IBM TRIRIGA Application Platform
3.7.0

Application Building for the IBM TRIRIGA Application Platform: Performance Framework

Note

Before using this information and the product it supports, read the information in “Notices” on page 45.

This edition applies to version 3, release 7, modification 0 of IBM® TRIRIGA® Application Platform and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Performance framework

IBM TRIRIGA Workplace Performance Management and IBM TRIRIGA Real Estate Environmental Sustainability provide viable solutions to help corporations strategically plan, manage, evaluate, and improve processes that are related to facilities and real estate.

IBM TRIRIGA performance framework is managed within TRIRIGA Workplace Performance Management and TRIRIGA Real Estate Environmental Sustainability, which include the following components:

- Data transform and fact table load services
- A metric builder that uses the Data Modeler
- A metric query engine
- Enhanced Report Manager for building metric reports
- Advanced portal features to render metric scorecards
- A series of prebuilt metrics, reports, and alerts that significantly improve the productivity of the many roles that are supported within TRIRIGA

Chapter 2. Data structures

TRIRIGA uses an extract, transform, and load (ETL) development environment as the mechanism for moving the data from business object tables to fact tables. In order to present the metrics, reports, scorecards, and other performance measures, the data must be in a form of fact tables and flat hierarchy tables that the reporting tools can process.

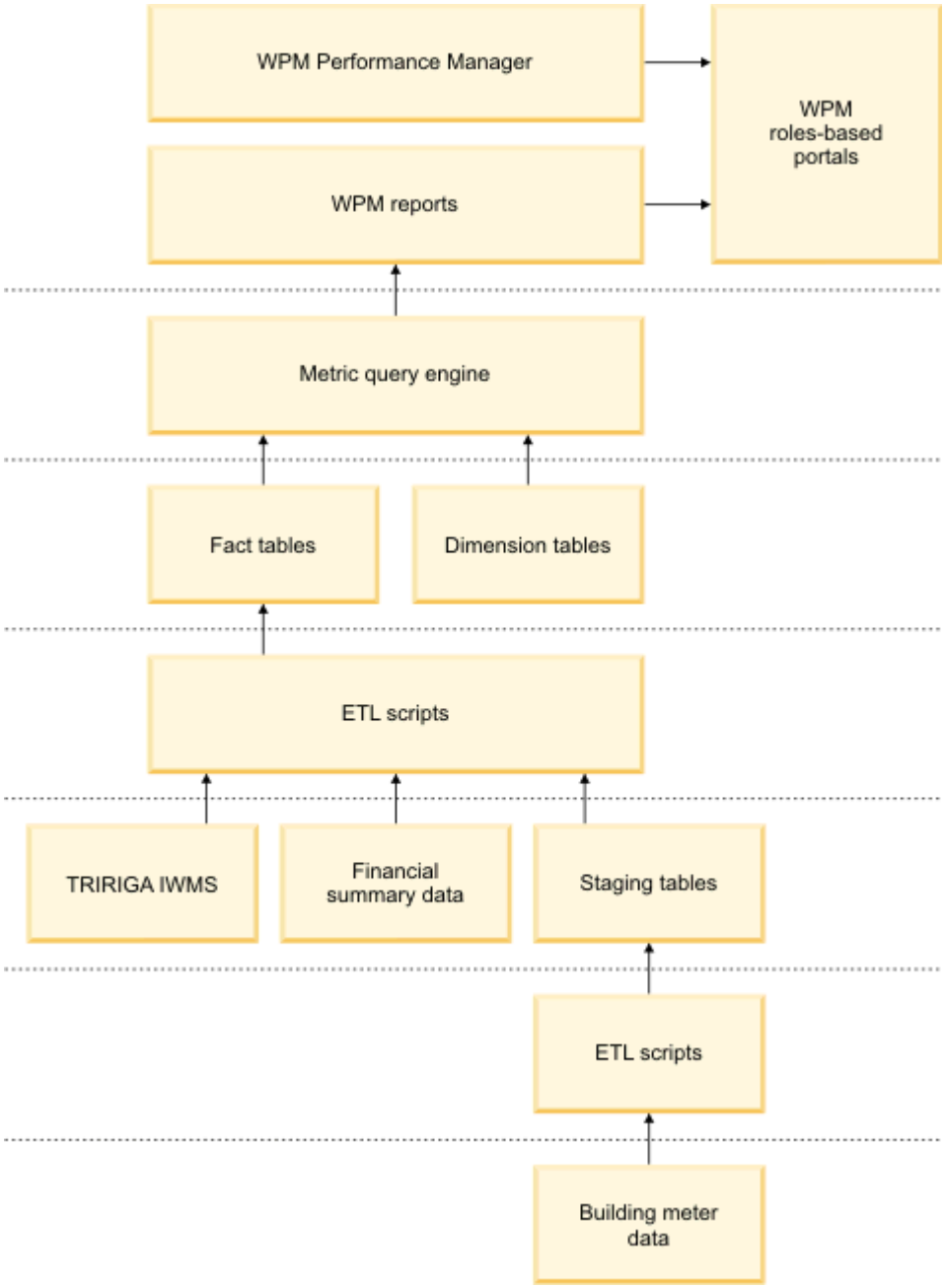
Architecture overview

The source data for TRIRIGA Workplace Performance Management comes from the TRIRIGA application database, financial summary data that is imported from an external financial system, and building meter data that is imported from external building management systems.

Using ETL technology, the source data is loaded into fact tables. The fact tables and dimension tables are in the same database repository as the TRIRIGA applications. The fact tables store the numerical data, referred to as facts, that is used to calculate the TRIRIGA Workplace Performance Management metric values. Each row in a fact table references one or more related business objects, classifications, or lists that group and filter the facts. These rows are called dimensions.

The metric query engine runs queries on the fact and dimension tables. Metric queries quickly recalculate metric values as the user moves up and down a hierarchical dimension.

The following diagram shows the distinct layers that make up this architecture and the flow of data between these layers:



Fact tables

Fact tables store the data that is used to calculate the metrics in metric reports. Fact tables are populated only through ETL transforms. To identify a business object as a fact table, from the Data Modeler set the Externally Managed flag in the fact table business object definition.

Each fact table is implemented in the IBM TRIRIGA Application Platform as a special business object that has some or all of the following elements:

Table 1. Fact tables

Fact table element	Description
Hierarchical dimensions	Each hierarchical dimension is a locator field to a business object that belongs to a hierarchical module (for example, a Building, Service Cost Code, or City). For each hierarchical dimension, a corresponding hierarchy structure supports metric reports. A hierarchical dimension can reference any or all business objects within a module. Be as specific as possible. Targeting a specific business object improves the granularity of your reporting. Each hierarchical dimension must have a corresponding hierarchy structure defined. Hierarchy structures are used for drill paths in metric reports.
Non-hierarchical dimensions	Each non-hierarchical dimension is either a list field or a locator field to a business object that belongs to a non-hierarchical module (for example, a Task or Person).
Numeric fact fields	Numeric fact fields are standard numeric fields, including or excluding Unit of Measure (UOM) properties. Numeric fact fields can be characterized as one of the following types: • Additive – Can be summed across all dimensions. • Semi-additive – Can be summed only across some dimensions. For example, the total number of people for a building captured monthly cannot be summed quarterly, since doing so would not yield a total for the quarter, whereas it can be summed by geography. Therefore, this fact is non-additive over time. • Non-additive – Cannot be summed across any dimension. For example, a ratio is a non-additive fact, since you cannot sum a ratio. Also, fields that contain values from different grains are non-additive.
UOM fields	Unit of measure (UOM) fields (except for Area fields) are captured in their local, entered, UOM.
Area fields	Area fields are captured in both Imperial (for example, square feet) and metric (for example, square meters) values.
Currency fields	Currency fields are captured by using the base currency. No currency conversion occurs.

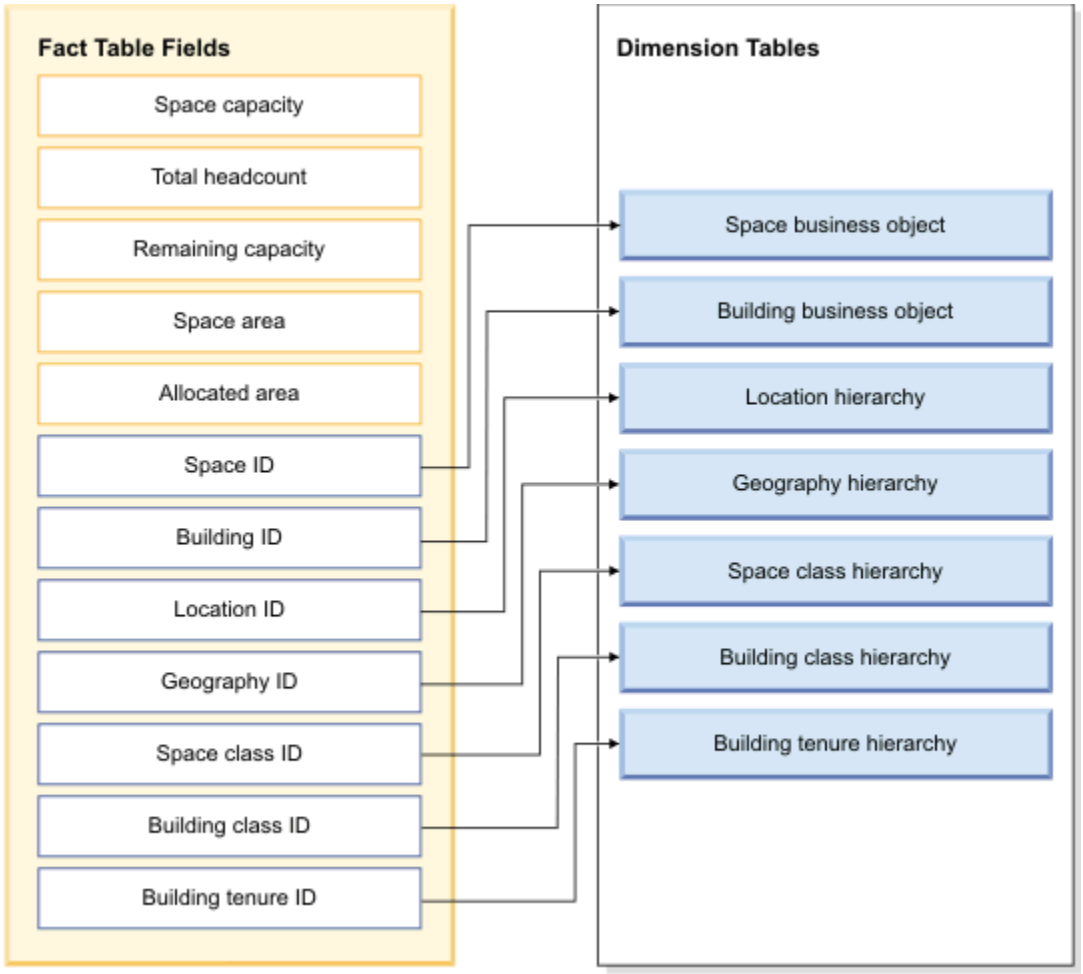
Table 1. Fact tables (continued)	
Fact table element	**Description**
Current time period	The time period dimension is a special dimension that is used to identify the date/time period for which a single fact record is applicable. This is most likely the time period when the data was captured. For cases when the time period dimension is not used as a drill path or filter, the triCapturePeriodTX field must be populated to indicate the dimension that is used to indicate the capture period. If this field exists, the corresponding business object for that dimension should contain a field that is named triCurrentBL, which is used to flag those dimension records that reflect the current period. These records are then used to filter the result set for the metric report.
Fiscal period	The fiscal period classification is used by the ETL process to define the capture period for fact records. This is the primary time period dimension in metric reports. Because it is possible to have different fact tables that contain data that is based on different capture frequencies with a single record for each level within the hierarchy, each level can be flagged as the current period. For example, if a year/quarter/month hierarchy is created in the fiscal period classification, it is possible to identify the current year, current quarter, and current month. A special ETL job type provides workflow to keep this data synchronized. As a general rule, all data in a single fact table should be captured at the same time period grain/level (year, quarter, month). If the time period grain/level is changed after data has been captured for a particular fact table, all data in that fact table must either be revised to the correct grain/level or truncated/removed.
Fact table business object	To identify a business object as one that will have fact tables supporting it, select the Externally Managed radio button in the business object properties when creating the business object.

Tip: Do not delete or change any of the fact business objects, fact tables, or ETL scripts that are delivered with the standard TRIRIGA software. Instead, to change an existing one, copy it, rename the copy, and tailor the copy to your needs.

Example fact table and associated dimensions

The fact and dimension tables are built by using the star schema method of data warehouse design. They are stored in the same database repository as the TRIRIGA applications.

