bottles into the Soda Bottles record.

We cannot use classification to roll up the totals for bottles from the Soda Bottles record to the Soda Containers record because in their totaled form the numbers are not connected to a classification. However, we can use a straight rollup to roll up bottle totals from the Soda Bottles record to the Soda Containers record. The only thing to do differently is make the Root Classification of the appropriate Classification Rollup fields in the Soda Containers record be the same as the Root Classification of the corresponding fields in the Soda Bottles record, as shown below:

(g) Field List

- Field Name | Field Label | Field Type
- cstCanCocaColaQuantityCR | Can Coca-Cola Quantity | Classification Rollup
- cstCocaColaQuantityCR | Coca-Cola Quantity | Classification Rollup
- cstNameTX | Name | Text
- cstRollupQuantityCR | Quantity | Classification Rollup

The Can Coca-Cola Quantity field and the Can Quantity field are used to roll up totals for cans. The Coca-Cola Quantity and the Rollup Quantity fields are used to roll up bottle totals from the Soda Bottles record. Below are the details of the Coca-Cola Quantity field:

(h) Field Properties

- Section: `General`
- Field Type: Classification Rollup
- Name: cstCocaColaQuantityCR
- Label: Coca-Cola Quantity
- Description: _____
- Purpose: _____
- Required: ☐
- Do not Auto Populate: ☐
- Result Column: ☐
- Mobile Field: ☐
- Read Only: ☑
- Rollup Source: `Roll Up`
- Display Mask: `#.####`
- Storage Precision: `12`

- Rounding Rule: `Half Up`
- Root Classification: `Coca-Cola`
- Formula: ☐
- Threshold Source Attribute

Notice that the Coca-Cola Quantity field is a straight rollup with its Root Classification set to `Coca-Cola`. The Root Classification is set to `Coca-Cola` so that it will match the Coca-Cola Quantity field in the Soda Bottles record. Similarly, the Rollup Quantity field has its Root Classification set to `Brands` so that it will match the Rollup Quantity field in the Soda Bottles record.

Financial Transactions

In IBM TRIRIGA, a financial transaction is a way of saving numbers that are the result of an action or a sequence of actions. Because of their level of complexity, this chapter begins by explaining what financial transactions are at a conceptual level.

After describing the IBM TRIRIGA Application Platform's concept of a financial transaction, this chapter explains how to create financial transactions and use them to save numbers. This is followed by a discussion of the two uses for the numbers saved by a financial transaction:

- The results of a financial transaction can be rolled up into records that are not part of a hierarchy.
- The results of a financial transaction are relatively easy to share with accounting programs, inventory management programs, and other programs that measure the cumulative effect of actions over time.

What is a Financial Transaction?

The words "financial transaction" sound meaningful. They may suggest a way of modeling what accountants call a financial transaction. However, a financial transaction in IBM TRIRIGA is not quite what it sounds like. A financial transaction in IBM TRIRIGA saves one or more numbers. These numbers may be the total of other numbers. A financial transaction may correspond to just part of what accountants call a financial transaction.

The number saved by a financial transaction is usually associated with a financial entity or accounting code. For example, it may be associated with the total of the line items on an invoice or with a cost code. These are not mutually exclusive. Even though it is called a financial transaction, the number saved by a financial transaction does not have to be associated with anything that is financial in nature. For example, it may correspond to the total weight of items placed in a shipping container. The number saved by a financial transaction does not necessarily represent an entire transaction nor is it necessarily financial in nature.

A financial transaction saves a number or a set of numbers. Every number saved by a financial transaction is kept in a collection of numbers that includes every number ever saved by a financial transaction. When a number is saved in this collection, it is saved along with information about the transaction, including when the transaction happened and what kind of transaction it was. Because the numbers are saved with this transaction information, it is possible to select a set of numbers in this collection that were saved by transactions that fit certain criteria, for example, numbers that were saved by a particular kind of transaction during a particular range of time.

This ability to select numbers based on their transaction information is used in two ways:

- It is possible to use the total of a selected set of numbers as the value of a field. This is a kind of rollup that can be used with records that are not organized into a hierarchy. The details of how this works are discussed in Financial Rollup.
- It is possible to use the transaction information to select numbers to be imported by external programs that run outside IBM TRIRIGA, such as accounting programs and inventory management programs.

To save a number with a financial transaction:

- Create a financial transaction record from the Financial Transaction business object in the System module. This is usually done by a workflow.
- Specify information about the transaction by setting the values of the financial transaction record's fields and by associating the financial transaction record with other records related to the transaction.
- Perform a Post Transaction action on the financial transaction record, causing the number and other transaction details specified in the financial transaction record to be saved.

A financial transaction stores the following properties with the number it saves:

Property Name	Description
System Date	The date and time that the Post Transaction action was performed on the financial transaction record.

Property Name	Description
Transaction Date	The real world time and date when the transaction is considered to have happened. Unless transactions are immediately posted when they actually happen, this is distinct from the system date. This information comes from a field in the financial transaction record.
Conversion Group	If the number being saved has a unit of measure that is `Currency`, you can arrange for the number to be converted to other currencies when it is saved. If a conversion group is specified, it means that conversion to other currencies should be done. The name of a conversion group identifies the currencies the number should be converted to and the conversion rates to do the conversion. If a currency conversion rate does not exist for the posting date and time of the transaction, the most recent currency conversion rate is used. The use and management of currency conversion rates is discussed in Multiple Currencies.
Transaction Type	This is a string, normally chosen from a list, that identifies the type of transaction. Transaction types are in the list named Cost Transaction Types. It is common to add additional transaction types to the list to support an application. Examples of transaction types are: Budget Transfers, Inventory Adjustment, and Manifest Weight.
Primary Object	Most financial transactions contain total amounts or measurements for a real world entity that is represented by a record. For example, the purpose of a financial transaction could be to contain the value of a sales order or to contain the total weight of goods listed on a shipping manifest. The record that represents the real world entity for which a financial transaction contains a total or measurement is called the financial transaction record's Primary Object. There is no requirement that a financial transaction have a primary object. However, it is unusual for a financial transaction not to have a primary object.