The following diagram shows an example of one of the preconfigured fact tables in TRIRIGA Workplace Performance Management:



The diagram shows the space fact with five facts, including space capacity, total headcount, remaining capacity, space area, and allocated area. The space fact also references seven dimensions, including space, building, location, geography, space class, building class, and building tenure. The dimensions in the fact table link the facts to the corresponding dimension tables. Some dimensions are hierarchical, such as location and geography and others are not, such as space and building.

Flat hierarchy tables are used to identify the children of a selected business object. Flat hierarchy tables enable the metric query engine to browse hierarchical modules, business objects, and classifications.

The following table shows an example of a flat hierarchy that is based on geography:

Table 2. Geography Flat Hierarchy Example

SPEC_ID	**Level Number**	**Level 1 SPEC_ID**	**Level 2 SPEC_ID**	**Level 3 SPEC_ID**	**Level 4 SPEC_ID**
World	1	World	N/A	N/A	N/A
North America	2	World	North America	N/A	N/A
EMEA	2	World	EMEA	N/A	N/A
APAC	2	World	APAC	N/A	N/A
United States	3	World	North America	United States	N/A
Canada	3	World	North America	Canada	N/A
Nevada	4	World	North America	United States	Nevada
Texas	4	World	North America	United States	Texas

Using the example, if you wanted to identify all of the geographies that are children of North America, you first look up the North America SPEC_ID in the first column of the geography flat example table. Then, you might use the level number for North America, which is 2, to determine the filter column. By using the SPEC_ID and level number, you can identify all the geographies that are children, grandchildren, or any level below North America.

Metrics structure

The TRIRIGA Workplace Performance Management functionality focuses on capturing metric facts and enabling metric reporting.

Most TRIRIGA metrics are multidimensional, where the same metric provides a high-level summary view (for example, the Total Operating Cost/Area for the entire organization and portfolio) and, by drill down through various dimensions or filters, a role-specific view (for example, Total Operating Cost/Area for North American Operations for the facilities that are managed by North American operations).

Metrics measure process performance that is capable of identifying actionable results. Typically, the measures are ratios, percentages, or scores. Metrics have targets, thresholds, action conditions, accountability, and action task functionality.

TRIRIGA Workplace Performance Management includes the following types of metrics, which are the Scorecard categories in the Key Metrics portal section:

Customer Metrics
Measure customer satisfaction

Financial Metrics
Measure financial performance

Portfolio Metrics
Measure operational utilization and asset life cycle health

Process Metrics
Measure process efficiency and effectiveness

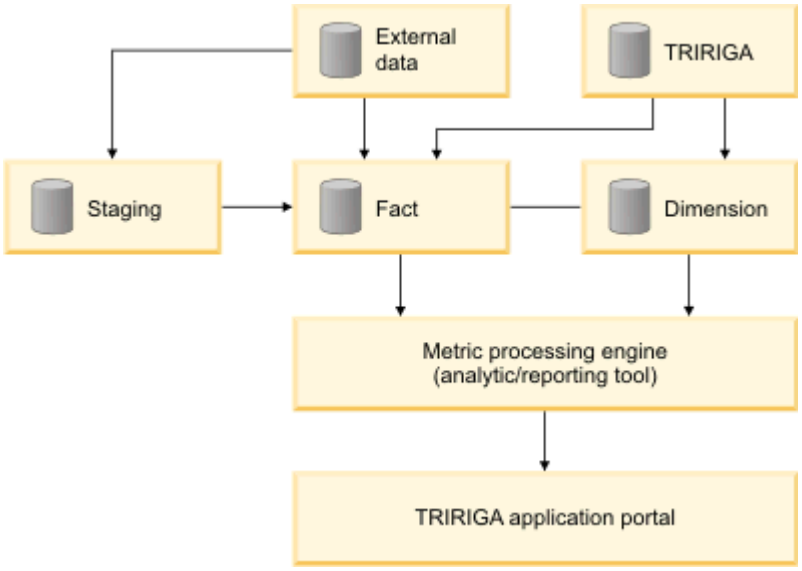
Reporting and Analysis Metrics
Analyze a specific performance metric

Additionally, TRIRIGA Real Estate Environmental Sustainability includes the following types of metrics:

Environmental
Measure performance of environmental initiatives

Building meters
Measure characteristics of a building as reported by meters and sensors

The following high-level process diagram depicts how metrics are defined, captured, and presented to the user:



The TRIRIGA database is the primary source of data for gathering operational data to be loaded into fact tables. Optionally, you can extract data from other sources to be loaded into fact tables.

Each fact table contains the lowest level of aggregated data (such as Building level) for each metric category. For efficiency reasons, a fact table is a de-normalized (flattened) table containing data elements from multiple TRIRIGA tables.

The dimension table contains dimensions for each metric. Dimensions are stored in a separate table for efficiency. The fact table contains a key (Spec ID) for each dimension. The dimension table can be either a flat hierarchy table or an TRIRIGA business object table.

The Metric Processing Engine (Analytic/Reporting Tool) generates metrics using data stored in fact tables along with metric setup data and dimension data.

Metric data, along with notifications, actions, and alerts, are presented to users in a role-based portal in various forms (including reports, queries, and graphs) as defined in the metric setup table. A user can drill down into a specific object or drill path to further analyze the metric data presented to them.

Metric reporting is dependent on metric fact tables. These fact tables are implemented using the Data Modeler but are identified with a unique object type that signifies that it is a metric object. Metric objects are populated using an ETL development environment, which is different from all other object types that are updated through the metadata layer. The scheduling of the ETL process is controlled from within the TRIRIGA system using the Job Scheduler.

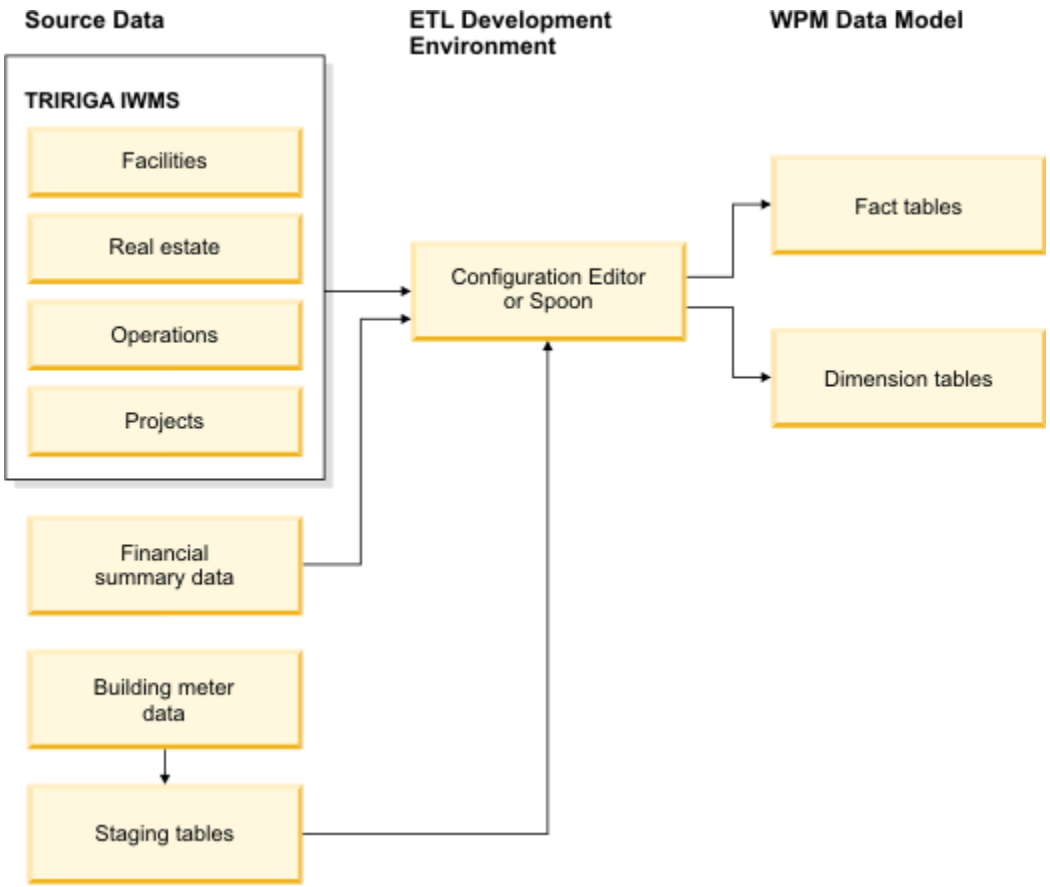
ETL integration

TRIRIGA uses the Pentaho ETL development environment, Spoon, to generate transform XML files. These transforms, when run through the API, move data from source to destination tables.

ETL integration architecture

TRIRIGA uses an ETL environment to create the ETL scripts that populate the fact tables. The ETL development environment is the Pentaho Data Integration tool, Spoon. The ETL development environment enables the creation of SQL queries that read data from the TRIRIGA business object tables and map and transform the results to the fact table fact and dimension columns.

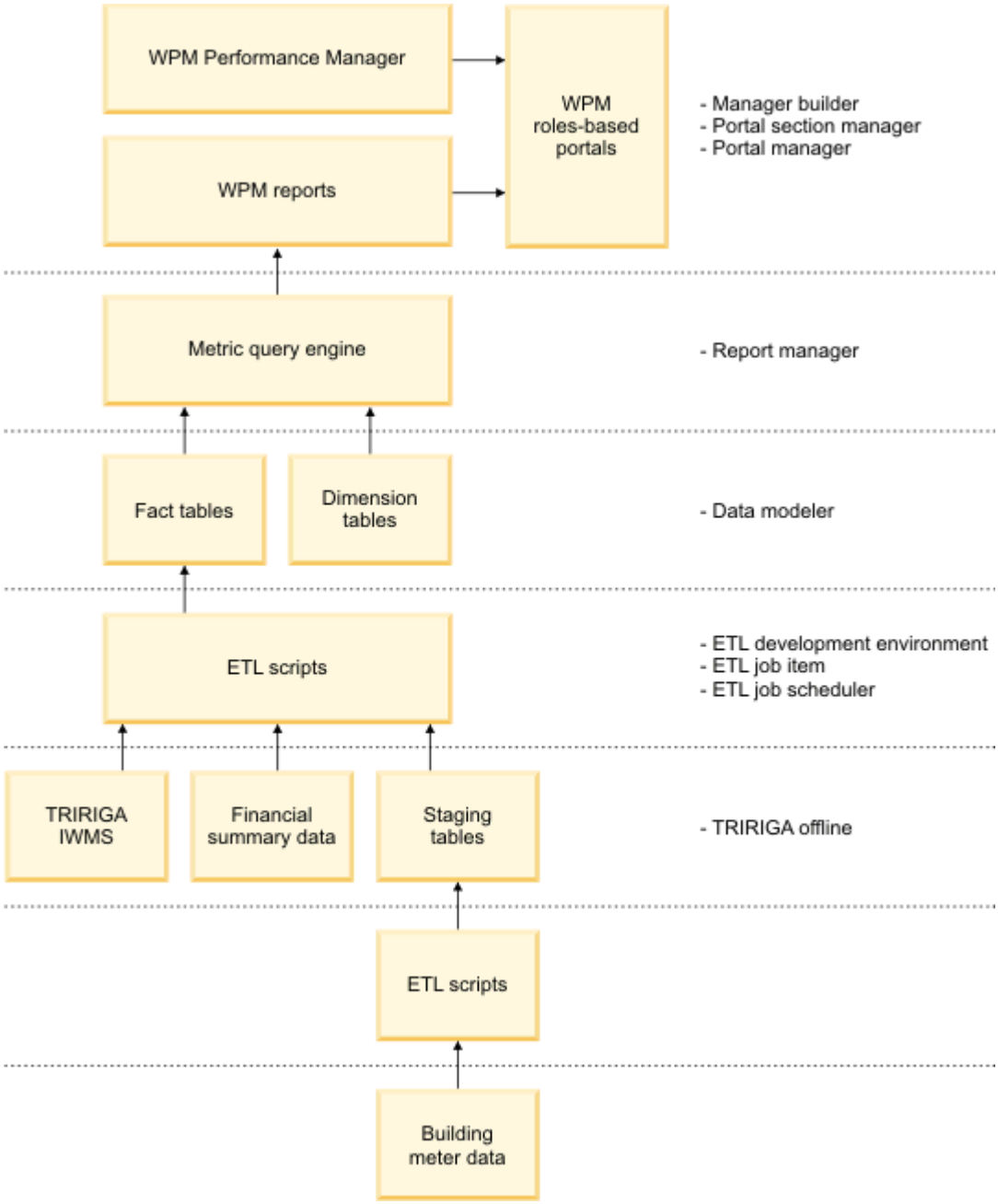
The following diagram shows the flow of data between the source data, ETL development environment, and the TRIRIGA Workplace Performance Management data model layers:



ETL job items are the business objects that reference the ETL scripts that are used to populate the fact tables.