Property Name	Description
Reference Object	Some financial transactions that have a primary object also need to be associated with another object. For example, suppose your business uses purchase orders that have a different delivery date for each line item. Following the rules of accounting, each line item must be entered in your company's books when it is delivered. To make this happen, create financial transaction records that correspond to individual line items. Since each of these financial transaction records contribute to the value of a line item, the record that represents the line item is the financial transaction record's primary object. To be able to easily retrieve all financial transaction amounts that are for the same purchase order, associate the financial transactions with the record that represents the purchase order. The object with which a financial transaction record is associated is called its Reference Object.
Reverse Transaction Flag	If a transaction is reversed, it means that the number it posted should be ignored when computing totals. If the Reverse Transaction Flag is true, it means that this transaction reverses other transaction(s). If the Reverse Transaction Flag is set, the financial transaction's primary object should be specified because a financial transaction with its reverse flag set is matched up with the transactions it reverses by their primary object. A financial transaction that has its reverse flag set reverses every previous financial transaction that has the same primary object. This is most convenient if the real world entity represented by the primary object is being cancelled or replaced. This mechanism is not helpful for selectively reversing previous financial transactions with the same primary object. If a reverse financial transaction is being posted because the entity represented by the primary object is being cancelled, there is no need to associate any number(s) with the reverse transaction. When the transaction is posted, it will mark all previously posted financial transactions that have the same primary object as having been reversed.

Transaction Types

The numbers that a financial transaction saves become associated with a financial transaction indirectly by putting them in records that appear in a financial transaction record's `CostTransactionCodes` multiple record section. The records in this multiple record section contain the numbers that are added up to get the number that is recorded when the financial transaction is posted.

A financial transaction's transaction type does not come directly from the financial transaction record. Instead, a transaction type is stored in each of the multiple record section's records, along with the numbers to be added.

If all of the records in the multiple record section have the same transaction type, that is the transaction type recorded for the financial transaction when it is posted. In this case, the number recorded for the financial transaction is the total of the numbers in all records in the multiple record section. For example, suppose that a financial transaction record's multiple record section contains records that correspond to items put in a shipping container. Also suppose that the transaction type in all records in the multiple record section is Manifest Weight. When the financial transaction is posted, the number recorded will be the total weight of the items in the shipping container. The posted transaction's type will be Manifest Weight.

If a Financial Transaction record's multiple record section contains records that do not have the same transaction type, the records in the multiple record section are organized by their transaction type. A total is computed for each transaction type. Each total is recorded as if it came from a separate financial transaction. Each recorded transaction has a different transaction type and amount; all other information in the recorded transactions is the same. For example, consider a financial transaction that reflects a simple retail sales order. Ignoring sales taxes and other complications, the records in the financial transaction record's multiple record section would probably contain two different transaction types. The multiple record section would contain records that correspond to the items being purchased. The transaction type for these records might be Inventory Depletion. The multiple record section would also contain records that correspond to the payment received with the sales order. The transaction type for these records might be Payments Received. When this financial transaction is posted, it will be posted as if it were two financial transactions. One has the transaction type Inventory Depletion and the other has the transaction type Payments Received. The two transactions share the same primary object and other information.

The reasons for packing what appear to be multiple financial transactions into one financial transaction record include:

- It is convenient. Only one financial transaction record needs to be created. Only one Post Transaction action needs to be performed.
- When multiple financial transactions are recorded from a single financial transaction record, the IBM TRIRIGA Application Platform guarantees that either all of the transactions are recorded or none of the transactions are recorded. This is important.

  This all-or-nothing outcome of posting a financial transaction is very important because it ensures that numbers recorded from financial transactions will never be inconsistent with reality because of a computer failure. This all-or-nothing property ensures that the financial transaction for the sales order example above is never only half posted. The total for the goods sold will never be posted without also posting the amount of the payment that was received for the goods.

When a Post Transaction action is performed on a financial transaction record with a multiple record section containing multiple transaction types, either a transaction is recorded for each transaction type or no transactions are recorded at all. This all-or-nothing property of a Post Transaction action has a special name. When the outcome of an action will either be fully recorded or not recorded at all, the action is *atomic*.

Cost Codes

It is often the case that you want to associate the number recorded by a financial transaction with an accounting code. Records created from business objects in the Cost Code module are used to describe accounting codes.

To associate numbers with a financial transaction record, put the records that contain the numbers in the financial transaction record’s multiple record section. The records that go in the multiple record section are created from a business object defined in the Cost Code module. This means that every number recorded by posting a financial transaction is associated with an accounting code.

Many kinds of accounting codes are organized into a hierarchy. Consider the hypothetical hierarchy of cost codes below:

- ⊕ Material Charge
- ⊖ Service Charge
- ⊖ Labor Charge

- Technician 1
- Technician 2
- Technician 3
- Supplies Charge
- ⊖ Surcharges
- After Hours Charge
- Rush Charge

Because the cost codes are in a hierarchy, if a number is associated with the cost code Technician 2, it also is associated with Labor Charge and Service Charge. It is associated with these cost codes because they are above Technician 2 in the hierarchy. A total of all charges that apply to Service Charge or to Labor Charge includes all charges associated with the Technician 2 cost code.

When a financial transaction is posted, its details are recorded with a method intended to minimize the need to interpret the data using information that may change over time. The organization of accounting codes can change over time. It would not be good to assume that between the time a transaction was posted and a report is generated, the structure of the accounting codes has not changed.

To avoid the need for this sort of assumption, posted financial transactions are recorded with their associated accounting code and with every accounting code that occupies a higher position in the hierarchy of accounting codes. For example, if a financial transaction is posted that is associated with the Technician 2 cost code, the recorded financial transaction will include the Technician 2, Labor Charge, and Service Charge cost codes.

For the purposes of this discussion, we will call an accounting code that is directly associated with a financial transaction before it is posted a *detail accounting code*. Accounting codes that are recorded with the posted financial transaction because they occupy a position in the hierarchy above the detail accounting code we will call *summary accounting codes*.

When a financial transaction is posted, only one detail accounting code is recorded for a transaction. If the numbers in a financial transaction record’s multiple record section are associated with different accounting codes, when the financial transaction is posted, it is posted as if there were multiple financial transactions, each containing only one of the accounting codes as the detail accounting code.

This is similar to what happens when different transaction types are associated with the numbers in a financial transaction record's multiple record section. If the numbers in a financial transaction record's multiple record section are associated with different transaction types, when the financial transaction is posted, it is posted as if there were multiple financial transactions, each containing only one of the transaction types.

Neither of these statements tells the complete story. Actually, if the numbers in a financial transaction record's multiple record section are associated with different combinations of transaction type and accounting code, when the financial transaction is posted, it is posted as if there were multiple financial transactions, each containing only one combination of transaction type and accounting code.

Creating and Posting Financial Transactions

This section discusses the details of how to create a financial transaction and then post it. The first step is to create a new financial transaction record. Once the financial transaction record is created, the next step is to set the values of its fields. The fields of a Financial Transaction record are as follows:

Field Name	Description
BusinessObjectID	The value of this field is set when a Post Transaction action is performed on a financial transaction record. The value is a number that identifies the record that is the financial transaction's primary object.
CompanyID	The value of this field is set when a Post Transaction action is performed on a financial transaction record. The value is a number that identifies the platform environment in which the record that is the financial transaction's primary object was created.
ID	This is a control number that is generated when a financial transaction record is created. It serves as the name of the record.
ConversionGroup	The value of this field is from the list named Conversion Group. The Currency Conversions tool associates a name from this list with every currency conversion rate. When the financial transaction is posted, if the unit of measure is a currency and a value is specified for `ConversionGroup`, converted amounts are saved with the financial transaction. If a currency conversion rate does not exist for the posting date and time of the transaction, the most recent currency conversion rate is used. This is discussed in more detail in Multiple Currencies.

Field Name	Description
CostTransactionCodes Amount Total	This is a total of all the amounts in the cost codes. This field is read-only. Its value is computed automatically by the IBM TRIRIGA Application Platform.
Description	This field contains a description of the transaction.
ReverseTransactions	If this is true, it means that the transaction reverses another transaction.
TransactionDate	The time and date in this field are recorded as the transaction date. If no value is specified, the system date is used as the transaction date when the transaction is posted.
TransactionID	This field is not used for anything.
UserId	The IBM TRIRIGA Application Platform sets the value of this field when a PostTransaction action is performed on a financial transaction record. It is a number that identifies the MyProfile record of the user that initiated the Post Transaction action.

To establish the record that will be used as a financial transaction's primary object, create an association from the financial transaction record to the record that will be the primary object. The name of the association must be `Financial Primary Object`. Remember that the platform will not allow you to create an association unless there is a corresponding association definition.

To establish the record that will be used as a financial transaction's reference object, create an association from the financial transaction record to the record that will be the reference object. The name of the association must be `Financial Reference Object`.

`costTransactionCodes`

A financial transaction record contains a multiple record section named `costTransactionCodes`. This `costTransactionCodes` multiple record section contains the details of the financial transaction. Each record in the `costTransactionCodes` multiple record section is created from a business object named Cost Code Container that is part of the Cost Code module. The Cost Code Container business object is a *link* business object.

The `costTransactionCodes` multiple record section has its Live Link property set to false. This multiple record section that contains link records and has its Live Link property set to false results in a multiple record section that is used as follows:

- You can add any record to the `costTransactionCodes` multiple record section that is in the Cost Code module's hierarchy. The record you choose from the hierarchy should be the one that represents the detail accounting code that you want to associate with a number in the financial transaction.
- When you add a record to the `costTransactionCodes` multiple record section, its connection with the multiple record section is indirect. A new Cost Code Container record is automatically created by the multiple record section. The Cost Code Container record is linked to the record being added to the multiple record section. The Cost Code Container record is actually what is then added to the multiple record section.
- Initially, the newly-added Cost Code Container record has the same values in its fields as the underlying record it is linked to.
- Because the multiple record section has its Live Link property set to false, any changes made to the records in the multiple record section will not affect their underlying linked records.

Before you create a financial transaction, the Cost Code module's hierarchy should contain records that correspond to all accounting codes that the financial transaction will use. If no business object exists in the Cost Code module that is suitable for representing a particular accounting code, add a suitable business object to the Cost Code module. Any new business object added to the Cost Code module should be a stand-alone business object that includes all fields in the Cost Code business object.

After adding a record to the `costTransactionCodes` multiple record section, set its Amount field to the correct detail amount and its `CostTransactionType` field to the correct transaction type.

These modifications to records in the financial transaction record's multiple record section affect only the link records in the multiple record section. The underlying Cost Code records are unaffected; you can use them in the multiple record section of other financial transaction records.

There should only be one currency or other unit of measure used in a financial transaction record's multiple record section. The total for a financial transaction is recorded with one unit of measure or primary currency. If the records in the multiple record section contain different units of measure or currencies, the IBM TRIRIGA Application Platform will select one of the units of measure or currencies arbitrarily.

It will attempt to convert the other units of measure or currencies to the unit of measure or currency it selected.

Financial Transaction State Transition Family

The actions that can be performed on a financial transaction record are governed by the state transition family named Financial Transaction. After a Create action has been performed on a financial transaction record, it enters a state named New. While in the New state, in addition to OK and Apply, there are two other actions that can be performed on a financial transaction record:

Action Name	Description
PostTransaction	This action causes the data contained in a financial transaction record to be posted. The details of what happens when a financial transaction is posted are discussed in various parts of this chapter.
Retrieve Data	This action is defined to support an obsolete feature for old applications. It should not be used for any new applications.