TRIRIGA Workplace Performance Management uses standard TRIRIGA Application Platform tools.

The following diagram shows the application platform tools:



ETL integration process

To move data from the source to destination tables, you run the ETL transform files that you developed in Pentaho Spoon through the API.

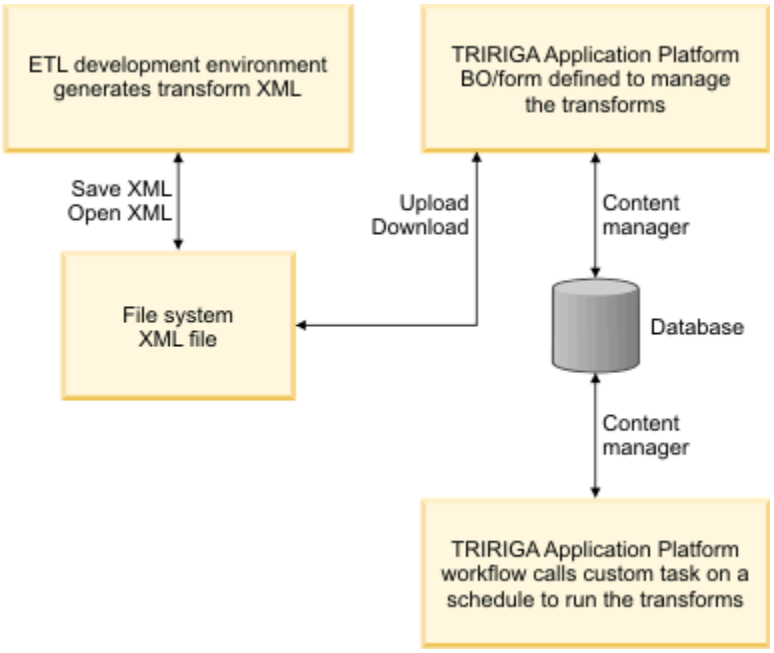
There must be a transform for each fact table. Fact tables are populated only through ETL transforms and not through the TRIRIGA application.

The TRIRIGA Application Platform includes a workflow that runs on a schedule to load the fact tables. The workflow calls a platform Custom workflow task, which retrieves the latest transform XML from the Content Manager and uses the Kettle API to run the transform.

The scheduled ETL process is available with the following licenses:

- Any IBM TRIRIGA Workplace Performance Management license
- An IBM TRIRIGA Real Estate Environmental Sustainability Manager license
- An IBM TRIRIGA Real Estate Environmental Sustainability Impact Manager license

- An IBM TRIRIGA Workplace Reservation Manager license
- An IBM TRIRIGA Workplace Reservation Manager for Small Installations license

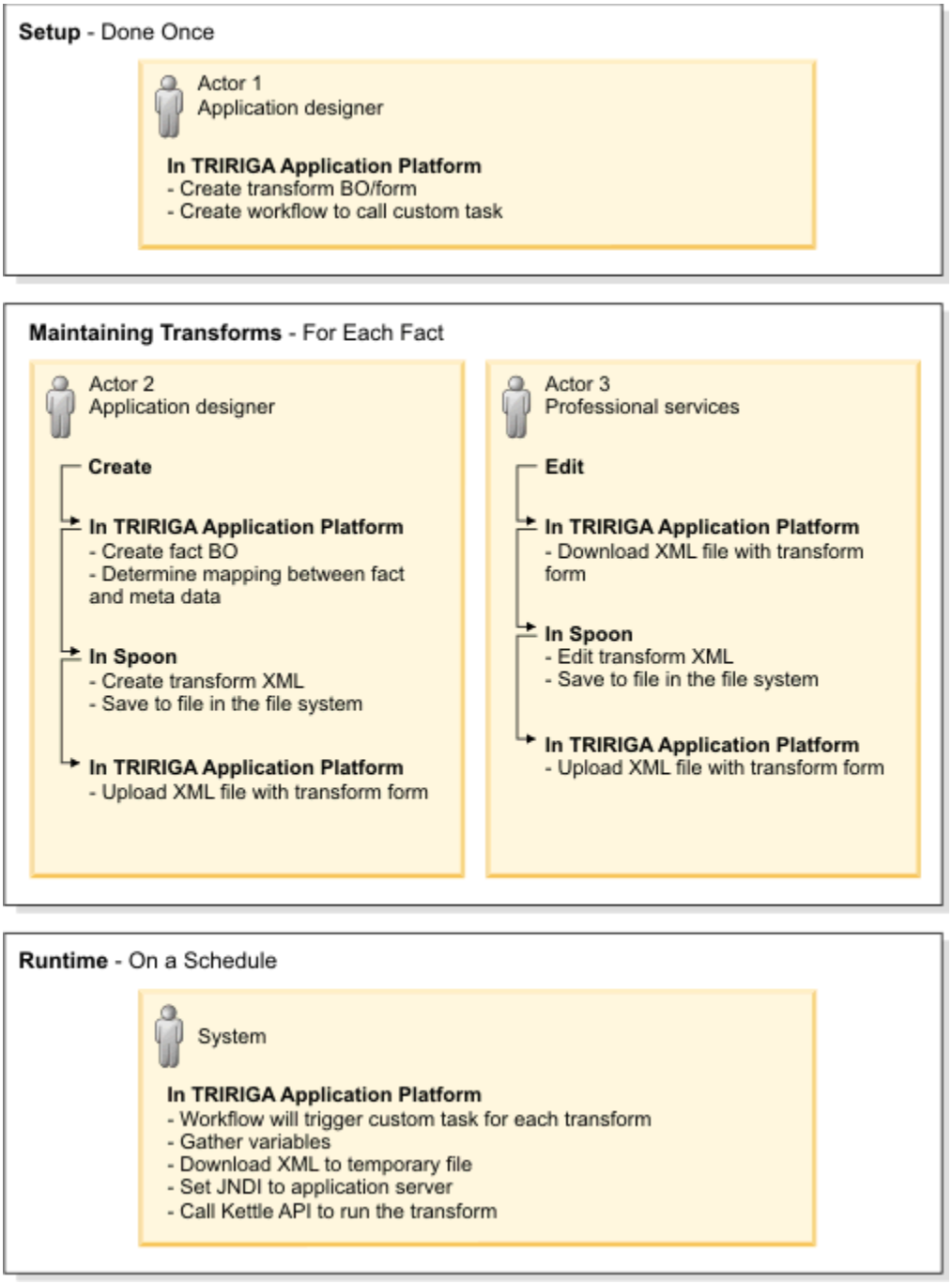


There are three main processes:

- Setup, which involves creating the Transform business object/form/navigation item and the workflow itself.
- Creating/Maintaining the Transform XML using an ETL development environment.
- Runtime, which is a scheduled workflow that executes a Custom workflow task to periodically run through the transforms to update the fact tables.

Restriction: Additional licenses may be required for the set up and maintenance processes, such as a license for the triJobItem Module. The licenses needed to create or edit the ETLs differ from the licenses needed for the runtime process.

The following diagram summarizes these processes for Pentaho Spoon ETL transforms:



Data is pulled by ETL scripts from business objects, including the Financial Summary business object, into which financial summary records are imported from spreadsheets or by customer-designed interfaces with a financial system.

Prerequisite setup for ETL integration

Within TRIRIGA Application Platform, a business object and a form manage the transforms. The source tables and destination fact table must be defined and the mappings understood to create the transformation.

There is a record for each fact table loaded through a transform. A binary field on the Transform business object pulls the transform XML file into the Content Manager. The form provides a way to upload/download the XML file so the transform XML can be easily maintained. TRIRIGA comes preconfigured with an ETL Job Item as the implementation of this business object or form.

Within the TRIRIGA Application Platform, a workflow runs on a schedule and calls a Custom workflow task for each fact table that needs to be loaded or updated. The Job Scheduler provides a mechanism that automatically calls the custom workflow task for ETL Job Items.

TRIRIGA ships all business objects, forms, and workflows required to support the as-delivered TRIRIGA Workplace Performance Management and TRIRIGA Real Estate Environmental Sustainability products.

Defining and maintaining ETL transforms

Use an ETL development environment to create a transformation to move data. During the transform, you can do calculations and use variables from TRIRIGA Application Platform and from the system.

Using ETLs with Pentaho Spoon

You can use Pentaho Spoon as an ETL development environment.

Overview of using Pentaho Spoon

You must first create the source and destination tables and establish the corresponding mappings. Next, you must identify variables that need to be passed into the transform and add these variables to the transform business object or form. Then, you can use Pentaho Spoon and the following steps to define and maintain transforms.

Tip: It might not be necessary to perform all of the following steps. The steps that are required depend on whether you are defining or maintaining a transform.

- Run the `spoon.bat` or `kettle.exe` file by opening Spoon. Select **No Repository** as you do not need to use one.
- Either open an existing XML file, that was downloaded to the file system by using the **Transform Form**, or use **File** > **New** > **Transformation** to create a new transform.
- Define the JNDI settings for the local database. Use TRIRIGA as the connection name. Set the database connection in the tool by using **View** > **Database Connections** > **New**. When the transform is run by the workflow, the connection is overwritten with the application server's connection information.
- Use the **Design** menu to lay out the transform as follows:
  - Extract rows from the tables by using **Design** > **Input** > **Table Input**.
  - Make sure that all row fields have values when used in a calculation with **Design** > **Transform** > **Value Mapper**.
  - Use **Design** > **Transform** > **Calculator** for calculations.
  - Provide the sequencing for the destination rows with **Design** > **Lookup** > **Call DB Procedure** by using the NEXTVAL stored database procedure.
  - Use **Scripting** > **Modified JavaScript Value** and other steps to transform data as necessary.
  - Identify the table to which the rows are output with **Design** > **Output** > **Table Output**
  - Map the fields by Generated Mappings against Target Step.
- Link steps by using **View** > **Hops** and lay out the transform, step-by-step.
- Test thoroughly by using **execute**, and other available utilities. Testing makes sure that the process is accurate and that the expected rows are returned and transformed appropriately.
- Save the transform by using **File** > **Save**. Do not save to the repository. Instead, set the file type to XML and save with the `.ktr` file extension. If you do not set the file type, the default is Kettle transform, which saves an XML file with the `.ktr` file extension.

Installing Pentaho Spoon

You can install Pentaho Spoon as an ETL development environment. Use version 5.4.0.1-130, which is the version with which TRIRIGA integrates.

Procedure

1. Locate Pentaho Spoon version 5.4.0.1-130 at https://sourceforge.net/projects/pentaho/files/Data%20Integration/5.4/pdi-ce-5.4.0.1-130.zip/download.
2. Extract the files from the `.zip` file and keep the directory structures intact.
3. Review the most recent version of Pentaho Spoon and accompanying detailed documentation at http://kettle.pentaho.org/.

Setting up a local JNDI

You must define the local JNDI settings for your database by updating the properties file.

Procedure

1. From the `pdi-ce-5.4.0.1-130/data-integration/simple-jndi/` directory, edit the `jdbc.properties` file and add the following properties:
   - `LocalJNDI/type=javax.sql.DataSource`
   - `LocalJNDI/driver=oracle.jdbc.driver.OracleDriver`
   - `LocalJNDI/url=jdbc:oracle:thin:@localhost:1521:orcl`
   - `LocalJNDI/user=tridata2`
   - `LocalJNDI/password=tridata2`
2. Update the information as appropriate, including the driver if you are using DB2 or SQL Server.
3. Save and close the file.

Creating transforms and database connections

You can create transforms and database connections for use between Pentaho Spoon and TRIRIGA

Procedure

1. Open the Spoon tool by running the Spoon `.bat` file in the `pdi-ce-5.4.0.1-130/data-integration/` directory. Choose to run without a repository.
2. To create a new transform, right-click **Transformations** and select **New**.
3. In **View** mode, create your database connection. Right-click database connections within **Transformations** and select **New**.
4. The Custom workflow task replaces the TRIRIGA connection with the application server JNDI settings. Configure the database connection as follows:
   - **Connection Name:TRIRIGA**
   - **Connection Type:Oracle**
   - **Access:JNDI**
   - **Settings:JNDI Name:LocalJNDI**
5. Select **Test** to make sure that the connection is set up correctly.
6. Save the database connection details.
7. Be sure to save the transform as an XML file not in the repository. The extension for the Kettle transformation is `.ktr`. The default for Kettle transformation saves the file as `.ktr`.

Running a transform from Pentaho Spoon

You can run a transform that is either completed or is in the process of being completed.

Procedure

1. Save the transform and select **Run**.
2. Set **variables**, if necessary.
3. Select **Preview** to display the changes to the input stream as each step is run.

Selecting Spoon steps

You can use the Design Mode to select the various Spoon step types and add them to a transform.