There is special processing logic within the IBM TRIRIGA Application Platform associated with the Post Transaction action. This special logic is what posts the financial transaction. After posting the financial transaction, the special logic sets the state of the financial transaction record to a state named Posted.

Multiple Currencies

If a financial transaction's unit of measure is a currency and a conversion group is specified in the financial transaction record, all applicable conversion rates in the specified conversion group are used to convert from the primary currency to other currencies.

The purpose for this is similar to the purpose of recording summary accounting codes with a posted financial transaction. After the transaction is recorded, you do not want to guess what currency conversion rates were in effect at the time of the transaction. By storing all relevant converted amounts with the transaction, there is no need to do any currency conversions on the transaction after it is posted.

The IBM TRIRIGA Application Platform maintains a collection of currency conversion rates for converting between different currencies. There are two mechanisms for managing currency conversion rates:

- Manually by using the Currency Conversions tool. The Currency Conversions tool is described in the "Data Types" chapter of *Application Building* under the heading "Money".

- By an external application. This is done via IBM TRIRIGA Connector for Business Applications. The details are discussed in the *IBM TRIRIGA Connector for Business Applications 3 Technical Specification* book.

There are properties that the IBM TRIRIGA Application Platform associates with each currency conversion rate. If the unit of measure for the primary number being posted for a financial transaction is a currency, the platform uses these properties to decide which conversion rates should be applied to the financial transaction's primary value. The properties of a currency conversion rate are as follows:

Property Name	Description
From currency	The currency that the conversion rate is intended to convert from.
To currency	The currency that the conversion rate is intended to convert to.
Start date and time	The first date and time that the currency conversion rate applies to.
End date and time	The last date and time that the currency conversion rate applies to.
Conversion Group	The name of a group of conversion rates that will be applied to the same financial transactions. If you want to apply the same conversion rate to financial transactions that specify different conversion groups, you need to store two conversion rates that are identical except for having different conversion groups.

Whichever mechanism you use to manage currency conversion rates, make sure the start and end dates and times for each currency are the same for every currency conversion rate. This practice can avoid a variety of confusing situations.

Database Transaction Tables

When a financial transaction is posted, its details are recorded by inserting them in rows of relational database tables. After the details of a financial transaction are stored in the database, you can:

- Import data from the database to external applications that use numbers accumulated from financial transactions to measure some aspect of your business or organization. For example, you may import the numbers into an accounting program or an inventory management program.
- Use fields of type Financial Rollup to accumulate or summarize transaction results for applications that run within the IBM TRIRIGA Application Platform.

If you are planning to import transaction data into external applications, you or somebody you work with will need to understand how to use relational databases in general and also understand the details in this part of this chapter. If you are planning to use recorded transaction data only with Financial Rollup fields, it is not necessary for you to understand how to use relational databases. However, you should skim this section to understand the organization of the data recorded from posted financial transactions.

Financial Rollup fields are discussed in detail in Financial Rollup.

When the data for a financial transaction is stored in a relational database, the data is stored in tables that are part of a schema named `TRIDATA`. The person who administers the IBM TRIRIGA Application Platform for your organization should be able to provide you with the details of accessing the database and the `TRIDATA` schema.

The IBM TRIRIGA Application Platform uses the following tables in the database to contain data from posted financial transactions:

- `BUDGET_TRANSACTION`
- `BUDGET_CODES`
- `BUDGET_CURRENCIES`

When a Post Transaction action is performed on a financial transaction record, financial transaction data is recorded for each combination of transaction type and accounting code that appears in the multiple record section of the financial transaction record. The transaction type is the value in the field named `CostTransactionType`. The accounting code is the name of the record specified in the Publish tab of the record's business object.

When a Post Transaction action is performed on a financial transaction record, for each combination of transaction type and accounting code, a row is added to the `BUDGET_TRANSACTION` table. The columns of the `BUDGET_TRANSACTION` table are as follows:

Column Name	Column Type	Description
TRANSACTION_ID	NUMBER	Uniquely identifies this transaction within this table. This is the table’s primary key.
TRANSACTION_TYPE	VARCHAR	The transaction type for this transaction.
DESCRIPTION	VARCHAR	A description of the transaction.
TRANSACTION_DATE	DATE/TIME	The date and time when the transaction is considered to have happened. The value in this column comes from the `TransactionDate` field of the financial transaction record that was posted to create this row, unless no value was specified for the `TransactionDate` field. If no value was specified for the `TransactionDate` field, this column is set to the same value as the `SYSTEM_DATE`.
SYSTEM_DATE	DATE/TIME	The date and time when this transaction was posted.
COMPANY_ID	NUMBER	Uniquely identifies the IBM TRIRIGA Application Platform environment from which this transaction was posted.
PROGRAM_ID	NUMBER	Reserved.
PROJECT_ID	NUMBER	Uniquely identifies the project internally associated with the financial transaction record from which this transaction was posted.
BO_ID	NUMBER	Uniquely identifies the business object used to create the record that is this financial transaction’s primary object. If this financial transaction does not have a primary object, the value is 0.
OBJECT_ID	NUMBER	Uniquely identifies the record that is this financial transaction’s primary object. If this financial transaction does not have a primary object, the value is 0.
OBJECT_VERSION	NUMBER	Reserved.