Procedure

1. To add a step to a transform, select **Step type** and drag the step in the left navigation onto your palette.
2. To link two steps, select **View** in the left navigation and double-click **Hops**.
3. Put in the **From** and **To** steps and select **OK**.
4. Alternatively, you might select **Ctrl+click** on two steps, right-click one of the steps, and select **New hop**.
5. To add a note to the transform, right-click the palette and select **New Note**.

Spoon example transform

You can download a copy of any of the existing **.ktr** scripts that are contained in an existing ETL job item to follow along in the step descriptions. The following shows an example of a Spoon transform.

Most of the as-delivered ETLs have the same flow as the example but the specifics are different, for example, the database tables from which data is extracted and how the data is transformed.

The example transform includes the following items:

- Pulls input rows and fields from T_TRIORGANIZATIONALLOCATION org, and T_TRISPACE space where org.TRILOCATIONLOOKUPTXOBJID = space.SPEC_ID.
- Uses IBS_SPEC.UPDATED_DATE to limit the rows that are selected, by using the date range that is passed in from the transform business object.
- Use Value Mapper to make sure that there is a value in all rows for space.TRIHEADCOUNTNU, space.TRIHEADCOUNTOTHERNU, and org.TRIALLOCPERCENTNU, if not set it to 0.
- Uses the Calculator to set TRIFACTTOTALWORKERSASS to (space.TRIHEADCOUNTNU + space.TRIHEADCOUNTOTHERNU) * org.TRIALLOCPERCENTNU.
- Gets TRICREATEDBYTX and TRIRUNDA, passed in from the Transform BO through Get Variables step.
- Uses Add Constant to set the sequence name and increment so that it is available in the input stream for sequencing step.
- Uses the DB Procedure NEXTVAL to set the SPEC_ID, set this step to use five threads for enhanced performance.
- Uses a JavaScript scripting step to determine whether the project was on time or not, and to calculate the duration of the project. Set this step to use three threads for better performance.
- Maps the fields to T_TRISPACEALLOCFACTOID.

Key things to consider as you build a transform include the following items:

- Test as you add each step to make sure that your transform is doing what you want.
- Transforms need to be developed in a defensive manner. For example, if you are making calculations that are based on specific fields, all rows must have a value in these fields, no empties. If not, the transform crashes. Use Value Mapper to make sure that all fields used in a calculation have a value.

- Dates are difficult to handle as the databases TRIRIGA supports keep DATE and TIME in the date field. Date solutions show how to handle date ranges in SQL.
- Make sure to use JNDI settings and that your transform database is independent, especially if your solution needs to run multiple database platforms (DB2, Oracle, and Microsoft SQL Server).
- Any attributes on the Transform business object are sent to the Transform as a variable. There are a couple exceptions. Attributes of type Time or System Variable are ignored. You can use the variables in your SQL or pull them into the input stream by using Get Variables with the following syntax: ${VariableName}, where VariableName is the attribute name.
- Make sure to completely test and set up the transform before you use variables in the Table Input. It is challenging to test JavaScript, Table Input Preview, and Table Mapping. You can set variables in the transform with **Edit** > **Set Environment Variables** or in the **Execute** page **Variable** section. By using variables more of the test functions within Spoon are made available.
- Test your connection before you use JNDI, before you run a search, or before you run a Spoon transform. The JNDI connection must be tested to avoid Spoon having any potential performance issues.
- Consider adding an index. It can be key to performance as the ETLs pull data from the T tables in a manner that is different from the regular application.

The preceding items detail the transform as you configure the Spoon steps used. The items concentrate on the main steps that are used by the transforms that are delivered with TRIRIGA. Spoon provides other step types that you can use to manipulate your data; use the steps as necessary, depending on your transform needs.

Configuring Spoon input steps
You can use input steps to bring data into the transform.

About this task

Table input is the source of most of your data. By using the specified database connection, you can set up SQL to extract data from tables.

Procedure

1. Double-click a table input step to open up the information for the step.
2. Set the connection to TRIRIGA or the source database.
3. Enter your SQL into the SQL table.
4. Select **OK** to save the table input.
5. Select **Preview** to preview the data that the table input includes.

   If you are using variables in SQL, the variables must be set for the **Preview** to function. You must either hardcode the variable values while testing or select **Edit** > **Set Environment Variables** to set the variable values. The variables in SQL are *$(triActiveStartDA_MinDATE} and ${triActiveEndDA_MaxDATE}*.

Results

The SQL provided extracts input rows from T_TRIORGANIZATIONALLOCATION organization and T_TRISPACE space, where org.TRILOCATIONLOOKUPTXOBJID = space.SPEC_ID. It uses dates from the transform business object to limit the data that is included.

Configuring Spoon transform steps
You can use transform steps to change input data or add information to the input stream.

About this task

In the Spoon example transform, the **Calculator**, **Add Constants**, and **Value Mapper** steps are used. You can add a sequence through Spoon but it is not database independent and does not work on SQL Server. Instead, you can use the provided DB Procedure.

Procedure

1. Use the **Value Mapper** step to ensure that fields have values or to set fields different values. You can set values to a target field based on the values of a source field. If the target field is not specified, the source field is set instead of the target field. You must ensure that all the fields in a calculation have a value. If a null value is encountered during a calculation the transform fails.
2. Double-clicking the **Value Mapper** opens the dialog to input the necessary information. In the Spoon example transform, it is used to set a field to 0 if it does not have a value.
3. Use the **Add Constants** step to add constants to the input stream and to set the values that are needed for the *NEXTVAL* DB Procedure.
4. You must have this step in all transforms that use the *NEXTVAL* DB Procedure. Set *SEQ_NAME* to SEQ_FACTSOID and *INCR* to 1.
5. Use the **Calculator** step to take fields and run a limited set of calculations. It provides a set of functions that are used on the field values. The **Calculator** step performs better than using JavaScript scripting steps.
6. The built-in calculations are limited. Select the **Calculation** column to show the list of available functions.

Configuring Spoon lookup steps
You can use lookup steps to extract extra data from the database into the data stream.

About this task

The Call DB procedure allows the transform to call a database procedure. Information flows through the procedure and back to the transform. You can create sequences for the fact table entries.

Procedure

1. Set up your DB procedure call to use *NEXTVAL*, to send in *SEQ_NAME* and *INCR* and to output by using *CURR_VALUE*.
2. Determine how many instances of this lookup step to run. When you are testing the transform, running this step with five instances greatly helps with performance.
   For example, for 30,000 records the performance time reduces from 90 seconds down to 30 seconds.
3. Change the number of threads that run a step by right-clicking the step and selecting **Change number of copies to start**.
4. Tune the number of threads that are running the DB Procedure step.

Configuring Spoon job steps
Even though you are not creating jobs that you need to get Kettle variables and fields into the input stream, you need to ensure that you can set an output field to a variable.

About this task

In the Spoon example transform, the *triCreatedByTX* and *triRunDA* variables are brought into the input stream. You also get variables to pull in the *ONTIME* and *DURATION* variables so that you can set them during the JavaScript scripting steps.

Procedure

It is important in case there is a failure during a transform to time stamp when the transform is run. The example does it by using the *triRunDA* variable and this provides an avenue for rollback, even though the process does not have explicit steps for it.

When you are setting fields to values in the transform, they must be the same type otherwise the transform fails.

Configuring Spoon scripting steps
You can use scripting steps to implement JavaScript features.

About this task

You can use it for specific data manipulations on the input stream that cannot be done with the Calculator. You can calculate the duration or set values into the stream, which is based on other values with an `if/then/else` clause. You can set values into the transform stream that are constants or are from a variable.

Procedure

1. Use JavaScript scripting if you need logic to set the values.
2. In the Spoon example transform, duration is calculated by subtracting two dates from each other. The duration then determines whether a plan was on time.
3. With the JavaScript scripting features, if you want information out of the Table Input rows, you must iterate to find the field you want. You cannot access the field directly, unless you alias the field in the Table Input step.

Example

The JavaScript scripting example details how to obtain and set the variables.

```
var actualEnd;
var actualStart;
var plannedEnd;
var plannedStart;
var duration;
var valueDuration;
var valueOnTime;

// loop through the input stream row and get the fields
// we want to play with
for (var i=0;i<row.size();i++) {
  var value=row.getValue(i);

  // get the value of the field as a number
  if (value.getName().equals("TRIACTUALENDDA")) {
    actualEnd = value.getNumber();
  }
  if (value.getName().equals("TRIACTUALSTARTDA")) {
    actualStart = value.getNumber();
  }
  if (value.getName().equals("TRIPLANNEDENDDA")) {
    plannedEnd = value.getNumber();
  }
  if (value.getName().equals("TRIPLANNEDSTARTDA")) {
    plannedStart = value.getNumber();
  }

  // these are the 'variables' in the stream that we want
  // to update with the duration and ontime setting
  // so we want the actual Value class not the value
  // of the variable
  if (value.getName().equals("DURATION")) {
    valueDuration = value;
  }
  if (value.getName().equals("ONTIME")) {
    valueOnTime = value;
  }
}

// calculate the duration in days
duration = Math.round((actualEnd - actualStart) / (60*60*24*1000));
// calculate the duration in hours
// duration = (actualEnd - actualStart) / (60*60*1000);

// set the duration into the 'variable' in the row
valueDuration.setValue(duration);

// determine ontime and set the value into the
// 'variable' in the row stream
if ((actualEnd == null) || (plannedEnd == null))
  valueOnTime.setValue("");
```

```
else if (actualEnd > plannedEnd)
  valueOnTime.setValue("no");
else
  valueOnTime.setValue("yes");
```

Select **Test Script** to make sure that the JavaScript compiles. The **Test Script** and **Preview** steps in Table Input cannot handle variables unless they are set. You can set variables in the transform by using **Edit** > **Set Environment Variables**. This makes more of the test function within Pentaho Spoon.

For example, you can use **Edit** > **Set Environment Variables** and set *triActiveStartDA_MinDATE* to *to_date('20061201', 'YYYYmmdd')*.

If you are using column aliases when you are defining your query, you must use the same alias when you are looking up the column with *getName*.

The following example, in the table input step, shows the select option:

```
SELECT mainProject.triProjectCalcEndDA ActualEndDate,
       mainProject.triProjectActualStartDA ActualStartDate
```

If you are looking up the value for *ActualEndDate*, use the alias and not the column name from the database, as illustrated:

```
if (value.getName().equals("ActualEndDate")) {
   actualEnd = value.getNumber();
}
```

Configuring Spoon output steps
You can use output steps to write data back to the database.

About this task

Table output and table output mapping stores information to a database. This information is then used in the fact tables. When you have all the information to save a transform, you can add output steps to the end of your transform and connect them to the last step.

Procedure

1. Double-click and add the connection information and the fact table you want to use as the output table.
2. When it is set up and the steps are connected, right-click the table output step.
3. Select **Generate mapping against this target step**.
4. Map the source fields to the target fields in the target database and select **OK**.

   Source fields include the additional fields added to the input stream. Ensure to set the table mapping before you use variables in the table input step.
5. To complete your transform, drag the mapping step in between the last two steps.
6. If necessary, you can modify and add more fields to the mapping step.

Testing the transform

You can test the transform after you add each Spoon step, or at the end when all Spoon steps are complete. Testing after each step makes debugging easier. Before you can test the transform, you must first save the transform.

About this task

The variables section details the variables that are used in the transform. When you test the transform by using Spoon, you can set values to these variables. When the transform is run from within TRIRIGA, the variables form part of the transform business object. Set and save the values that are used into the smart object before the custom workflow task is called.

Procedure

1. Set the *triRunDA* variable to the date and time of the workflow run. It does not need to be an attribute on the transform business object. It is the Number representation of the run date and time. triRunDA does not have six formats of the date since it is generated dynamically by the custom workflow task. triRunDA is needed for setting the create date of the fact row.
2. triCreatedByTX is an attribute on the transform business object.
3. triActiveStartDA_MinDATE and triActiveEndDA_MaxDATE are the wrapped representations of triActiveStartDA and triActiveEndDA. During Spoon testing, if you are testing on Oracle or DB2, you must wrap them with to_date ('the date you want', 'the format').
4. Click **Launch** to run the transform. If a step has an error, the step appears in red and the error is saved to the log file. You can access the log file through the log page.

Date solution

Several date variables require calculation and comparison when used by Pentaho Spoon. Date solution provides these calculations and comparisons.

There are three instances when date solution is required:

1. Compare two dates. This comparison is used to determine whether a project is on time.
2. Calculate duration between two dates in days. In some cases, this calculation is used to calculate the duration in hours.
3. Compare a date, such as modified date or processed date, to a range of dates, such as first day of month and last day of month.

The first and second instances are solved by using JavaScript scripting steps.

The third instance is solved by using a date range in the table input.

There are two types of dates. Dates that are stored as a Date in the database and dates that are stored as a Number in the database.

Tip: All TRIRIGA objects store **Date** and **Date and Time** fields as numbers in the database. Select a field as a number to interact with business object tables. Select a field as a date to interact with system platform table fields defined as date.

Select field as date

You can interact with system platform table fields defined as date by selecting a field as a date.

The following example code uses `IBS_SPEC.UPDATED_DATE` as the date field to determine whether a row is needed. `triActiveStartDA` and `triActiveEndDA` are the date range. These dates come from the triActiveStartDA and triActiveEndDA fields on the transform business object.

The IBS_SPEC table is not a TRIRIGA object. It is a system platform table that is used to track objects in TRIRIGA. It includes a field that changes every time an object in TRIRIGA is updated. The field is the UPDATED_DATE field and in the database it is a date field, not a number field.

In the following example code, `${triActiveStartDA_MinDATE}` and `${triActiveEndDA_MaxDATE}` are used. These wrapped dates fields fetch all records from 12:00 am on the start date to 11:59 pm on the end date.

```
SELECT org.SPEC_ID ORG_SPEC_ID, org.TRIORGANIZATIONLOOKUOBJID,
space.CLASSIFIEDBYSPACESYSKEY, org.TRIALLOCPERCENTNU, org.TRIALLOCAREANU,
space.TRIHEADCOUNTNU, space.TRIHEADCOUNTOTHERNU, spec.UPDATED_DATE
FROM T_TRIORGANIZATIONALLOCATION org, T_TRISPACE space, IBS_SPEC spec
WHERE org.TRILOCATIONLOOKUPTXOBJID = space.SPEC_ID
and space.SPEC_ID = spec.SPEC_ID
and spec.UPDATED_DATE >= ${triActiveStartDA_MinDATE}
and spec.UPDATED_DATE <= ${triActiveEndDA_MaxDATE} order by UPDATED_DATE
```

In Oracle or DB2, ${triActiveStartDA_MinDATE} displays like to_date ('20070701 00:00:00', 'YYYYmmdd hh24:mi:ss') and ${triActiveEndDA_MaxDATE} displays like to_date ('20070731 23:59:59', 'YYYYmmdd hh24:mi:ss').

In SQL Server, these dates look slightly different because of database specifics, but are set up to capture all the rows between the two dates.

Select field as number
You can interact with business object tables by selecting a field as a number.

Instead of using IBS_SPEC.UPDATED_DATE as the date determination field for the TRIRIGA date, this method compares the determination field directly to triActiveStartDA and triActiveEndDA, since they are all numbers in the database.

In the following example code, triCaptureDA is a field on T_TRISPACE.

```
SELECT org.SPEC_ID ORG_SPEC_ID, org.TRIORGANIZATIONLOOKUOBJID,
space.CLASSIFIEDBYSPACESYSKEY, org.TRIALLOCPERCENTNU, org.TRIALLOCAREANU,
space.TRIHEADCOUNTNU, space.TRIHEADCOUNTOTHERNU, space.TRICAPTUREDA
FROM T_TRIORGANIZATIONALLOCATION org, T_TRISPACE space, IBS_SPEC spec
WHERE org.TRILOCATIONLOOKUPTXOBJID = space.SPEC_ID
and space.TRICAPTUREDA >= ${triActiveStartDA_Min}
and space.TRICAPTUREDA <= ${triActiveEndDA_Max} order by space.TRICAPTUREDA
```

Similar to the date fields, use the *Min* and *Max* variables to make sure that the start is 00:00:00 and the end is 23:59:59. For example, use these variables to make your search pick up a record on December 31st at 13:54 in the afternoon.

Date variables
For each Date or Date and Time attribute on the Fact Transform business object, the system creates six Kettle variables.

The following table summarizes these Kettle variables:

Table 3. Kettle variables

Kettle Variable	**Description**
${triActiveStartDA}	No suffix = is the value in milliseconds since January 1, 2014, with no changes to the time. This variable is for fields that are represented as a number.
${triActiveStartDA_Min}	Min = is the value in milliseconds since January 1, 2014, with the time value set to 00:00:00 for the specified date. This variable is for fields that are represented as a number.
${triActiveStartDA_Max}	Max = is the value in milliseconds since January 1, 2014, with the time value set to 23:59:59 for the specified date. This variable is for fields that are represented as a number.
${triActiveStartDA_DATE}	DATE = is the wrapped value in date format, with no changes to the time. This variable is for fields that are represented as date in the database. For Oracle or DB2 it is wrapped and displays like: to_date (‘20070615 22:45:10’,’YYYYmmdd h24:mi:ss’) For SQL Server it displays like: ‘20070615 22:45:10’
${triActiveStartDA_MinDATE}	MinDATE = is the wrapped value in date format, with the time value set to 00:00:00. This variable is for fields that are represented as date in the database.

Table 3. Kettle variables (continued)	
Kettle Variable	**Description**
${triActiveStartDA_MaxDATE}	MaxDATE = is the wrapped value in date format, with the time value set to 23:59:59. This variable is for fields that are represented as date in the database.

When you specify the ${triActiveStartDA_Min} and ${triActiveStartDA_Max} variables to see a time period between two dates, you need to capture all the rows within the time period. You need to start at midnight and stop at 1 second before midnight. If you use only the date value, you might not get all the rows that you want, depending on the time on the variable. You must specify the minutes and seconds because both TRIRIGA databases store dates in a date time or number field.

The *${triActiveStartDA_MinDATE}* and *${triActiveStartDA_MaxDATE}* variables help with date comparisons.

For example, for triActiveStartDA whose value is 20070615 22:45:10,

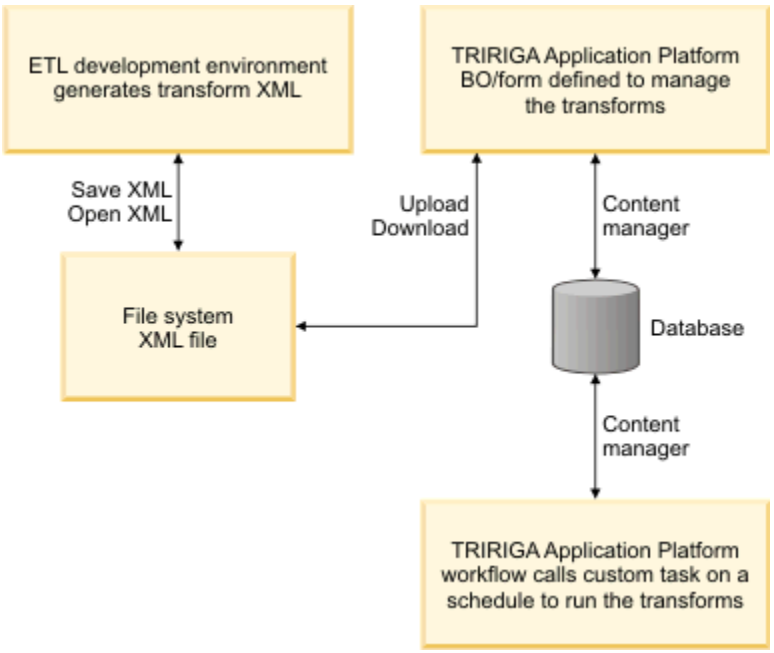
```
triActiveStartDA_MinDATE =
(Oracle) to_date('20070615 00:00:00','YYYYmmdd h24:mi:ss')
(SQL Server) '20070615 00:00:00'
triActiveStartDA_MaxDATE =
(Oracle) to_date('20070615 23:59:59','YYYYmmdd h24:mi:ss')
(SQL Server) '20070615 23:59:59'
```

Moving ETL Scripts into TRIRIGA from Kettle

Once the transform is completed and tested, it must be uploaded to the TRIRIGA ETL job item.

Remember: Save the transform with the file type as XML and extension `.ktr`.

The following graphic describes the flow between the ETL environment and TRIRIGA.



Variables passed to Kettle
All variables that are passed to Kettle are of type String. Number variables are converted by the custom workflow task to type String. The TRIRIGA field types that are supported in Kettle are Text, Boolean, Date, Date and Time, Locators, and Numbers.

Table 4. The following are example fields on the ETL Job Item:

Field name	**Field label**	**Field type**
triActiveEndDA	Active End Date	Date
triActiveStartDA	Active Start Date	Date
triBONamesTX	BO Names	Text
triControlNumberCN	Control Number	Control Number
triCreatedByTX	Created By	Text
triLocator	triLocator	Text
triModuleNamesTX	Module Names	Text
triNameTX	Name	Text
triTransformBI	Transform File	Binary

Table 5. The following are the variables passed to Kettle:

Variable that is passed to Kettle	**Description**
triNameTX	(Text)
triActiveStartDA	(Number) date in milliseconds since January 1, 2014
triActiveStartDA_DATE	(Date) wrapped if Oracle or DB2, time is whatever it was on the attribute
triActiveStartDA_MinDATE	(Date) wrapped if Oracle or DB2, time is 00:00:00
triActiveStartDA_MaxDATE	(Date) wrapped if Oracle or DB2, time is 23:59:59
triActiveStartDA_Min	(Number) date in milliseconds since January 1, 2014, time is 00:00:00
triActiveStartDA_Max	(Number) date in milliseconds since January 1, 2014, time is 23:59:59
triActiveEndDA	(Number) date in milliseconds since January 1, 2014
triActiveEndDA_DATE	(Date) wrapped if Oracle or DB2, time is whatever it was on the attribute
triActiveEndDA_MinDATE	(Date) wrapped if Oracle or DB2, time is 00:00:00
triActiveEndDA_MaxDATE	(Date) wrapped if Oracle or DB2, time is 23:59:59
triActiveEndDA_Min	(Number) date in milliseconds since January 1, 2014, time is 00:00:00
triActiveEndDA_Max	(Number) date in milliseconds since January 1, 2014, time is 23:59:59
triActiveEndDA	(Number) date in milliseconds since January 1, 2014

Table 5. The following are the variables passed to Kettle: (continued)	
Variable that is passed to Kettle	**Description**
triCreatedByTX	(Text)
triRunDATE	(Number) Run Date set by Custom workflow task
triLocator	(Text – Locator) is a locator field that contains a reference to another business object. This variable contains the text value of that record's field
triLocator_IBS_SPEC	(Text - Locator) contains the spec_id of the record in the triLocator field. You can use this spec_id to find information related that record through other database tables

triControlNumberCN and triTransformBI are not passed to Kettle.

Important: Things to remember about variables:

- There are six variables for each Date, and Date and Time, type field. TRIRIGA wraps the value and hands it to Kettle in six different formats.
- Variables in Kettle are all strings. If you need a variable to be a number in the script, you need to use a conversion. You can set a number field like, for example, TRICREATEDDA, with a variable like, for example, triRunDATE. Kettle does some implicit conversions, but if you want to do any calculations with a variable, you must first convert the variable to a number.
- For dates, you must use the correct representation. For example, you cannot include spec.UPDATED_DATE >= ${triCreatedDA} in your selection. spec.UPDATED_DATE is a date while triCreatedDA is a number. The results are inaccurate or the SQL fails.
- The attribute types supported to pass to Kettle are limited to Text, Boolean, Date, Date and Time, and Numbers. All other TRIRIGA data types are skipped (except Locators).
- For Locator fields, two variables are created, one for the text of the Locator and the other for the SPEC_ID of the linked record. You can use the SPEC_ID to find information that is related to that record through other database tables.

Debugging ETL scripts in the application

To debug ETL scripts in the application, you must first set up logging and then trigger the RunETL Custom workflow task to view the log information.

Setting up logging

TRIRIGA provides debugging capabilities when ETL scripts run in the TRIRIGA application.

Procedure

1. In the Administrator Console, select the Platform Logging managed object. Then select the option to turn on ETL logging.
2. Select **Category ETL** > **Transforms** > **Run Transform** to turn on debug logging in the TRIRIGA platform code that processes ETL job items. Log messages are printed to server.log.
3. Select **Category ETL** > **Transforms** > **Kettle** to turn on debug logging in the Kettle transforms. Log messages are printed to the server.log.
4. Apply the changes. Now when an ETL Script runs, ETL related information will be put into the server log.

   Important: Because of the large volume of information you may encounter in a log, set Pentaho Spoon logging to debug for only one execution of the ETL job item.

Debugging using ETL jobs
Once you have set up logging, you will need a way to trigger the RunETL Custom workflow task to see any information in the logs.

About this task

If you are using the ETL Job Item, then you can simply click the **Run Process** action on that form.

Procedure

Do not forget to fill the field values in the form that the ETL Script would expect.

Only use the **Run Process** action for debugging purposes. For production, use the Job Scheduler instead. Note that Run Process will update tables in the database, so do not use this action in a production environment.