Column Name	Column Type	Description
MODULE_ID	NUMBER	Uniquely identifies the module that contains the business object used to create the record that is this financial transaction's primary object. If this financial transaction does not have a primary object, the value is 0.
ORGANIZATION_ID	NUMBER	The financial transaction record that was used to post this transaction has a field named `OrgName`. If `OrgName` referred to an Organization record when the transaction was posted, this column contains the number that uniquely identifies the Organization record. If `OrgName` did not refer to an Organization record when the transaction was posted, this column contains 0.
GEOGRAPHY_ID	NUMBER	The financial transaction record that was used to post this transaction has a field named `GeographyName`. If `GeographyName` referred to a Geography record when the transaction was posted, this column contains the number that uniquely identifies the Geography record. If `GeographyName` did not refer to a Geography record when the transaction was posted, this column contains 0.
USER_ID	NUMBER	Uniquely identifies the My Profile record associated with the user that initiated the Post Transaction action that posted this transaction.
REF_OBJECT_ID	NUMBER	Uniquely identifies the record that is this financial transaction's reference object. If this financial transaction does not have a reference object, the value is 0.
REF_OBJECT_VERSION	NUMBER	Reserved.
REVERSE_FLAG	NUMBER	The value in this column is normally 0. If this transaction has been reversed by another transaction, the value in this column is 1.
REF_MODULE_ID	NUMBER	Uniquely identifies the module that contains the business object used to create the record that is this financial transaction's reference object. If this financial transaction does not have a reference object, the value is 0.

Column Name	Column Type	Description
REF_BO_ID	NUMBER	Uniquely identifies the business object used to create the record that is this financial transaction's reference object.
REVERSE_DATE	DATE/TIME	When a row is inserted, the value in its `REVERSE_FLAG` column is 0 and this column's value is the same as the `SYSTEM_DATE` column. If the financial transaction represented by this row is reversed, the value in the `REVERSE_FLAG`column is changed to 1 and the value of this column becomes the date and time when the transaction was reversed.
LOCATION_ID	NUMBER	The financial transaction record used to post this transaction has a field named `LocationName`. If `LocationName` referred to a Location record when the transaction was posted, this column contains the number that uniquely identifies the Location record. If `LocationName`field did not refer to a Location record when the transaction was posted, this column contains 0.

When the data for a financial transaction is posted, the recorded data includes the detail accounting code and all summary accounting codes above it in the hierarchy. Since a financial transaction has a one-to-many relationship with its accounting codes, the accounting codes are stored in a separate table named `BUDGET_CODES`. The `BUDGET_CODES` table is as follows:

Column Name	Column Type	Description
TRANSACTION_ID	NUMBER	Identifies the financial transaction this row is part of. This is the same number that appears in the `TRANSACTION_ID` column of the row in the `BUDGET_TRANSACTION` table that represents this financial transaction.

Column Name	Column Type	Description
CODE_TYPE	VARCHAR	The rows in this table correspond to two different kinds of records: ▪ The first kind of record is a record in the Cost Code module's hierarchy that represents an accounting code. ▪ The second kind of record is a record that has an association named `Rollup Associated Object` with the first kind of record. To distinguish between the two kinds of rows, this column can have these values: ▪ If the row corresponds to the first kind of record, this column's value will be the highest level summary accounting code in the hierarchy that is associated with the financial transaction. ▪ If the row corresponds to the second kind of record, this column's value will be `Associated Object`. You can tell which row represents the associated first kind of record because it will have the same values in its `TRANSACTION_ID` and `CODE_SEQ` columns as this row.
CODE_SEQ	NUMBER	Rows in this table correspond to a record created from a business object in the CostCode module. The record either represents the accounting code directly associated with the financial transaction identified in the `TRANSACTION_ID` column or it is a record above the accounting code in the Cost Code module's hierarchy. The number in this column indicates what level in the Cost Code module's hierarchy the corresponding record occupies. Records directly under the root of the Cost Code module's hierarchy are at level 0 (zero). The records under that are at level 1 (one), and so on. For a given value in the `TRANSACTION_ID` column, rows in this table with the highest value correspond to the record that represents the detail accounting code associated with the financial transaction.
CODE_REF_ID	NUMBER	Uniquely identifies the record that this row corresponds to.
CODE_STRING	VARCHAR	The name of the record this row corresponds to.

Column Name	Column Type	Description
MAPPED_BUDGET_ CODE	VARCHAR	This column is obsolete and is no longer used.
STATUS	VARCHAR	Reserved.
SYSTEM_DATE	DATE/TIME	The same date and time as in the `SYSTEM_DATE` column of the row in the `BUDGET_TRANSACTION` table that represents this financial transaction.
USER_ID	NUMBER	The same number as in the `USER_ID` column of the row in the `BUDGET_TRANSACTION` table that represents this financial transaction.

If the unit of measure for a financial transaction is a currency, when it is posted the recorded data includes the amount in the primary currency and in any number of other currencies. Since there is a one-to-many relationship between a financial transaction and its amounts, the amounts are stored in a separate table named `BUDGET_CURRENCIES`. The `BUDGET_CURRENCIES` table is as follows:

Column Name	Column Type	Description
TRANSACTION_ID	NUMBER	Identifies the financial transaction this row is part of. This is the same number that appears in the `TRANSACTION_ID` column of the row in the `BUDGET_TRANSACTION` table that represents this financial transaction.
CURRENCY_CODE	VARCHAR	The name of the unit of measure or currency for the number in the `AMOUNT` column.

Column Name	Column Type	Description
CONVERSION_RATE	NUMBER	If the unit of measure named in the CURRENCY_CODE column is a currency, this column contains the conversion rate between the financial transaction's primary currency and the currency specified in the CURRENCY_CODE column of this row. The primary currency for a financial transaction is reflected in a row that has the financial transaction's TRANSACTION_ID and the value 1 in the CONVERSION_RATE column. The number in the CONVERSION_RATE column is the number that was multiplied by the AMOUNT denominated in the primary currency to calculate the AMOUNT in this row.
AMOUNT	NUMBER	Contains the number saved by this financial transaction in the unit of measure named in the CURRENCY_CODE column.
SYSTEM_DATE	DATE/TIME	The same date and time as in the SYSTEM_DATE column of the row in the BUDGET_TRANSACTION table that represents this financial transaction.
USER_ID	NUMBER	The same number as in the USER_ID column of the row in the BUDGET_TRANSACTION table that represents this financial transaction.