Example

The following shows a sample log output:

```
2011-01-21 14:01:27,125 DEBUG [com.tririga.platform.workflow.runtime.taskhandler.
ETL.RunETL](WFA:11325389 - 3070255 triProcessManual:38447392 IE=38447392) Kettle
variable set - ${triCalendarPeriodTX_SPEC_ID} = 3103902

2011-01-21 14:01:27,125 DEBUG [com.tririga.platform.workflow.runtime.taskhandler.
ETL.RunETL](WFA:11325389 - 3070255 triProcessManual:38447392 IE=38447392) Kettle
variable set - ${triCalendarPeriodTX} = \Classifications\Calendar Period\2010
\Q4 - 2010\October - 2010

2011-01-21 14:01:27,125 DEBUG [com.tririga.platform.workflow.runtime.taskhandler.
ETL.RunETL](WFA:11325389 - 3070255 triProcessManual:38447392 IE=38447392) *** object
field found = BoFieldImpl[name=triEndDA,id=1044,Section=BoSectionImpl[name=General,
id=BoSectionId[categoryId=1,subCategoryId=1],Business Object=BoImpl
[name=triETLJobItem,
id=10011948,module=ModuleImpl[name=triJobItem,id=22322]]]]

2011-01-21 14:01:27,125 DEBUG [com.tririga.platform.workflow.runtime.taskhandler.
ETL.RunETL](WFA:11325389 - 3070255 triProcessManual:38447392 IE=38447392) Kettle
variable set - ${triEndDA_MinDATE} = to_date
('20101031 00:00:00','YYYYmmdd hh24:mi:ss')

2011-01-21 14:01:27,125 DEBUG [com.tririga.platform.workflow.runtime.taskhandler.
ETL.RunETL](WFA:11325389 - 3070255 triProcessManual:38447392 IE=38447392) Kettle
variable set - ${triEndDA_MaxDATE} = to_date
('20101031 23:59:59','YYYYmmdd hh24:mi:ss')

2011-01-21 14:01:27,125 DEBUG [com.tririga.platform.workflow.runtime.taskhandler.
ETL.RunETL](WFA:11325389 - 3070255 triProcessManual:38447392 IE=38447392) Kettle
variable set - ${triEndDA_DATE} = to_date('20101031 00:00:00','YYYYmmdd h24:mi:ss')

2011-01-21 14:01:27,125 DEBUG [com.tririga.platform.workflow.runtime.taskhandler.
ETL.RunETL](WFA:11325389 - 3070255 triProcessManual:38447392 IE=38447392) Kettle
variable set - ${triEndDA} = 1288508400000

2011-01-21 14:01:27,125 DEBUG [com.tririga.platform.workflow.runtime.taskhandler.
ETL.RunETL](WFA:11325389 - 3070255 triProcessManual:38447392 IE=38447392) Kettle
variable set - ${triEndDA_Min} = 1288508400000

2011-01-21 14:01:27,125 DEBUG [com.tririga.platform.workflow.runtime.taskhandler.
ETL.RunETL](WFA:11325389 - 3070255 triProcessManual:38447392 IE=38447392) Kettle
variable set - ${triEndDA_Max} = 1288594799000

2011-01-21 14:02:10,595 INFO  [SpaceFact](WFA:11325389 - 3070255 triProcessManual:
38447392 IE=38447392) SpaceFact - Process Remove Nulls (LEGALINTEREST_SPEC_ID)'.0
ended successfully, processed 3282 lines. ( 76 lines/s)

2011-01-21 14:02:10,595 INFO  [SpaceFact](WFA:11325389 - 3070255 triProcessManual:
38447392 IE=38447392) SpaceFact - Process Remove Nulls ( REALPROPERTYUSE_SPEC_ID)'.
0 ended successfully, processed 3282 lines. ( 76 lines/s)

2011-01-21 14:02:10,595 INFO  [SpaceFact](WFA:11325389 - 3070255 triProcessManual:
38447392 IE=38447392) SpaceFact - Process Remove Nulls ( REALPROPERTYTYPE_SPEC_ID)'.
0 ended successfully, processed 3282 lines. ( 76 lines/s)

2011-01-21 14:02:10,595 INFO  [SpaceFact](WFA:11325389 - 3070255 triProcessManual:
38447392 IE=38447392) SpaceFact - Process Filter rows'.0 ended successfully,
```

```
processed 3307 lines. ( 76 lines/s)

2011-01-21 14:02:10,595 INFO  [SpaceFact](WFA:11325389 - 3070255 triProcessManual:
38447392 IE=38447392) SpaceFact - Process Dummy (do nothing)'.0 ended successfully,
processed 25 lines. ( 0 lines/s)

2011-01-21 14:02:10,595 INFO  [SpaceFact](WFA:11325389 - 3070255 triProcessManual:
38447392 IE=38447392) SpaceFact - Process Query for Space'.0 ended successfully,
processed 0 lines. ( 0 lines/s)
```

Performance tuning tips

Use the following tips to improve performance of ETLs with Spoon.

Summary

1. When you are finished getting your ETL to do what you want it to do, take a baseline performance measurement.
2. Using Spoon, run the ETL against a database where you have thousands of rows added to your fact table.
3. Make sure that you are using a JNDI connection and running Spoon on the network where the database lives so that you do not have network latency. Do not run it through a VPN.
4. Get a completed list of your run. For example, from a run of the triSpacePeopleFact ETL.

Analysis

1. JavaScript and DB Procedure (Get Next Spec ID) steps have multiple copies. Right-click on the step and changing the number of copies to start.
   - For the triSpacePeopleFact ETL in the preceding example run, changing the JavaScript and DB Procedure (Get Next Spec ID) steps to three copies of each:
   - Unaltered: 12.9, 12.7, 12.5, 12.6
   - Three copies of each: 11.4, 11.6, 12.3, 12.2
2. Change the default row set size from 1000 to 10000. New transformations have this set automatically. Right-click the ETL and open the properties of the transform.
3. Analyze the run. Is there a bottleneck? Is there a step that is slower than the others? Possibly other steps can have multiple copies for better throughput.
4. Is the Data Input step a bottleneck? Will an index to the database help? If so, add an index and rerun. Is the performance better? Maybe use a Filter step instead of using the database to filter down the result set.
5. Analysis is an iterative process. Always have multiple copies of the JavaScript and DB Procedure (Get Next Spec ID) steps.
6. An ETL run with 300-800 rows per second is performing well and definitely in the acceptable performance range.

For the triSpacePeopleFact ETL, after initial development substantial improvements were achieved by just doing Steps 1 and 2.

Whereas, for the triSpaceFact ETL, substantial improvements were achieved by doing Steps 1, 2, and 4.

The following shows the triSpacePeopleFact ETL run with Steps 1 and 2:

```
Query for Space People: Time = 11.6 sec; Speed (r/s) = 743.6
```

The following shows the triSpaceFact ETL run with Steps 1 and 2:

```
Query for Space: Time = 313.9 sec; Speed (r/s) = 24.0
```

Notice that it is clear that the Query for Space step, which is the Data Input step, is a bottleneck at 24 rows per second.

Notice that the Query for Space People is not a bottleneck like the Query for Space step. The triSpaceFact ETL runs well without any modifications besides Steps 1 and 2, getting over 700 rows per second.

For Step 4 on the triSpaceFact ETL, look at the SQL for the Query for Space task. Notice in the SQL that there are SUMs. SUMs are expensive, especially since there are two of them and none of the fields are indexed.

Add an index to T_TRIORGANIZATIONALLOCATION.TRILOCATIONLOOKUPTXOBJID. It is only necessary to add an index to TRILOCATIONLOOKUPTXOBJID, even though the TRISTATUSCL is in the SELECT SUM WHERE. TRISTATUSCL is a 1000 character field and made the index slow and not even viable on SQL Server.

```
CREATE INDEX IDX01_TRIORGALLOC ON T_TRIORGANIZATIONALLOCATION
(TRILOCATIONLOOKUPTXOBJID) NOPARALLEL;
```

Rerun the ETL.

The following shows the triSpaceFact ETL run with Steps 1, 2, and 4.

```
Query for Space: Time = 3.2 sec; Speed (r/s) = 2378.3
```

Notice that the change in the Data Input step rows per second (2378.3) and how long the ETL took to run (3.2 seconds for 7544 rows).

Important: Things to keep in mind while you are developing your ETLs:

- Avoid complex SQL and aggregate functions like COUNT, MIN, MAX, and SUM. If you need to use these functions, see whether an index helps out the Data Input step. Do not create an index on a field that is large varchar; SQL Server can handle only indexes < 900 bytes.
- Avoid OR and NOT and using views (M_TableName in TRIRIGA databases) if possible.
- Use the Calculator step instead of JavaScript if that is possible. The JavaScript step can be expensive.
- Have only one JavaScript scripting step.

Running ETL transforms

Use the TRIRIGA Job Scheduler to run the ETL job items and job groups that are used to move data into the TRIRIGA fact tables or flattened hierarchy tables.

ETL job items, job groups, and job schedulers

ETL job items define an ETL transformation script or rebuild hierarchy process that is used to capture information from the TRIRIGA database, transform it, and load it into fact tables. ETL job groups are simply collections of ETL job items. Job schedulers define when ETL job items and job groups are to run. A job schedule must be activated to run the jobs that are associated with it.

The TRIRIGA Workplace Performance Management analytic/reporting tool calculates metrics that are used to generate reports and graphs by using queries against fact tables that are grouped by the data in flat hierarchy tables. ETL job items are used by the Job Scheduler to trigger the workflows that extract data from source tables, such as TRIRIGA business object tables, and loading it into fact tables. ETL job items also can be used to update flat hierarchies.

There are two types of ETL job items in TRIRIGA Workplace Performance Management :

Kettle Transformation
Extracts data from TRIRIGA business object tables and loads the data into TRIRIGA Workplace Performance Management fact tables.

Rebuild Hierarchy
Extracts data from TRIRIGA business object tables and loads the data into TRIRIGA Workplace Performance Management flattened-hierarchy tables.

Creating or modifying ETL job items

ETL job items define the scripts that capture information from the TRIRIGA database, transform it, and load it into fact tables or flattened hierarchy tables.

Procedure

1. Click **Tools** > **System Setup** > **Job Scheduling** > **ETL Job Item**.
2. Select an existing job item or click **Add**.
3. In the General section, enter an ID for the ETL job item.

   Include the fact table name in the **ID** field. The status is supplied by the TRIRIGA system when the ETL job item is created.
4. Enter a name and description for the ETL job item.

   In the Details section, the **Job Item Class** field is set to ETL by the TRIRIGA system.
5. Select the **Job Item Type**.
6. If the job item type is **Rebuild Hierarchy**, complete the following steps.
   a) Enter the hierarchy module name.

      If specified, it must be a module with a hierarchy defined in TRIRIGA, such as Location. When this ETL job item runs, all flat hierarchies for this module are rebuilt.
   b) Enter the hierarchy name.

      When this ETL job item runs, the flat hierarchy for this business object is rebuilt. If specified, the hierarchy name takes precedence over the hierarchy module name.
   c) The TRIRIGA system ignores information that is entered in the calendar period, fiscal period, and other dates in the Details section.
   d) To rebuild all flat hierarchies of a specific module, specify the hierarchy module name and leave hierarchy name blank.
   e) To rebuild a single flat hierarchy, specify both the hierarchy module name and hierarchy name.
   f) If both the hierarchy module name and hierarchy name are blank, or either contains All, all flat hierarchies are rebuilt.
7. If the job item type is **Kettle Transformation**, complete the following steps.
   a) Specify the transform file by browsing for and selecting the file.

      The TRIRIGA system expects the Kettle transform file to be in .ktl or .xml format.
   b) Optional: After you upload the transform file, it can be viewed by clicking **View Content**.
8. If the job item type is **Kettle Transformation**, complete the following steps.
   a) Enter the module names. If more than one name is specified, they must be delimited with a comma.
      - Each module name is converted into a variable for the transform file in the format *${Module.<moduleName>.ViewName}* where *<moduleName>* is the module name.
      - Each variable's value that is passed into the ETL is the name of the View for that module. This variable's value can be used if the ETL must know the name of a specific Module's view.
   b) Enter the business object names. If you specify more than one business object name, the names must be delimited with a comma.
      - Each business object name is converted into a variable for the transform file in the format *${BO.<boName>.TableName}* where *<boName>* is the business object name.
      - A business object is not guaranteed to be unique across the database unless the module name is included. If you use a business object that is not uniquely named, include the module name in the comma-separated list. Use the following syntax: *<moduleName>::<boName>*, where *<moduleName>* is the module name and *<boName>* is the business object name.

- A variable is provided to the transform file as *${BO.<moduleName>::<boName>.TableName}*. Each variable's value is the name of the table for that business object. This variable's value can be used if the ETL must know the name of a specific Business Object's table.

9. If the job item type is **Kettle Transformation**, complete the following steps.
   a) Typically, ETL job items are run under the control of one or more job schedules.
   b) To unit test the ETL job item, set the date parameters in the Details section.
   c) The following date parameters depend on the ETL, some ETLs use the information, some ETLs do not. The date parameters are overwritten when an ETL job item is run by the Job Scheduler.
      - Select the **Calendar Period** to pass a variable that contains the calendar period of the job.
      - Select the **Fiscal Period** to pass a variable that contains the fiscal period of the job. The fiscal period is used by the **Capture Period** field in the TRIRIGA Workplace Performance Management fact tables.
      - Select the **Date** to pass a variable that contains the date record of the job. The date record is used to stamp the **Date Dimension** field in the TRIRIGA Workplace Performance Management fact tables.
      - Select the **Date** to pass a variable that contains the date of the job. Enter a date or click the **Calendar** icon and select a date.
      - Select the **Start Date** to specify the date of the first data capture and to pass a variable that contains the start date of the job. Enter a date or click the **Calendar** icon and select a date.
      - Select the **End Date** to specify the date of the last data capture and to pass a variable that contains the end date of the job. Enter a date or click the **Calendar** icon and select a date.
10. The Metrics section summarizes logging data for this ETL job item. The **Average Duration** is calculated based on the **Total Duration** and the **# of Runs** (**Total Duration** / **# of Runs**).
11. The Logs section shows the time and status from each time the ETL job item is run. This data is summarized in the Metrics section.
12. Click **Create Draft**.
13. Click **Activate**.

Results
The ETL job item record is created and ready to be included in a job group, a job schedule, or both.

What to do next
For unit testing, click **Run Process** to trigger the Kettle transform or the Rebuild Hierarchy process that is specified in the ETL job item.

Adding or modifying job groups

To simplify job scheduling, use job groups to make collections of ETL job items to be run on the same schedule.

Procedure
1. Click **Tools** > **System Setup** > **Job Scheduling** > **Job Group**.
2. Select an existing job group or click **Add**.
3. In the **Job Group** form, enter a name and description for the job group.
4. Add or remove ETL job items from the job group in the Job Items section by using the **Find** or **Remove** actions.
5. Adjust the sequence of the job items.
6. Click **Create**.

Results
The job group record is available to include in a job schedule.

Creating or modifying job schedulers

Use job schedulers to schedule when TRIRIGA runs ETL job items and job groups. You can schedule jobs to run on an hourly, daily, weekly, or monthly basis.

Before you begin
Although TRIRIGA includes predefined job schedulers, no jobs are scheduled until you revise the predefined job schedulers or create a new job scheduler with a start date and end date and click **Activate**.

Procedure
1. Click **Tools** > **System Setup** > **Job Scheduling** > **Job Scheduler**.
2. Select an existing job scheduler or click **Add**.
3. Enter a name and description for the job scheduler.
4. In the Schedule section, select the **Schedule Type** frequency from the list presented.
   - If you select the **Daily** schedule type, the **Hourly** and **Every** fields are displayed. Click the **Hourly** check box to schedule jobs to run hourly, then click the **Every** field to specify the number of hours between each scheduled job.
   - If you select the **Advanced** schedule type, the **Recurrence Pattern** field displays in the Schedule section. Click **Recurrence Pattern** to open the **Job Event** form, which provides flexible options for scheduling jobs.
5. In the Schedule section, select the **Run Historic Captures?** check box to indicate that historic data is included in the metrics that are calculated from the results of the activities in this job schedule.

   When this job schedule is activated, this option instructs the job scheduler to generate and run jobs where the scheduled date is earlier than today. The parameters, such as start date, end date, and fiscal period, for the time frame or period are passed into each job item to simulate a historic capture. However, since the process is being run now, success is entirely dependent on how the ETL scripts use these parameters. Running historic captures is an advanced option that requires a complete understanding of how each script works.
6. Select the starting date for the first capture, the ending date for the last capture, and the capture lag from the end date of each capture, which is the amount of time that the system waits to start the workflows that process the job.

   Each data capture period is calculated by using the start date and the schedule type. The system triggers a workflow to run the ETL job items immediately after the capture lag from the end of each capture period.

   For example, if the start date is 01/01/2014, the schedule type is monthly, and the capture lag is one day, the following events are scheduled:

Table 6. Job runs scheduled from the beginning of the month

Job runs - Just after midnight on	Captured data start date	Captured data end date
02/01/2014	01/01/2014	01/31/2014
03/01/2014	02/01/2014	02/28/2014
04/01/2014	03/01/2014	03/31/2014

   The End Date determines the last event to capture.

   Using the preceding example, if 03/15/2014 were the End Date, the last event would be scheduled as follows:

Table 7. Job runs scheduled from the middle of the month		
Job runs - Just after midnight on	**Captured data start date**	**Captured data end date**
03/16/2014	03/01/2014	03/15/2014

- When the **Reset Capture Period?** check box is selected, it forces the system to ensure that the capture period is kept current every time a job runs.
- For example, if an activated job scheduler is configured with schedule type of monthly, the system wakes up every month and runs the job items in the record. During the wake-up, if the Job Scheduler's reset capture period is selected, the system ensures that the capture period is set correctly based on the wake-up date.
- When you specify job items, job groups, or both and activated this job schedule record, the list of scheduled events for this Job Schedule shows in the Scheduled Jobs section.

7. In the Job Items section, use the **Find** or **Remove** actions to select the ETL job items and job groups to include in the schedule and click **OK**.

   The job items run in the order that is specified in the Sequence column.

   - When this job schedule is activated, the Metrics section summarizes logging data for this job schedule.
   - The Average Duration is the calculated based on the Total Duration and the Number of Runs (Total Duration / Number of Runs).
   - When this job schedule is activated, the Logs section shows the time and status from each time this job schedule is run.
8. Optional: Adjust the sequence of the job items.
9. Click **Create Draft** and then click **Activate**.

Results

The TRIRIGA system creates a set of scheduled jobs that are based on the schedule type and the start and end dates. These can be seen in the scheduled jobs section of the job schedule.

Customizing transform objects

TRIRIGA provides ETL job items and transform objects. Rather than defining a new transform object, you can customize an existing ETL job item transform object. If you use an existing transform object, then you must define or maintain the transform. However, you do not need to define or maintain the business objects, forms, or workflow tasks, as they are already defined.

Defining transform business objects, forms, and workflows

You can use the existing TRIRIGA ETL job item as an example, when you define a new transform object. Defining a new own transform object also requires that you define and maintain the business objects, forms, and workflows.

Before you begin

Create the source and destination tables (business objects) and establish the corresponding mappings.

Procedure

1. Create the transform business object.
   a) Identify variables to be passed into the transform and add them to the transform business object. For example, time period.
   b) Ensure that there is a binary field for the transform XML.

2. Create the transform form and provide a navigation item, or other method, to show the transform form.
3. Create the workflow that calls the ETL transform custom workflow task.
   a) Set up the workflow to run on a schedule.
   b) Iterate through all the transform business object records that must be run, calling the custom workflow task for each one.

Saving transform XML into the Content Manager

After you define the transform XML, you can save it in the file system and upload it into the TRIRIGA Content Manager. The transform XML can then be tested by using an ETL development environment.

Procedure

1. Open the navigation item for your transform business object. Edit an existing transform business object or add a new one.
2. Use the binary field to upload the transform XML into the TRIRIGA Content Manager.
3. Update other fields, if necessary.
4. Save the transform record.

Configuring workflow run time

After you upload the transform XML into the TRIRIGA Content Manager, you can and set up the workflow to run on a schedule. The workflow iterates through all the transform business object records and calls the custom workflow task for each record.

Procedure

1. The workflow gets records for the transform business object.
2. The workflow determines the records to be sent to the custom workflow task.
3. The workflow iterates through calling custom workflow task for each record. The class name must be **`com.tririga.platform.workflow.runtime.taskhandler.ETL.RunETL`**.
4. The custom workflow task:
   a) Loads the transform XML from the Content Manager into a temporary file.
   b) Gathers all the fields on the business object and creates a variable to hand to the ETL tool. Special handling is needed for date/date and time formats.
   c) Creates the ETL environment.
   d) Sets the TRIRIGA connection to the local application server JNDI.
   e) Runs the transform by using the ETL API.
   f) Returns false if an error occurs during processing, otherwise return true to the workflow.

Running an ETL custom workflow task specification

The workflow iterates through all the transform business object records and calls the custom workflow task for each record. The custom workflow task includes a defined specification.

When the custom workflow task is called for each transform business object, the fields on it are processed as follows:

1. The **`triTransformBI`** field is required and holds the reference to the transform XML file that you want to run.
2. The **`triBONamesTX`** field, if present, is parsed as a comma-separated list of business object names. The custom workflow task creates variables of the form *${BO.<boName>.TableName}*. For example, if the field contains triBuilding, there is a *${BO.triBuilding.TableName}* variable available in the ETL script. This variable contains the actual database table name that stores triBuilding records. Since business object names might not be unique, you have the option of specifying the module by using the form

<moduleName>::<boName>, which results in a corresponding *${BO.<moduleName>::<boName>.TableName}* variable. For example, Location::triBuilding is available as the variable *${BO.Location::triBuilding.TableName}* in the ETL script.

3. The **triModuleNamesTX** field, if present, is parsed as a comma-separated list of module names. The custom workflow task creates variables of the form *${Module.<moduleName>.ViewName}*.

Chapter 3. Metrics

A metric is an operational objective that you want to measure. All TRIRIGA metrics use the same metric technology and data but different metrics are intended for different purposes.

Metrics can be divided into the following two purposes:

Performance metrics
Metrics that measure process performance to identify actions that can be taken for improvement. Typically, the measures are ratios, percentages, or scores. Performance metrics have targets, thresholds, action conditions, accountability, and action task functions.

Analysis and reporting metrics
Metrics that are information for general reporting or further analysis of a related performance metric. This information is useful for dimensional analysis and navigation, so it uses the metric capabilities of the performance management application. Analysis and reporting metrics do not have targets, thresholds, action conditions, accountability, and action tasks. The key metrics values for the **Results**, **Target**, and **Status** fields are blank for this metric type.

For more information, see the *IBM TRIRIGA 10 Workplace Performance Management User Guide*.

Metrics reports

Metric reports use the dimensions and fact fields that are defined in the fact tables to represent a specific metric calculation. Metric reports show aggregated values for a single metric.

Collections of metric reports for each user's role appear in the portal Home page and in the Performance Manager.

Do not edit or modify the as-delivered metric reports as you customize TRIRIGA software to your company's environment. Instead, create a derivative metric report by copying an existing metric report, renaming it, and editing it, or by creating an entirely new metric report. View the metric reports at **My Reports** > **System Reports** with the Module filter set to triMetricFact, the Display Type filter set to Metric, and the Name filter set to Metric. To view the metric tabular reports that are related to metric graphic reports, set the Name filter to Related Reports.

Each metric report consists of the following elements:

Drill paths
A special reporting feature that takes advantage of hierarchical dimensions, providing report users with the ability to move up and down a hierarchy.

Filters
Provide report users with the ability to change or filter the data that is presented in a report.

Metric calculation
The metric, which is the main focus of the report.

Related reports
Display more, possibly non-metric, data for a metric report.

For more information, see the *IBM TRIRIGA Application Platform 3 Reporting User Guide*.

Key metrics

The Scorecard portal includes Key Metrics sections that collect multiple metrics into a single view.

Do not edit or modify the as-delivered key metrics as you customize TRIRIGA for your organization. Rather, create derivative key metrics by copying an as-delivered key metric, renaming it, and editing it. You can also create an entirely new key metrics portal section.

The process of using the Scorecard Builder, Portal Builder, and Navigation Builder to set up a portal is documented in the *IBM TRIRIGA Application Platform 3 User Experience User Guide*.

Tips:

- In the **Queries** tab of the Scorecard form, include all queries that are to appear in this Key Metrics portal section in the Selected Queries section.
- When you put the Scorecard in a navigation item, specify the Default Report.
- The response time that users experience while they moving around in their Home portal and their Performance Manager can vary. The response time is directly related to the number of metrics that are included in the Scorecard and the amount of data behind each metric. The more precise the filters, the more performance improves.

Form metrics

An alternative method to displaying metrics is by using a form view.

Displaying metrics in a form is accomplished by defining a query section in a form, where the query referenced is a metric query. The data that the metric query selects can be filtered to display based on the parent record that the form is displayed in. Related switching of reports allows the exchanging of queries that are based on user actions.

At design time, define a metric query with a **$$RECORDID$$** and **$$PARENT::[Section]:: [Field]$$** filter. At runtime, the parent record the form is displayed in implicitly filters the data that the Metric Query Engine selects to display.

You can define other metric queries as related reports. At runtime, when the metric query is displayed within a form-related report, switching control enables the user to exchange the displayed query.

Data filtering

Before metrics are displayed in a form, the data is filtered.

At runtime, before the query is run,

- The **$$RECORDID$$** filter is replaced by the record ID of the parent record that the form is being displayed within.
- A **$$PARENT::[Section]::[Field]$$** filter is resolved to the defined parent record field's value.
- If a metric query is displayed outside of a parent record, any parent-sensitive filters, such as **$$RECORDID$$** and **$$PARENT::[Section]::[Field]$$** are ignored.