If you will be importing data to other applications, it is possible that you will need to access data that is not in the transaction data. If this is the case, these are the ways to access the additional data:

- Use Eclipse Business Intelligence Reporting Tools (BIRT), which is described in the *IBM TRIRIGA Application Platform 3 Reporting User Guide* book.
- Use IBM TRIRIGA Connector for Business Applications to run a query and not directly access the relational database.
- Use the relational database that the IBM TRIRIGA Application Platform uses to store data in your environment.

You can run a query through IBM TRIRIGA Connector for Business Applications. IBM TRIRIGA Connector for Business Applications is described in the *IBM TRIRIGA Connector for Business Applications 3 Technical Specification* document. This tool gives a great deal of flexibility in how data for import is presented to the importing program. The flexibility comes from the fact that using IBM TRIRIGA Connector for

Business Applications usually involves writing a custom program. A custom program is usually able to present data in whatever form is desired.

Another option is to fetch the data directly from the database. Look through descriptions of columns in the `BUDGET_TRANSACTION` and `BUDGET_CODES` tables. You will notice columns that contain a number that identifies a record, business object, or module. You can use these numbers to find data in the appropriate database tables.

Financial Rollup

A field of type Financial Rollup contains a number that is the total of amounts recorded in posted transactions that fit specified criteria. The uses for Financial Rollup fields include:

- Keeping a total of the transactions posted with a particular purchase order or line item as the primary object.
- Keeping a total of the transactions posted with a particular accounting code.
- Keeping a total of transactions posted as part of a particular project.

For the amount in a posted financial transaction to be included in a total in a Financial Rollup field, the field must be in a record referenced by the financial transaction. When a financial transaction is posted, its amount is added to the total in Financial Rollup fields if the field is in a record referenced by the financial transaction and the transaction meets the criteria specified for the field.

A financial transaction can reference a record in several ways:

- A record can be referenced as a financial transaction’s primary object.
- A record can be referenced as a financial transaction’s reference object.
- A Capital Project record can be referenced by a financial transaction as its project.
- A record in the Cost Code module’s hierarchy can be referenced as a financial transaction’s detail or summary accounting code.

Financial Rollup Fields

This section describes how to specify criteria for deciding which financial transactions are included in a Financial Rollup field’s total.

When editing the properties of a Financial Rollup field, the Data Modeler’s Property panel looks like the following:

(a) Field Properties

- Section: `General`
- Field Type: Financial Rollup
- Name: triBudgetCurrentRollupFR
- Label: Budget Current
- Description: _____
- Purpose: _____
- Required: ☐
- Do not Auto Populate: ☐
- Result Column: ☐
- Mobile Field: ☐
- Read Only: ☑
- Use Custom UOM Precision and Mask: ☐
- Token: [Set Token]
- Hierarchy Object
- UOM Source Attribute
- Start Date Source
- End Date Source
- Threshold Source Attribute

Most of the properties in a Financial Rollup field are the same as when editing other kinds of fields and are described in detail in the “Data Types” chapter of *Application Building*. Seven properties appear that are specific to Financial Rollup fields. They are as follows:

Property Name	Description
Use Custom UOM Precision and Mask	This property specifies whether or not this field will use the properties for Storage Precision and Display Mask as they are defined on the UOM. If it is not checked, the platform will refer to the UOM’s Storage Precision and Display Mask. If it is checked, the Storage Precision and Display Mask can be overridden.

Property Name	Description
Token	Use the Token property to specify a set of criteria for which financial transactions will be included in the field's total. This set of criteria is called a *financial token*. A financial token can be shared by multiple Financial Rollup fields. One way to specify the token is to click the List icon ▾ and select an existing financial token to be used by the field. If you change your mind and want to create a new financial token, click the Clear icon ⊗ to clear the previous selection. The other way to specify the token uses the hyperlink next to the icons. If the text of the hyperlink is `Set Token`, clicking the hyperlink causes a Token form to pop up for creating a new financial token. If the text of the hyperlink is something else, it is the name of an existing token. In this case, clicking the hyperlink causes that token's form to pop up. The Token form is described in "Financial Tokens". When editing an existing financial token be especially careful if the financial token is shared with other field definitions. Any changes you make to the financial token affects all field definitions that share the financial token. Financial tokens are the main tool for selecting which financial transactions are included in a Financial Rollup field's total. Financial tokens are explained in greater detail in "Financial Tokens".

Property Name	Description
Hierarchy Object	Use the Hierarchy Object property to place an additional condition on which financial transactions are included in a Financial Rollup field’s total. The Hierarchy Object property restricts the financial transactions included in a Financial Rollup field’s total to only those associated with a particular accounting code. The value for the Hierarchy Object property is chosen from a drop-down list. The drop-down list always includes the choices `None` and `Self`. For Financial Rollup fields that are part of a business object in a hierarchy module, always choose `Self`. For Financial Rollup fields that are part of a business object that is not in a hierarchy module, the usual value is `None`. If the value for the Hierarchy Object is `None`, the system rolls up all financial transactions where the primary object or reference object is the same as the current record. If the value for the Hierarchy Object is defined (not `None`), the system rolls up all financial transactions association to that hierarchy and ignores any current content. If the Hierarchy Object is defined but there is no value at runtime, the system returns no result. You can restrict a Financial Rollup field to include only financial transactions associated with a particular accounting code if the field is part of a business object not in a hierarchy module. To do this, there must be a locator field in the same business object as the Financial Rollup field. The locator field must reference a record in the Cost Code module’s hierarchy. If there are any locator fields in the same business object as a Financial Rollup field, the drop-down list in the field’s Hierarchy Object property includes the names of the locator fields. Choose as the value of the Hierarchy Object property the name of the locator field that references a record in the Cost Code module’s hierarchy. The total in the Financial Rollup field will only include financial transactions associated with the accounting code represented by the record referenced by the locator field.

Property Name	Description
UOM Source Attribute	A user can be given the option of setting the currency UOM of the Financial Rollup fields they see in a form. There are several steps to making this work for the user, one of which is specifying the UOM Source Attribute property in the Financial Rollup field. The following must be accomplished: ▪ The business object must include a UOM field named `Currency`. ▪ In each Financial Rollup field, specify `Currency` in the UOM Source Attribute. Choose the value for the UOM Source Attribute from the drop-down list. The drop-down list includes the choice of `blank` or `Currency`. Selecting `Currency` allows the token to use any currency UOM field that is part of the same business object. Do this for each Financial Rollup field in the business object that the user sees in a form. ▪ Update each form to display the currency field in the General section (so the user can select a currency UOM) and to display the UOM for each Financial Rollup field (check the *Show UOM* check box). ▪ Have active Currency Conversion rates for each currency UOM available to the user. If a currency conversion rate does not exist for the posting date and time of the transaction, the most recent currency conversion rate is used. Then the user can override the currency UOM defined in the token, indicate a different currency UOM, and see the Financial Rollup fields in the selected currency UOM. The UOM Source Attribute specified must be part of the same business object as the Financial Rollup field and only applies to currency UOM fields. The UOM Source Attribute only affects the currency UOM displayed at runtime; currency values are still calculated and saved when the financial transaction is posted. If the UOM Source Attribute for a Financial Rollup field is blank, the currency users see is the currency UOM set in the Financial Rollup field's token at design time.

Property Name	Description
Start Date Source End Date Source	The Start Date Source and End Date Source properties can be used to make the start and end dates used in a financial rollup be based on fields in the record. This makes the date range dynamic. To set a dynamic date range, select the Start Date Source and End Date Source from the drop-down lists. The values in the lists are Date or Date and Time fields in the Financial Rollup field’s business object. An example of how the Start Date Source and End Date Source properties are used in the system is capturing financial cost transactions within the context of a fiscal period located in the parent/reference record. If these properties are blank, the system uses the dates in the fields’ financial token.

Property Name	Description
Threshold Source Attribute	When there is a value in the Threshold Source Attribut*e* property, this Financial Rollup field is a scored Financial Rollup field, meaning it will be scored (as described in “Comparison” in *Application Building*). The score is displayed as a red, yellow, or green image to the left of the Financial Rollup field’s value. The value of the Threshold Source Attribute property must be a locator field that references the Threshold business object. This locator field must be in either a General section or a one-to-one smart section. For a Financial Rollup field that is in a smart section, the linked locator field must reside on the referenced business object. The Threshold record defines the threshold ranges and contains a UOM value that is used to convert the scored value. Select the linked locator field in the Threshold Source Attribute drop-down list. The list only shows locator fields in the same business object that references the Threshold business object. A scored Financial Rollup field can be used in a form section, non-table smart section, table smart section, or a vertical table section. The score for the field is calculated during the rendering of the field and is not stored. The platform compares the UOM of the Financial Rollup field with that of the Threshold record. If a conversion is needed, the platform converts the Financial Rollup value before comparing it with the threshold value to calculate the score. During runtime, the system uses the threshold record that is currently selected by the linked locator fields. If the locator field does not have a value, the Financial Rollup field displays without a score. The score is updated when the tab is reloaded. This means that if the value of the Financial Rollup field, or the threshold record, or the locator field that links to the threshold record, is changed, the score will not update (if needed) until the tab is reloaded.

Financial Tokens

In a Financial Rollup field, clicking Set Token or the hyperlinked token name causes the system to display a financial token form. It looks like the following:

(a) Financial Token Form

- Group: `Group 1`
- Token Name: ______
- Date Type: Transaction Date
- Start Date: ______
- End Date: ______
- UOM Type: `--Any—`
- UOM: --Any--
- Transaction Type: `--Any--`

The first two fields in the form are used to determine the name of the financial token. The first part of a financial token's name comes from the list in the *Group* field. The second part of a financial token's name comes from what is entered in the *Token Name* field.

Type whatever name you like in the Token Name field. To help avoid naming conflicts between applications, the first part of the name is chosen from the list in the Group field.

If you are writing a new application, you may want to add a new name to the Group field's list. Clicking the New hyperlink to the right of the Group field causes a form to pop up in which you can add a new name to the Group field's list. This form looks like the following:

(b) Add a New Group

- Group Name: ______

After you have specified the name of a financial token, use the financial token in one of the following ways to restrict which financial transactions will be included in Financial Rollup field:

- You can restrict a Financial Rollup field to financial transactions from a specified range of time. This restriction can be based on either the financial transactions' transaction date or system date, depending on which radio button is selected.

If you do not specify a date in the *Start Date* field or the *End Date* field, there is no date-based restriction in the financial transaction unless specified in Start Date Source and End Date Source. If you specify a date in the Start Date field, only financial transactions with a date on or after the specified date will be included. If you specify a date in the End Date field, only financial transactions with a date on or before the specified date will be included. If you specify a date in both the Start Date field and the End Date field, only financial transactions with a date on or between those dates will be included.

- You can restrict a Financial Rollup field to financial transactions that have a number recorded with a particular unit of measure or currency. Use the *UOM Type* and *UOM* fields to do this.

  When the *UOM Type* is set to `Currency`, you can choose `SYSTEM_BASE` in the *UOM* field to specify that the UOM of the financial token is the system's base currency, which is the currency that is specified in the `BaseCurrency` property in the `TRIRIGAWEB.properties` file.

  Always specify a unit of measure or currency in every financial token. Financial Rollup fields do not have a unit of measure associated with them, so no conversion is performed if transactions with different units of measure are added into the same Financial Rollup field.

  By specifying a unit of measure in the financial token, you ensure that all numbers totaled in a Financial Rollup field have a consistent unit of measure or currency. If you have defined a source UOM field for the Financial Rollup field, the source UOM would take precedence over any UOM defined on the Financial Token.

- You can restrict a Financial Rollup field to financial transactions that have a specified transaction type with the *Transaction Type* field.

  Always specify a transaction type in every financial token to avoid mixing numbers from different transaction types in the same Financial Rollup field.

At the bottom of the financial token form are two buttons labeled *Update* and *Insert*. At any given time, one is enabled and the other is grayed out. If the Update button is enabled, clicking it updates the existing financial token. If the Insert button is enabled, clicking it creates a new financial token.

Updating Totals in Financial Rollups

When a financial transaction is posted, all Financial Rollup fields affected by the posted financial transaction are updated to include the number from the newly-posted financial transaction. The only time a Financial Rollup field is automatically updated is when a financial transaction is posted.

Normally, updating the totals in a Financial Rollup field only when a financial transaction is posted is sufficient to keep the totals in Financial Rollup fields correct. This is not sufficient if the criteria for including financial transactions in a Financial Rollup field change. There is no automatic mechanism to recalculate the total in a Financial Rollup field after a change in its criteria for including financial transactions in its total.

The circumstances in which the total in a Financial Rollup field can become incorrect include the following:

- If a Financial Rollup field’s financial token changes, the field’s total may be incorrect.
- If the value in the Hierarchy Object property of a Financial Rollup field’s definition changes, the field’s total may be incorrect.
- If the value in the Hierarchy Object property of a Financial Rollup field’s definition is the name of a locator field and the accounting code the locator field refers to changes, the field’s total may be incorrect.

If you anticipate that any of these will happen to a Financial Rollup field, provide for your application to explicitly force recalculation of totals in Financial Rollup fields. The IBM TRIRIGA Application Platform gives you a way to create a mechanism that will explicitly recalculate the totals in all Financial Rollup fields in a record. To create such a mechanism, configure a state transition to run some special logic that is intended for this purpose.

To configure a state transition to force the recalculation of a Financial Rollup field’s total, follow these steps:

- Navigate to the Data Modeler (described in the “Data Modeling” chapter of *Application Building*).
- Select the business object with the state transaction family that contains the state transition you want to configure.
- Click Tools and select BO State Transition. This causes a form to pop up that allows you to edit the state transition family.

- In the right panel, click the state transition you want associated with the special recalculation logic. This causes the color of the transition to change from gray to cyan. The fields that contain the state transition’s properties appear in the left panel. Type and apply the following:

(a) Transition Properties

- Action: cstRecalculate
- Label: Recalculate
- Class or Ejb Name: com.tririga.architecture.budget.ejb.BudgetHome
- Method Name: refreshAllTokens
- Apply the properties.

The next time a record goes through the configured state transition, the totals in all of its Financial Rollup fields will be recalculated.

Comparison of Rollup Methods

Since the IBM TRIRIGA Application Platform provides two different methods to roll up totals, you will have to choose which one to use when you build an application that rolls up totals. The chart below discusses the differences between the two methods.

Topic	Classification Rollup Method	Financial Rollup Method
Flexibility	Classification Rollup fields can only roll numbers up through records organized in a hierarchy.	Financial Rollup fields roll up numbers independent of the organization of records.
Complexity	The Classification Rollup mechanism is the simpler of the two. This makes it easier to be correct.	The Financial Rollup mechanism is a two step process, using financial transactions and Financial Rollup fields. Because more details and steps are involved, there are more opportunities to introduce bugs.
User Interface	The IBM TRIRIGA Application Platform has special support for viewing records in a hierarchy. This makes it easier for users to explore rolled up numbers.	There is no particular feature in the IBM TRIRIGA Application Platform that helps users see numbers rolled up using Financial Rollup fields.

Topic	Classification Rollup Method	Financial Rollup Method
Performance	If detail numbers change frequently and rolled up totals must always be kept up-to-date, the Classification Rollup mechanism will have a lot of overhead and perform slowly. Rolling up changed numbers one level of a hierarchy at a time is time consuming. Recomputing all rolled up numbers in a hierarchy is also time consuming. If detail numbers change frequently but rolled up numbers need be up-to-date only occasionally, the Classification Rollup mechanism may have better performance. In a case like this, roll up totals only when they need to be correct. No rollups are done when numbers change, so no overhead is associated with changes.	If rolled up numbers must always be up-to-date, using Financial Rollup fields is usually the best performing option. With a single operation, the posting of a financial transaction, only the fields that need to be updated are updated. If rolled up numbers only need to be up-to-date occasionally and changes to the detail data are frequent, the performance of Financial Rollup fields is less attractive. Financial Rollup fields are updated every time a financial transaction is posted. If the rolled up values seldom need to be updated and changes to the detail are frequent, a lot of time can be wasted keeping rolled up values up-to-date.
UOM and Currency	No unit of measure or currency is associated with Classification Rollup fields.	No unit of measure or currency is associated with a Financial Rollup field. However, a financial token can restrict the numbers that it totals so they all have the same currency or unit of measure.
History	Rolling numbers up through a hierarchy does not leave any record of how the rolled up values came to be what they are. The only clues are the current detail values.	The posted transactions recorded in the database can be used as an historical record of how every Financial Rollup field came to have its current value.
Data Sharing	Rolled up values can be shared with external applications using the same mechanisms available to share any other kind of data in the IBM TRIRIGA Application Platform.	In addition to the usual mechanisms for sharing data with external applications, data in posted transactions can be accessed directly through the underlying database without interacting with the IBM TRIRIGA Application Platform.

INDEX

Notices

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Privacy Policy Considerations

IBM Software products, including software as service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at www.ibm.com/privacy and IBM's Online Privacy Statement at www.ibm.com/privacy/details in the section entitled “Cookies, Web Beacons and Other Technologies” and the "IBM Software Products and Software-as-a-Service Privacy Statement" at www.ibm.com/software/info/product-privacy/.

Trademarks