Define **$$RECORDID$$** and **$$PARENT::[Section]::[Field]$$** filters against fields that are defined as a drill path in the fact table. The metric that is filtered for a program or a classification, and that uses these filters, acts similar to selecting from the drill path list. It also filters for the specified record and includes all children of that record.

When a **$$PARENT::[Section]::[Field]$$** filter value is specified on a hierarchical filter, a null parent field is the equivalent of choosing the root node of the drill path. It also includes all of the records that match the root value. A non-hierarchical filter behaves much like a business object query filter and filters for records with null value when the parent field value is null.

$$RECORDID$$** and **$$PARENT::[Section]::[Field]$$ filters behave as if they are runtime filters except that their value is passed from the parent record.

If a metric query has **$$RUNTIME$$** and **$$PARENT::[Section]::[Field]$$** filters that are defined against the same field, when the query is displayed inside a form, the **$$RUNTIME$$** filter control is not used. Instead, the value from the**$$PARENT::[Section]::[Field]$$** is used as the filter value.

Sub reports

Form metrics can include both tabular and graphical sub reports.

A tabular sub report defines a Performance Manager's related report. A non-tabular sub report represents a metric graph that can be switched within the form.

Both tabular and graphical reports can be added to the Sub Reports section of a metric query. At runtime, a Performance Manager displays only tabular reports as Related Reports, and a Metric in a form query section has only graphical reports as options in the Sub Report section swap controls.

The type of the sub report (for example, metric, query, or report) displays in the Sub Reports section of the Report Builder. For metric queries, the Tabular Output flag is available.

A metric query can display other metric queries as sub reports.

Chapter 4. Hierarchy flattener

The definition of a hierarchy in the TRIRIGA Application Platform provides flexibility for implementations with multiple permutations of data hierarchies, for example, organizations. However, reporting tools prefer a more structured data summary to simplify reporting and maximize performance.

The purpose of the hierarchy flattener tool is for an administrator to define a set of named structures that are used by the TRIRIGA Metric Reporting engine to quickly process the hierarchical data within the system. The resulting structure is referred to as a flat hierarchy and these structures are used in a metric reports.

Administrators use the job scheduler, with the help from ETL job items, to run the hierarchy flattener process. System developers trigger the process by using a custom workflow task. In this case, the class name that is used to run the hierarchy flattener process is

```
com.tririga.platform.workflow.runtime.taskhandler.flathierarchy.
RebuildFlatHierarchies
```

Triggering the process by using a custom workflow task is discussed in *Application Building for the IBM TRIRIGA Application Platform 3*.

Important: Whenever a hierarchy in TRIRIGA is changed, the hierarchy tree is promptly updated. For example, suppose that you add a new *triBuilding*, then the Location hierarchy is updated. However, the corresponding flat hierarchy for *triBuilding* is not updated until you rebuild it with a Rebuild Hierarchy Job Item. Therefore, it is important to schedule rebuild hierarchy job items at the same time as when you plan to capture TRIRIGA Workplace Performance Management data through fact tables. Rebuild hierarchy job items ensure that you keep information current.

Flat hierarchies

Hierarchy structure definitions depend on what the flattening is based on. The flat hierarchy can be based on the standard parent-child relationships for a specified module's hierarchy. The flat hierarchy can also be based on specific levels in the module's hierarchy, and their respective business objects.

Each hierarchy structure definition contains a single header record that identifies hierarchy name, module, and hierarchy type. The hierarchy name describes the hierarchy and the module is the module that the hierarchy represents. The hierarchy type is used by the flattening process to understand how to flatten the data. There are two hierarchy types, **Data** and **Form**.

A data hierarchy is used to flatten the path of data based on the standard parent-child relationships for the specified module's hierarchy. This hierarchy type has no named levels because TRIRIGA Application Platform applications allow different types of data to be represented at the same physical level in a module's hierarchy. For example, a location hierarchy might have data for both property, building and floor, and for building and floor. Thus the first level in the hierarchy would contain a mixture of properties and buildings, and the second level would contain a mix of buildings and floors.

A form hierarchy is used to flatten the path of data based on the parent-child relationships for the specified module's hierarchy and the business objects that represent levels. Only one business object can represent each level.

Each form hierarchy must specify explicit levels that contain the level number, the business object that the level represents, and the type. The type is used by the flattening process to understand how to find the data for the level. The type has three options: **Find**, **Ignore**, and **Recurse**.

- When the type value is **Find**, the system searches through the sublevels of the instance data for a particular thread until a record is found for the specified form. If no records are found, the remaining levels in the hierarchy definition are ignored and no more flat data is created for that thread. If a record is found, the system creates a flat data record for that node and proceeds to the next level in the definition. This mode provides the capability to collapse a tree to better align your business data.

- When the type value is **Ignore**, the system searches for the specified form, one level below the last parent. If a record is not found, the system creates a gap for this level and proceeds with the next level in the definition. If a record is found, the system creates a flat data record for that node and proceeds to the next level in the definition. This mode provides the capability to expand a tree to better align your business data. To facilitate the reporting process, the gaps must be given a name or label. Use the **Gap Label** value in the Hierarchy Structure Manager for this purpose.
- When the type value is **Recurse**, the system searches through the sublevels of the instance data for a particular thread until a record is found for the specified form. If no records are found, the remaining levels in the hierarchy definition are ignored and no more flat data is created for that thread. For each record found, the system creates a flat data record for that node before it proceeds to the next level in the definition.

Examples of flat hierarchies

You can reference flat hierarchy examples to better understand the structure of a flat hierarchy definition.

Sample flat hierarchy header records

The following table shows examples of flat hierarchies that are based on modules within hierarchies:

Table 8. Sample header records

Hierarchy name	Module	Hierarchy type
Space Hierarchy	Location	GUI
Land Hierarchy	Location	GUI
City Hierarchy	Geography	GUI
Full Location Hierarchy	Location	Data
Full Organization Hierarchy	Organization	Data
Internal Organization Hierarchy	Organization	GUI
External Organization Hierarchy	Organization	GUI

Sample flat hierarchy level records

The following table shows examples of flat hierarchies that are based on levels within hierarchies:

Table 9. Sample flat hierarchy level records

Hierarchy name	Level number	Form	Find module
Space Hierarchy	1	Property	Ignore
Space Hierarchy	2	Building	Find
Space Hierarchy	3	Floor	Find
Space Hierarchy	4	Space	Recurse
Internal Organization Hierarchy	1	Company	Find
Internal Organization Hierarchy	2	Division	Ignore
Internal Organization Hierarchy	3	Department	Recurse

Hierarchy structure manager

You can define hierarchies and level information by using the Hierarchy Structure Manager. The Hierarchy Structure Manager provides a single interface for creating, updating, and deleting flat hierarchies.

Accessing hierarchy structures

To add, modify, or delete hierarchies, access the hierarchy structures functionality.

Procedure

1. Click **Tools** > **Builder Tools** > **Data Modeler**.
2. Click **Utilities**.
3. Click **Hierarchy Structures**.

Creating a data hierarchy

A data hierarchy is used to flatten the path of data based on the standard parent-child relationships for the specified module's hierarchy. When you create a data hierarchy, named levels are not required as different types of data can be represented at the same physical level in a module's hierarchy.

Procedure

1. Click **Create Hierarchy**.
2. In the **Name** field, enter a name to describe what the hierarchy represents.
3. From the **Module** list, select the relevant module for the data hierarchy.
4. From the **Hierarchy Type** list, select **Data**.
5. Click **Create**.
6. Click **Save**, then click **Close**.

Creating a form hierarchy

A form hierarchy is used to flatten the path of data based on the parent-child relationships for the specified module's hierarchy and the business objects that represent levels. When you create a form hierarchy, only one business object can represent each level.

Procedure

1. Click **Create Hierarchy**.
2. In the **Name** field, enter a name to describe what the hierarchy represents.
3. From the **Module** list, select the relevant module for the form hierarchy.
4. From the **Hierarchy Type** list, select **Form**.
5. Click **Create**. The Levels section displays. Enter information for the level 1 form.
6. From the **Business Object** list, select the relevant business object.
7. From the **Form** list, select the relevant form.
8. From the **Type** list, select **Find**.

   The **Gap Label** is the label that is specified when **Ignore** is selected from the **Type** list and a record is not found.
9. Click **Save**.
10. Continue entering and saving information until all levels are defined.
11. Click **Save**, then click **Close**.

Chapter 5. Fact tables

Fact tables consist of the measurements, metrics, or facts of a business process. Fact tables store the data that is used to calculate the metrics in metric reports.

Fact table information is based on the as-delivered TRIRIGA Workplace Performance Management and TRIRIGA Real Estate Environmental Sustainability products. The implementation at your company might be different.

Each fact table has an ETL to load data and another to clear data. The names of the ETLs that clear data end with – Clear, for example **Building Cost Fact – Clear**. To view the ETLs, click **Tools** > **System Setup** > **ETL Job Items**.

List of fact tables and metrics supported

You can reference the list of fact tables and metrics that are supported in the TRIRIGA implementation at your company.

Accessing fact tables

Click **Tools** > **Builder Tools** > **Data Modeler**.

Locate *triMetricFact* and select it to reveal the list of fact table business objects.

Accessing metrics

Click **Tools** > **Builder Tools** > **Report Manager** > **System Reports**.

Filter by **Business Object** and **Module** to sort the metric information and obtain the relevant list of reports.

Tip: #FM# means that a metric is also a form metric in the system.

Facts that require special staging tables and ETLs

For most fact tables, the process to load the stored data to calculate metrics is simple. However, some fact tables require special staging tables, and ETLs to assist with the loading process.

The following table shows the facts that require special staging tables and ETLs:

Table 10. Facts that require special staging tables and ETLs

Fact table name	**Fact table business object**
Financial Summary	triFinancialSummary
Standard Hours	triStandardHours
Standard Hours Details	triStandardHoursDetails
Asset Analytic Hourly Fact	triAssetAnalyticHFact
Asset Energy Use Daily Fact	triAssetEnergyUseDFact
Asset Energy Use Hourly Fact	triAssetEnergyUseHFact
Asset Energy Use Monthly Fact	triAssetEnergyUseMFact

Dependent ETLs

Some ETLs depend on other ETLs to assist with the loading process.

The following table shows the ETLs that depend on other ETLs:

Table 11. Dependent ETLs

Fact table name	Fact table business object
Building Cost Fact Load ETL	This ETL depends on the availability of data in the Financial Summary Table. The Financial Summary Table can be loaded either by backend integration with your financial system or by using the Offline Financial Summary Excel process. To Facilitate the Offline Financial Summary Excel process, there is a special ETL to push the data from the Excel/Offline process to the Financial Summary Table. In the as-delivered TRIRIGA Workplace Performance Management, the special ETLs are named Load Financial Summary From Offline Staging and Clear Financial Summary From Offline Staging. If you are importing financial summary data with the Offline Financial Summary Excel process, you must first run the Load Financial Summary From Offline Staging ETL. You must then run the Building Cost Fact Load ETL.
Building Fact Load ETL	This ETL depends on the availability of data in the Financial Summary Table. The Financial Summary Table can be loaded either by backend integration with your financial system or by using the Offline Financial Summary Excel process. To Facilitate the Offline Financial Summary Excel process, there is a special ETL to push the data from the Excel/Offline process to the Financial Summary Table. In the as-delivered TRIRIGA Workplace Performance Management, the special ETLs are named Load Financial Summary From Offline Staging and Clear Financial Summary From Offline Staging. If you are importing financial summary data with the Offline Financial Summary Excel process, you must first run the Load Financial Summary From Offline Staging ETL. You must then run the Building Fact Load ETL.
Resource Fact Load ETL	Dependent on Standard Hours Load ETL.
Standard Hours Load ETL	Dependent on Standard Hours Details Load ETL.
Asset Daily Fact ETL Asset Hourly Fact ETL Asset Monthly Fact ETL	These ETLs depend on the availability of data in a staging table. The staging table types must be generic. The staging table must include specific fields. The staging table can be loaded by back-end integration with the building management system. The staging table can also be loaded by using an ETL to bring the data in from an external database. See the Integrated Service Management Library for more details including, sample staging tables and sample ETLs.

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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.

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 actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work
must include a copyright notice as follows:
© (your company name) (year).
Portions of this code are derived from IBM Corp. Sample Programs.
© Copyright IBM Corp. _enter the year or years_.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux® is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other product and service names might be trademarks of IBM or other companies.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a 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 http://www.ibm.com/privacy and IBM's Online Privacy Statement at https://www.ibm.com/privacy/details/us/en/ in the section entitled “Cookies, Web Beacons and Other Technologies.”

Part Number:

(1P) P/N: