IBM TRIRIGA Application Platform 3.7.0

Connector User Guide

Note

Before using this information and the product it supports, read the information in “Notices” on page 47.

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. Integrating data with external applications

You use the IBM TRIRIGA Application Platform connector tools to import, update, or export data from IBM TRIRIGA; or link or share data with IBM TRIRIGA applications. These tools include the TRIRIGA integration object, IBM TRIRIGA Connector for Esri Geographic Information System (GIS), IBM TRIRIGA Connector for Business Applications SOAP API, IBM TRIRIGA DataConnect, IBM TRIRIGA Data Integrator, and OSLC REST API integration.

About this task

The following integration technologies are provided as part of the on-premises IBM TRIRIGA Connector for Business Applications (CBA) license or the IBM Facilities and Real Estate Management on Cloud (TRIRIGA) Enterprise User license:

- IBM TRIRIGA Connector for Business Applications SOAP API
- IBM TRIRIGA OSLC REST API
- TRIRIGA integration object

Chapter 2. Integrating data with the TRIRIGA integration object

The TRIRIGA integration object is a utility business object in the `triIntegration` module that contains the metadata that controls the integration between IBM TRIRIGA and external systems. When you create integration definitions with the TRIRIGA integration object, you use standard IBM TRIRIGA tools such as workflows and queries to build integration connections.

Overview of the TRIRIGA integration object

When you use the integration object to define a new integration definition, you must select the data scheme. Depending on your scheme selection, you must define several elements that can include the data source, import or export file, outbound query, data map, or response map.

Integration object records

To define a new integration definition, you must create an integration object record from the TRIRIGA integration object. To view existing integration object records, select **Tools** > **System Setup** > **Integration** > **Integration Object**.

Data schemes

When you define a new integration definition, you must select from several options or schemes to define the payload, protocol, and transport for the data.

The database scheme uses database tables to import or export data. The file scheme uses files to import or export data. The DataConnect scheme imports files into DataConnect staging tables. The HTTP post scheme exports data with the HTTP protocol.

Database scheme

When you select the inbound database scheme, you must define the data source and the data map. The data map is used to map the data from the external source to existing IBM TRIRIGA fields.

When you select the outbound database scheme, you must define the data source and the outbound query. The outbound query is used to define which fields are exported from your IBM TRIRIGA database.

File scheme

When you select the inbound file scheme, you must define the import file and the data map. The data map is used to map the data from the external source to existing IBM TRIRIGA fields.

When you select the outbound file scheme, you must define the export file and the outbound query. The outbound query is used to define which fields are exported from your IBM TRIRIGA database.

DataConnect scheme

When you select the inbound IBM TRIRIGA DataConnect scheme, you must define the data source, the import file, the DataConnect job, and the data map. The data source is used to access the DataConnect staging tables. The data map is used to map the data from the file to the DataConnect staging tables.

With this scheme, you can use workflows to process and validate data, which gives you more control over error handling.

For information about DataConnect, see *Application Building for the IBM TRIRIGA Application Platform: Data Management*.

HTTP post scheme

When you select the outbound HTTP post scheme, you must define the server to which the data is sent, the outbound query, and the response map. The outbound query is used to define which fields are exported from your IBM TRIRIGA database. The response map is used to map the response parameters from the HTTP request to your existing IBM TRIRIGA fields.

Database tools

When you select the database scheme or DataConnect scheme, you can test the database connection, generate the test data, or generate the SQL for the database table. To export binary large object fields, you must use a dynamic outbound query.

Database connection testing

You select the **Test DB Connection** action to verify that the server can communicate with the database. The table name is used to run a `select 1+1 from [table name]` query to the database. The color of the Database section header changes to red if an error occurred. To view the errors, review the server logs in the IBM TRIRIGA administrator console. For information about the administrator console, see the *IBM TRIRIGA Application Platform 3 Administrator Console User Guide*.

Required table access for the DataConnect scheme

The DataConnect scheme is separate from the IBM TRIRIGA internal APIs and uses the IBM TRIRIGA Connector for Business Applications to communicate with TRIRIGA. The DataConnect scheme allows access to the database to inspect, read, and write to the tables.

The DataConnect scheme requires access to the following tables. Except for the DC_JOB table and S_ table, the following tables are used to determine the columns that are available for the DataConnect objects. The same tables must have Read capabilities for the user that is selected in the data source.

IBS_SPEC_TYPE_STAGE
Read access.

IBS_SPEC_TYPE
Read access.

SYS.COLUMNS (Microsoft SQL Server)
Read access.

SYS.TABLES (Microsoft SQL Server)
Read access.

IBS_MODULE
Read access.

ALL_TAB_COLUMNS (Oracle Database)
Read access.

DC_JOB
Read and write access.

S_
Read and write access.

Generic SQL creation

You select the **Generate SQL for Table** action to create the generic SQL to define your database table. Before you select this action, enter the name of the database table and complete the data map. Each table that is used for inbound integration must have the following columns:

IMD_ID
The transaction ID that is unique to each entry in the table. This ID is not the same as the record ID.

IMD_STATUS
Identifies the status of each entry in the table. The default status of Ready identifies records to be processed. Other states include Processing, Completed, and Failed.

IMD_MESSAGE
Identifies any errors in importing the row. The same error messages are also displayed in the Integration Object form.

TRIRIGA_RECORD_ID
The internal record ID that IBM TRIRIGA uses to identify the record that is being updated or created.

The following sample code was generated for a people-record inbound integration with the previous fields.

```
-- Auto generated script for MSSQL.
CREATE TABLE example_people_in (
    IMD_ID NUMERIC (18, 0) IDENTITY(1,1) NOT NULL,
    IMD_STATUS VARCHAR(1000) COLLATE SQL_Latin1_General_CP1_CS_AS null DEFAULT 'Ready',
    IMD_MESSAGE VARCHAR(1000) COLLATE SQL_Latin1_General_CP1_CS_AS null,
    TRIRIGA_RECORD_ID VARCHAR(1000) COLLATE SQL_Latin1_General_CP1_CS_AS null,
    TRI_ID VARCHAR(1000) COLLATE SQL_Latin1_General_CP1_CS_AS null,
    FIRST_NAME VARCHAR(1000) COLLATE SQL_Latin1_General_CP1_CS_AS null,
    LAST_NAME VARCHAR(1000) COLLATE SQL_Latin1_General_CP1_CS_AS null,
    LANGUAGE VARCHAR(1000) COLLATE SQL_Latin1_General_CP1_CS_AS null
)
```

Test data creation

You select the **Generate Test Data** action to load your database with test data. This action is only available in the inbound direction.

After you create the database table and can connect to it, you can load your database with test data. You use the test data to run simple functional testing to verify that the mappings are correct and that the integration process works correctly. Enter the number of rows to be inserted into the database table in the **Test Rows** field. The data is generated as alphanumeric values. If you specify a default value in your data map, that value is inserted for all rows. This default value is useful when the field might be a locator field, a number field, a date field, or a list that requires a specific value.

If an external system is used to populate the staging database, the external system needs to populate the non-metadata fields only. The IMD_* columns are auto-populated. After the record is updated or created, the TRIRIGA_RECORD_ID column contains the record ID that identifies this record in IBM TRIRIGA internally.

Binary large object (BLOB) support

BLOBs are supported for both inbound and outbound database schemes. Supported BLOB types include binary fields, note fields, and document manager content. Image fields are not supported. To export BLOB fields, you must use a data map as a dynamic outbound query.

When you import binary fields and note fields in the data map, they are imported like other field types. The **External** attribute value for the BLOB field must match the database column name.

When you import or export document manager objects, the integration object must be configured to interact with the document business object of the document module. For instance, the data map for importing documents must be configured to map to the document business object.

No actual value exists for the binary content of the document on the object. Instead, you select any base object column, such as a text field in the General section, to use as a placeholder for your content. The external name can be whatever you want, but the value in the Default column must be CONTENT. This value is the trigger that connects the document manager object to the content.

File tools

When you select the file scheme or DataConnect scheme, you can test the file connection. You can read from or write to anywhere on the network that the server can access.

Inbound integrations

For inbound integrations, you can import standardized delimited flat files. You can manually provide a binary field that is a copy of the integration field that is imported. You can provide a manual field to help in testing so that local processing can be repeated without overwriting the import file. After the file is imported, it is renamed with the processing date and moved to the processed folder.

If multiple workflow agents are running in your IBM TRIRIGA environment, all of the servers must either have access to the same file location or their own copy of the file. The workflow that runs during the integration execution process must have access to the file regardless of which workflow agent starts the workflow.

Outbound integrations

For outbound integrations, you can export file formats that include standardized delimited flat files and `.json`, `.xml`, or `.xslt` files.

HTTP post tokens

When you select the outbound HTTP post scheme, you can add tokens to the values in the **Http URL**, **Http URI**, and **Headers** fields. You can also add tokens to the **Parameters** field, which is only visible for the **Parameter** post type.

A token is a value that is sourced from the outbound query results for the HTTP request. The token name must exactly match the column label in your query, and must be surrounded by curly brackets.

For example, to add the **accountId** column value from your outbound query results to the **Http URI** field, enter `/ws/account/{accountId}/property` in the **Http URI** field. At run time, IBM TRIRIGA renders the value that is returned in the query result. If the value of **accountId** is 123, then the URI that is sent is `/ws/account/123/property`.

When a batch request is made with tokens, the values in the last result row provide the tokens.

Outbound queries

For outbound integrations, you must define an outbound query to select which fields are exported from your IBM TRIRIGA database. You can define an outbound query from the IBM TRIRIGA report manager. You can also define a data map as a dynamic outbound query.

Outbound queries from the report manager

When you define an outbound query with a query from the IBM TRIRIGA report manager, the mapping of data is defined in the query definition. The report labels that you specify for each field must represent the external table column name, XML node name, label value, or text field header column name.

Data maps as dynamic outbound queries

When you define an outbound query as a dynamic query, the mapping of data is defined in the data map. With this outbound usage of the data map, you do not map any inbound data from the external source to existing IBM TRIRIGA fields. Instead, you select which fields are exported from your IBM TRIRIGA database.

Database scheme
If you are using the database scheme and must export binary large objects (or BLOBs), you must use a dynamic query. You can select binary, note, or document manager content fields for export.

Default data
To specify default data for fields, you use a dynamic query. For example, if you are using the HTTP post scheme, you can use a dynamic query to pass a static parameter or value. For default data to work correctly with a dynamic query, you must first define and save your data map with the default data set, and then select the **Generate SQL for Table** action. This action sets the column to use the default data for new inserts.

Data maps

For inbound integrations, you must define a data map that maps the data from the external source to existing IBM TRIRIGA fields. However, for outbound integrations, you can also define a data map as a dynamic outbound query to select which fields are exported from your IBM TRIRIGA database.

Data maps for inbound integrations

When you define the data map, you must specify the module, business object, form, and fields. After you define the data map, you must save the map before you save the record, or your changes are lost.

The **Default Action** list is used if the mapped record cannot be found and must be created. If the mapped record exists, the default action is ignored, and the record is updated. The available actions that are listed in the **Default Action** list are the transitions from the null state that are available for the record.

The data map consists of the following hierarchical form elements:

Form
Displays the hierarchical representation of the form. Each form consists of tabs.

Tab
Displays the hierarchical representation of the tab. Each tab consists of sections.

Graphics section
Cannot be mapped.

Query section
Cannot be mapped.

Multi-tab section
Cannot be mapped.

Smart section field

Triggers a popup window to define how to identify the field. When you specify a smart section field, you use the popup window to select the form to use as a filter. Then, you select the field to use as a filter to retrieve the record ID. The record ID of the linked smart section must be identified by the IBM TRIRIGA Connector for Business Applications.

Locator field
Acts as a standard field, but the integration fails if the data cannot be located or is not unique.

Standard field
Is a text field that is treated as a string.

Date field
Is not displayed in the hierarchy. Instead, you use the field attributes table to define the correct IBM TRIRIGA date format for the incoming data.

Read-only field
Indicated by a strikethrough line and cannot be mapped, but it can be used as a key field to help identify the record.

Required field
Indicated by a red font and is required to create the record. The integration fails if this field is not mapped.

Each field in the data map possesses the following field attributes:

Base Parent
For hierarchical business objects such as geography, location, and organization, the path that identifies the root of the hierarchy. To ensure that the record is created under the correct root of the hierarchy, you must specify the path. If you do not specify the path, the record is created at the same level as the root and you cannot see or access the record from the form. An example for JBoss Application Server is `\Location`. An example for Oracle WebLogic Server is `\\Location`.

Type
The metadata definition type of the field. All fields are treated as strings. Examples include List, Number, and Text.

External

The name of the external field that you want to map to IBM TRIRIGA. Because the values can be used for database columns or formatted files, do not use spaces, special characters, or numbers. An example is User_Language.

When used in the response map of the HTTP post scheme, you can specify an XPath string or a JSONPath string.

isKey
Indicates whether the incoming record has a field that is defined as a key. If so, the value for the incoming row of data is used as a filter for the queried business object and exactly one record ID is returned. To update records in IBM TRIRIGA, you need the record ID to identify the record. If no record ID is found or multiple record IDs are returned, then the row creates a record with all values for the row.

isParent
Indicates whether the incoming record has a field that is defined as a parent. If so, the value for the incoming record is used as a filter in the query that is used to capture the record ID for the parent. By identifying the parent record of another record, the hierarchy can be created. To identify the child record of another record, you must include the record ID of the parent in the create-record or update-record request. If you define this value in the data map, this parent identification is handled automatically. If no parent is found or multiple parents are returned, then a warning is logged, and the record is created at the same level as the root.

Default
The constant value that is applied to all instances of the record. An example is US English. If you specify a default value, the **External** value is ignored, and the default column must exist only in this data map. The default value is only used at run time by the file scheme and HTTP post scheme. If you selected the database scheme, you must select the **Generate SQL for Table** action to set the default value in the staging table; the value is ignored at run time.

Data maps for inbound DataConnect integrations

When you define the data map, you must specify the module, business object, form, and staging table. After you define the data map, you must save the map before you save the record, or your changes are lost.

The data map for DataConnect integrations is similar to data maps that are used for other inbound integrations with the following exceptions or notes:

Modules
The list of modules is specific to those modules with business objects that have enabled the **Has Staging Table** property.

Business Objects
The list of business objects is specific to those business objects that have enabled the **Has Staging Table** property.

Forms
The value in the **Forms** field must be specified for the process to work correctly.

Staging Table
When a business object is selected, a query runs and displays the available database columns from the staging table that are associated with the business object. The **Staging Table** field is populated and is read-only.

Fields
The hierarchy shows the fields that are defined as staging table fields. The read-only fields, which are indicated by strikethrough lines, are used by the automatic process and cannot be mapped. The other fields are available.

Field attributes

When you select an available field, a new row is added to the field attributes table. The **External** attribute shows the name of the database column by default, but you can change it to match your file header. The **isKey**, **isParent**, and **Default** attributes are read-only and are not used.

Because you are mapping the import file to the staging database columns, the first row of the import file must have columns that match the **External** attribute values. The sequence of the columns does not matter, but the column names are case-sensitive. For example, if the **External** values in the data map are listed as **Field1**, **Field2**, **Field3**, then the actual external columns can be named **Field2**, **Field1**, **Field3**. If the names are identical, the fields are applied correctly.

Data maps for outbound integrations

For outbound integrations, you must define an outbound query to select which fields are exported from your IBM TRIRIGA database. You can define an outbound query from the IBM TRIRIGA report manager. You can also define a data map as a dynamic outbound query.

Response maps

For outbound HTTP post integrations, you must define a response map that maps the response parameters from the HTTP request to existing IBM TRIRIGA fields. Although the response map is organized like the data map, it applies to the HTTP post scheme only and it allows simple response values only.

Data validation

You can validate integration object records and specify the maximum number of validation errors that can occur before the importing process is stopped. You can also choose to validate these records with or without loading any records into TRIRIGA staging tables or creating any DataConnect jobs. Time formatting options are also provided on the data map tab for validation purposes. Data validation is only available when you import data with the DataConnect scheme.

Data validation options

When you specify whether to validate and identify invalid options, the following options are available:

- Specify whether to create a data file containing invalid records and attach it to the integration instance record.
- Specify whether to load staging tables with valid records or validate without loading staging tables. If you specify to load staging tables with valid records, the DataConnect Job that is created from processing the integration object loads data from the staging table into the environment. Data is loaded if there is at least one valid record in the data file.
- Specify the maximum number of validation errors that can occur before the importing process is stopped. Processing is stopped when the maximum or the end of the data file is reached. A value of 0 or a negative value indicates no maximum and the entire data file is processed regardless of the number of invalid records.
- Specify whether to validate records based on time formatting options.

Time formatting options

When you specify whether to validate records based on time formatting, the following options are available:

- Time formatting options are visible in the data map when validate is checked. Date, date and time, duration, and time values that are specified in the data file must correspond with the formats that are specified in the **Time Formatting Options** section. Otherwise, a validation error occurs.
- To ensure that the data and time format values are converted to the correct epoch values, you must choose the **Time Zone** region that corresponds with the date and time values in the data file. Daylight Saving Time is handled automatically based on the specified time zone region. The GMT offset values provided next to each time zone region might not be accurate for some time zone regions because of Daylight Saving Time. The values are provided to give order to the list.
- **Time Zone: Included with field values** takes time zone specifications from the date and time values in the data file. You can examine all valid time zones for including with imported date and time values by selecting **Valid Time Zones**. The time zone must be at the end of the field values in the data file, for example, "1/31/2015 10:00:00PM Pacific Standard Time". Note that using abbreviations, for example, "CST", might produce unexpected results since different time zone regions can have the same abbreviation, for example, "China Standard Time" and "Central Standard Time".
- **Lenient Format Parsing** parses date and time format ranges in a lenient manner when the data file is being validated. When lenient format parsing is checked, a date such as "February 942, 1996" is treated as being equivalent to the 941st day after February 1, 1996. When lenient format parsing is cleared, such dates cause a validation failure.

Data validation conditions

The following table provides details about the conditions that are validated on incoming data for integration object execution.

Table 1. Integration object validation

Condition	Description
Numeric field type	Checks to see whether the value contains numeric values.
Date, Date and Time, Duration, and Time data types	Checks to see whether the value corresponds with the format specified for the field type in the **Time Formatting Options** section.
Boolean data type	Checks to see whether the value contains Boolean values (‘true’ or ‘false’, case insensitive).
Classification data type	Checks to see whether the value exists in the classification in TRIRIGA.
List data type	Checks to see whether the value exists in the list in TRIRIGA.
Text field length	Checks to see whether the value exceeds the size that is specified for the field in the Data Modeler.
Locator fields	Checks all possible records for the locator and checks to see whether the value corresponds to one of the records.

Importing or exporting data with database tables

If your integration requires database tables to import or export your data, you use the TRIRIGA integration object to define the integration. The inbound database scheme maps the data from the external source to existing IBM TRIRIGA fields. The outbound database scheme defines which fields are exported from your IBM TRIRIGA database.

Importing data with the database scheme

When you select the inbound database scheme, you must define the data source and the data map. The data map is used to map the data from the external source to existing IBM TRIRIGA fields.

Procedure

1. Go to **Tools** > **System Setup** > **Integration** > **Integration Object**.
2. Add an integration object record.
3. Specify the integration name, database scheme, inbound direction, data source, and other integration details.
4. Test the database connection or generate the test data.
5. Define the data map by specifying the module, business object, form, and fields. Save the data map.
6. If you defined default data or must create generic SQL, generate the SQL for the database table.
7. Create and save the record.
8. Execute the integration.
9. Inspect any errors.

Exporting data with the database scheme

When you select the outbound database scheme, you must define the data source and the outbound query. The outbound query is used to define which fields are exported from your IBM TRIRIGA database.

About this task

You can define an outbound query from the IBM TRIRIGA report manager. You can also define a data map as a dynamic outbound query.

Procedure

1. Go to **Tools** > **System Setup** > **Integration** > **Integration Object**.
2. Add an integration object record.
3. Specify the integration name, database scheme, outbound direction, data source, and other integration details.
4. Test the database connection.
5. Define the outbound query.
6. If you selected a dynamic query, define the data map by specifying the module, business object, form, and fields. Save the data map.
7. If you defined default data or must create generic SQL, generate the SQL for the database table.
8. Create and save the record.
9. Execute the integration.
10. Inspect any errors.

Importing or exporting data with files

If your integration requires files to import or export your data, you use the TRIRIGA integration object to define the integration. The inbound file scheme maps the data from the external source to existing IBM TRIRIGA fields. The outbound file scheme defines which fields are exported from your IBM TRIRIGA database.

Importing data with the file scheme

When you select the inbound file scheme, you must define the import file and the data map. The data map is used to map the data from the external source to existing IBM TRIRIGA fields.

Procedure

1. Go to **Tools** > **System Setup** > **Integration** > **Integration Object**.
2. Add an integration object record.
3. Specify the integration name, file scheme, inbound direction, import file, and other integration details.
4. Test the file access.
5. Define the data map by specifying the module, business object, form, and fields. Save the data map.
6. Create and save the record.
7. Execute the integration.
8. Inspect any errors.

Exporting data with the file scheme

When you select the outbound file scheme, you must define the export file and the outbound query. The outbound query is used to define which fields are exported from your IBM TRIRIGA database.

About this task

You can define an outbound query from the IBM TRIRIGA report manager. You can also define a data map as a dynamic outbound query.

Procedure

1. Go to **Tools** > **System Setup** > **Integration** > **Integration Object**.
2. Add an integration object record.
3. Specify the integration name, file scheme, outbound direction, export file, and other integration details.
4. Test the file access.
5. Define the outbound query.
6. If you selected a dynamic query, define the data map by specifying the module, business object, form, and fields. Save the data map.
7. Set the **Localized?** check box to have query results localized.
   - If the file export type is **Flat**, all field values in the file will be localized.
   - If the file export type is **JSON**, the localized value will be contained in the JSON element where the record field name is appended with "_DISPLAY". For example, "name_DISPLAY":"Some Localized Value".
   - If the file export type is **XML**, the localized value will be contained in the record field child XML element named "displayValue" for each record element that is returned in the XML. For example, `<displayValue><![CDATA[Some Localized Value]]></displayValue>`.
8. Create and save the record.
9. Execute the integration.
10. Inspect any errors.

Importing data into DataConnect staging tables

If your integration requires an ETL tool, but you do not have the access or training to use one, you use the TRIRIGA integration object to define the integration. The inbound IBM TRIRIGA DataConnect scheme maps the data from the import file to the DataConnect staging tables.

Importing data with the DataConnect scheme

When you select the inbound IBM TRIRIGA DataConnect scheme, you must define the data source, the import file, the DataConnect job, and the data map.

About this task

The data source is used to access the DataConnect staging tables. The data map is used to map the data from the file to the DataConnect staging tables.

When you run the integration, the entries are inserted or updated in the DataConnect staging tables with an `upsert` (update or insert) action. You must select one or more fields as key fields. DataConnect uses the keys to determine whether it needs to insert or update the row.

Procedure

1. Go to **Tools** > **System Setup** > **Integration** > **Integration Object**.
2. Add an integration object record.
3. Specify the integration name, DataConnect scheme, data source, import file, and other integration details.
4. Test the database connection.
5. Test the file access.
6. Create and save the record.
7. Define the data map by specifying the module, business object, form, and fields. Save the data map.
8. Specify whether to validate and identify invalid records. If you specify not to validate, the DataConnect Job loads data from the staging table into the environment only if the entire data file is loaded successfully.
9. Save the record.
10. Execute the integration.
11. Inspect any errors.

Exporting data with the HTTP protocol

If your integration requires the hypertext transfer protocol (HTTP) to export your data, you use the TRIRIGA integration object to define the integration. The outbound HTTP post scheme maps the response parameters from the HTTP request to existing IBM TRIRIGA fields.

Exporting data with the HTTP post scheme

When you select the outbound HTTP post scheme, you must define the server to which the data is sent, the outbound query, and the response map.

About this task

The outbound query is used to define which fields are exported from your IBM TRIRIGA database. The response map is used to map the response parameters from the HTTP request to existing IBM TRIRIGA fields.

You can define an outbound query from the IBM TRIRIGA report manager. You can also define a data map as a dynamic outbound query.

Procedure

1. Go to **Tools** > **System Setup** > **Integration** > **Integration Object**.
2. Add an integration object record.
3. Specify the integration name, HTTP post scheme, external server, and other integration details.

4. Define the outbound query.
5. If you selected a dynamic query, define the data map by specifying the module, business object, form, and fields. Save the data map.
6. Define the response map by specifying the module, business object, form, and fields. Save the response map.
7. Create and save the record.
8. Execute the integration.
9. Inspect any errors.

Example: Exporting data in Esri geocoding integrations

You can select the outbound HTTP post scheme to export data from your IBM TRIRIGA database to an Esri server. Then, you can map the response parameters from the HTTP request to your location records to update their geocodes.

Background

Ichiro is an integration specialist at Company ABC. He develops IBM TRIRIGA integrations for the customers of Company ABC.

Ichiro is developing an outbound HTTP post integration with an external Esri geocoding service. For the Esri geocoding service to determine which addresses to geocode, the HTTP request must contain specific parameters in the query string. The response is in JSON format. The following HTTP URL provides Esri geocoding services through the REST API:

```
http://geocode.arcgis.com/arcgis/rest/services
  /World/GeocodeServer/findAddressCandidates
```

Ichiro remembers that since the latitude and longitude values for building records are being stored in the IBM TRIRIGA database, the HTTP URL must also contain specific parameters for Esri authentication. To refresh his memory, he visits Authenticate a request to the World Geocoding Service (https://developers.arcgis.com/rest/geocode/api-reference/geocoding-authenticate-a-request.htm). He uses his Client ID and Client Secret credentials to generate his unique Esri authentication token that is contained in the query string of the HTTP request. The following HTTP URL contains the required `forStorage` and `token` parameters and parameter values:

```
http://geocode.arcgis.com/arcgis/rest/services
  /World/GeocodeServer/findAddressCandidates?forStorage=true
  &token=Z43nqkyXY48Kx1iiPgCCZsh7Kf7ZxsKSH4xEO2Q8x34wiHU8wysAljsRzzYI51eg-
  zAr114CR0vgQ6fUt5a4eatD9FGejE4w8hTBY-f5vy5v1asxh_rXvLZsw6upiRPU
```

Step 1: Define the integration data scheme

Ichiro defines the integration object record with the following field details:

Name
Geocode Address

Scheme
Http Post

Direction
Outbound

Post Type
Query String

Response Type
JSON

Http URL

```
http://geocode.arcgis.com/arcgis/rest/services
  /World/GeocodeServer/findAddressCandidates?forStorage=true
  &token=Z43nqkyXY48Kx1iiPgCCZsh7Kf7ZxsKSH4xEO2Q8x34wiHU8wysAljsRzzYI51eg-
  zAr114CR0vgQ6fUt5a4eatD9FGejE4w8hTBY-f5vy5v1asxh_rXvLZsw6upiRPU
```

Query For Outbound section: Is Dynamic?
Yes

Step 2: Define the data map

Since Ichiro selected a dynamic query, he defines the data map by specifying the `Location` module, `triBuilding` business object, `triBuilding` form, and fields. The **External** field attribute values represent the parameter names that are added to the query string. Since the **Default** field attribute is not used, the field data is dynamically pulled from the IBM TRIRIGA location records. He defines the following fields with the following **External** attribute values:

triAddressTX
Address

triZipPostalTX
ZIP

triCityTX
City

triStateProvTX
State

For two more fields, Ichiro defines two more **External** values to specify the corresponding **Default** values that are required by the Esri geocoding service. His **External** values are `outSR` for the WKID and `f` for the format. His corresponding **Default** values are `102100` for the WKID and `json` for the format. To verify his `102100` value, he checks the WKID values for the geographic coordinate system (resources.arcgis.com/en/help/rest/apiref/gcs.html) and the projected coordinate system (resources.arcgis.com/en/help/rest/apiref/pcs.html). He defines the following fields with his **External** and **Default** attribute values:

triIdTX
`outSR` and `102100`

triDescriptionTX
`f` and `json`

Ichiro saves the data map.

Step 3: Define the response map

Next, Ichiro defines the response map to retrieve the response from the Esri geocoding service and map the retrieved latitude and longitude values to the IBM TRIRIGA location record. He defines the response map by specifying the `Location` module, `triBuilding` business object, `triBuilding` form, and fields. Since the response is in JSON format, the **External** attribute values are in JSONPath syntax. He defines the following fields with the following **External** attribute values:

triGisLongitudeNU
`candidates[0].location.x`

triGisLatitudeNU
`candidates[0].location.y`

Ichiro saves the response map and saves the integration object record.

Step 4: Execute the integration definition

When Ichiro selects the **Execute** form action to start the integration definition, the following process occurs:

- The data map values are used to construct a dynamic query. The query is called.
- Because the post type is a query string, each result from the query is transformed into the following URL structure:

```
http://geocode.arcgis.com/arcgis/rest/services
  /World/GeocodeServer/findAddressCandidates?forStorage=true
  &token=Z43nqkyXY48Kx1iiPgCCZsh7Kf7ZxsKSH4xEO2Q8x34wiHU8wysAljsRzzYI51eg-
  zAr114CR0vgQ6fUt5a4eatD9FGejE4w8hTBY-f5vy5v1asxh_rXvLZsw6upiRPU
  &Address=6720+Via+Austi+Pkwy&ZIP=89119&City=Las+Vegas
  &State=Nevada&outSR=102100&f=json
```

  The actual parameters depend on the actual IBM TRIRIGA data.
- The following JSON object shows the response:

```
{
    "spatialReference": {
        "wkid":102100,"latestWkid":3857
    },
    "candidates": [{
        "address":"6720 Via Austi Pky, Las Vegas, NV, 89119",
        "location": {
            "x":-12819684.945332458,
            "y":4309927.0270621451
        },
        "score":100,
        "attributes": {

        }
    },
    {
        "address":"6720 Via Austi Pky, Las Vegas, NV, 89119",
        "location": {
            "x":-12819744.862808136,
            "y":4309924.3334144857
        },
        "score":100,
        "attributes": {

        }
    },
    {
        "address":"6721 Via Austi Pky, Las Vegas, NV, 89119",
        "location": {
            "x":-12819808.664131654,
            "y":4309995.0691041043
        },
        "score":79,
        "attributes": {

        }
    },
    {
        "address":"Via Austi Pky, Las Vegas, NV, 89119",
        "location": {
            "x":-12819804.948472099,
            "y":4309871.0261052754
        },
        "score":100,
        "attributes": {

        }
    }]
}
```

- The response map values are used to extract `candidates[0].location.x` and `candidates[0].location.y` from the JSON object and map them to the `triGisLongitudeNU` and `triGisLatitudeNU` fields in the IBM TRIRIGA location record. The location record is updated with the new data.
- Since no action was specified for the exported data in the Query for Outbound section, no actions are triggered on the location data.
- The final count of the integration process is collected and an execute history record is created with the details about the integration run and any errors that occurred.

Integration execution

In addition to the **Execute** form action, you can also execute an integration object record or integration definition from a custom task or a URL. With a custom task, you can filter an outbound query with data from the currently running process. With a URL, you can pass parameters from an external application to trigger the **Execute** action or retrieve the results of an IBM TRIRIGA report-manager query in JSON format.

Execution from forms

After you create the integration object record, you can use the **Execute** form action to start the integration definition.

This action runs the `triIntegration - Execute` workflow. The integration object record becomes read only and its status changes to Processing. When the integration processing is completed with no errors, the integration object record changes back to the Ready state. If an error occurred during processing, the status changes to Failed, and the record does not change its state. You must then manually inspect the errors that occurred. However, you can override the inspection and change the record back to the Ready state by clicking the **Complete** action.

The Execute History section contains records with the details about each integration run. Each detail record contains the status, processing counts, and duration of the run. When an integration run has errors, it also includes a log of the errors and a query section that lists the individual records that failed. When the integration run is triggered from another workflow, the integration object record that contains the full overview of errors also contains a truncated message. The truncated message is in a hidden text field that can be displayed, for example, in an Attention message.

Outbound integrations that export results to an XML format or JSON format can process a maximum of 999 records at one time. If more than 999 records must be exported, you can select the **Exceed 999 Result Limit and Return Maximum?** option to export all of the results. Because more memory will be used as the result set gets larger, system performance should be considered when selecting this option with larger result sets. For other ways to exceed the 999 result limit, see the topic *"Example: Execution from custom tasks"* to create workflow logic that can run your integration until all of the records are processed.

Example: Execution from custom tasks

After you create the integration object record, you can use a custom task to filter an outbound query with data from the currently running process.

Background

Ichiro is an integration specialist at Company ABC. He develops IBM TRIRIGA integrations for the customers of Company ABC.

Ichiro updates a subset of records, but he wants an integration to send a record when a user clicks a particular action. So he uses a special Custom task object that uses workflow variables to set data, and that communicates with the `Integration` class loader record. By using this feature, Ichiro can build the subset of records in the workflow by using the common methods. Then, he can pass the subset by referring to the Custom task, and assign the integration object record that he wants to trigger to a workflow variable named `IntegrationObject`.

Ichiro's basic workflow contains the following workflow tasks:

- Start task
- Query task that is named `Query For Integration Object`
- Variable Definition task that is named `IntegrationObject`
- Variable Definition task that is named `IntegrationInstance`
- Query task that is named `Query for subset of people`

- Custom task that is named `Custom Task`
- End task

Step 1: Define the workflow

Ichiro selects the Start task, sets the workflow as asynchronous, and sets the workflow to run when a person record (`triPeople`) is saved (`triSave`).

Step 2: Define the first Query task

Ichiro inserts the first Query task and names it `Query For Integration Object`. In this task, he queries (`Manager Default - Integration Objects`) for the integration object record that he wants to trigger. Then, he filters the results of this query for a specific name. Ichiro must have only one resulting integration object record.

Step 3: Define the first Variable Definition task

Ichiro inserts the first Variable Definition task and names it `IntegrationObject`. In this task, he defines the integration object record (`triIntegration`) as a variable. The result of the query from the previous task is assigned to this variable.

Step 4: Define the second Variable Definition task

Ichiro inserts the second Variable Definition task and names it `IntegrationInstance`. In this task, he defines the integration instance object (`triIntegrationInstance`) as a variable. This return variable is used for the return value from the Custom task at the end of the workflow.

Step 5: Define the second Query task

Ichiro inserts the second Query task and names it `Query for subset of people`. In this task, he obtains the filtered set of people records that are processed for the integration. He queries (`triEmployee – Find`) for the people records. Then, he filters the results for records where the person's given name (`triFirstNameTX`) contains `Ichiro`.

Ichiro verifies that the people business object (`triPeople`) contains the record ID (`triRecordIdSY`) field. For any business object that is used in the Query task that is passed to the integration object record, the record ID (`triRecordIdSY`) field must be included. When an integration is running, the integration object record uses the record ID to retrieve the remaining fields in the business object that is defined in the data map.

Step 6: Define the Custom task

Ichiro inserts the special Custom task that uses workflow variables to set data, and that communicates with the `Integration` class loader record. In this task, he passes the results of the `Query for subset of people` task as the set of records. Finally, Ichiro adds the return variable that is needed to obtain the integration instance record. The name of the return value must be `IntegrationInstance`.

This Custom task is also defined with the following values and behavior:

- The **Class Name** field for the Custom task is set to `Integration:com.tririga.custom.integration.Parameter`. This class name must be used when this method is implemented. The `Parameter` class inside of the `Integration` class loader path implements the `CustomParamBusinessConnectTask` Java Interface class with the following signature:

```
public CustomParamTaskResult execute(TririgaWS client,
  Map params, long userId, Record[] records)
```

- The `params` argument represents the assigned `IntegrationObject` variable so that the process has the instructions that it needs to continue the integration. The `records` argument represents the

Records section in the Custom task where the records to use for the workflow process are assigned. Instead of triggering the event on one object, two sets of objects are being passed to an event.

- The `Parameter` class looks for the integration object record from the `IntegrationObject` variable and processes the results in the `records` argument that are passed. This processing includes running the query from the Query for Outbound section of the integration object record and filtering by the record IDs of the records that are passed. As a result, data is being exported with the common utilities. But the export is no longer bound by all-or-nothing queries, or by triggering the event directly from the **Execute** form action.
- Outbound integrations that are executed from the `Parameter` class can process a maximum of 1000 records at one time, or 999 records at one time for XML or JSON format. If more than 1000 records must be exported, you can select the **Exceed Custom Task Parameter 1000 Result Limit and Return Maximum?** option. This option will also exceed the 999 result limit for XML and JSON formats. Because more memory will be used as the result set gets larger, system performance should be considered when selecting this option with larger result sets. For another way to exceed the 1000 result limit for the `Parameter` class, see the following section *"Batch the query results"* to create workflow logic that can run your integration in batches of 999 results until all of the records are processed.

Batch the query results

If there are more than 999 records to be exported, create your workflow logic to batch the query results into batches of 999, by performing the following steps:

- 1. Create a variable that will be used to manage the batching of the query results. For example, *Var Batched Query Results*. The **Module** and **Object** field values will be the module and query from Step 5.
- 2. Create an Iterator task that will iterate the results of the query from Step 5. Within the Iterator task, add an Assign Variable task to append each result to the *Var Batched Query Results* variable. Make sure that the **Append Results** check box is selected.
- 3. Create a Switch task that will be used to control each batched result. The expression will check for when the *Var Batched Query Results* variable result count is equal to 999.
- 4. Within the Switch task, add the Custom task that calls the integration object record. Make sure that the Records section of this Custom task specifies the *Var Batched Query Results* variable. Also make sure that the integration object record (called by this Custom task) does not have the **Overwrite** check box selected. The integration object record will also be designed with the **Is Dynamic** check box selected within the Query for Outbound section. The fields intended for export will be specified in the **Data Map** tab.
- 5. Still within the Switch task, add an Assign Variable after the Custom task. This will be used to clear the results of the *Var Batched Query Results* variable after the Custom task is run. This will also prepare the *Var Batched Query Results* variable for the next batch of 999 results.
- 6. After the Iterator task, add another Switch task. This will be responsible for processing the remaining results. The expression will check for when the *Var Batched Query Results* variable result count is greater than 0.
- 7. Within this Switch task, call the same Custom task that was added above.

Note that this process will not create one file. It will create one or more batched files of 999 records each, plus a final file with the remainder.

Execution from URLs

After you create the integration object record, you can use a URL to pass parameters from an external application to trigger the **Execute** action. You can also use a URL to retrieve the results of an IBM TRIRIGA report-manager query in JSON format.

IBM TRIRIGA Application Platform version 3.3 or newer is required to trigger an event externally with a URL that contains the authentication credentials. You can specify authentication in one of the following ways:

- Add the **UserName** and **Password** values to the HTTP request header in the standard security protocol of basic authorization.
- Add the **USERNAME Parameter** and **PASSWORD Parameter** values and their corresponding **UserName** and **Password** values to the HTTP request header.
- Add the **USERNAME Parameter** and **PASSWORD Parameter** values and their corresponding **UserName** and **Password** values as POST parameters or as a URL query string of the HTTP request.

Trigger the Execute action

To trigger the **Execute** action on the integration object record externally with a URL, you append a query string to the URL. The query string contains the credentials and sets the additional `ioName` parameter to the name of the integration object record. For example, to trigger a `Geocode Address` integration object record, you call the following URL:

```
http://localhost:8001/html/en/default/rest
  /Integration?USERNAME=username&PASSWORD=password&ioName=Geocode+Address
```

If the URL triggers the **Execute** action on the integration object record as expected, a `Successful` message is returned.

Triggering the **Execute** action on an integration object record is an asynchronous event, so it is not possible to get the results of the integration process synchronously. If you need immediate feedback of the results, you can tie a notification event to the completion of the integration object record. Then, the IBM TRIRIGA application can notify you when the integration process is complete. Another option might be to poll IBM TRIRIGA for that information.

Retrieve the query results

Any query that is defined in the IBM TRIRIGA report manager can be executed externally with a URL. To specify the query, you use the following parameters:

`action`
Required parameter. Must be set to `query`.

`module`
Required parameter if no continue token is used. The module for the query, for example, `triPeople`.

`bo`
Optional parameter. The business object for the query, for example, `triPeople`. If there is more than one business object, omit this parameter.

`query`
Required parameter if no continue token is used. The name of the query, for example, `triEmployee - Find`.

`f`
Optional parameter. The format of the returned results, which includes the following options:

`json`
Default format. Minimized JSON string.

`json-loc`
Minimized JSON string, containing localized values. Localized values will appear in the string name value pairs where the name field is appended with `_DISPLAY`, for each record in the JSON results. For example, `"name_DISPLAY":"Some Localized Value"`.

`pjson`
JSON string in print format for easier readability.

`pjson-loc`
JSON string in print format for easier readability, containing localized values. Localized values will appear in the string name value pairs where the name field is appended with `_DISPLAY`, for each record in the JSON results. For example, `"name_DISPLAY":"Some Localized Value"`.

xml
Minimized XML string.

xml-loc
Minimized XML string, containing localized values. Localized values will appear in the `displayValue` child element of the column element, for each record in the XML results. For example, `<displayValue><![CDATA[Some Localized Value]]></displayValue>`.

pxml
XML string in print format for easier readability.

pxml-loc
XML string in print format for easier readability, containing localized values. Localized values will appear in the `displayValue` child element of the column element, for each record in the XML results. For example, `<displayValue><![CDATA[Some Localized Value]]></displayValue>`.

tab
Tab-delimited text.

ct
Required parameter if the next batch of records is being requested. A continue token is returned when the number of query results exceeds the default 1000 results. When you attempt to retrieve the next batch of records, you must pass this parameter with the token that is specified in the previous response.

For example, to specify a query for all employees, you call the following URL:

```
http://localhost:8001/https://www.ibm.com/support/knowledgecenter/SSHEB3_3.7/com.ibm.tap.doc/
html/en/default/rest
  /Integration?USERNAME=username&PASSWORD=password&action=query
  &module=triPeople&bo=triPeople&query=triEmployee+-+Find&f=pjson
```

Execution of DataConnect processes

When you select the inbound IBM TRIRIGA DataConnect scheme, the integration definition can run two types of DataConnect jobs. The standard type creates one job for the selected business object. The multiple type loads multiple staging tables and runs them as one job. The multiple type requires a dependent integration definition that is available in the workflow.

Standard type
When the integration definition runs, this type of DataConnect job follows the following process flow.

1. Creates a job number from the maximum `DC_JOB` number.
2. Inserts a new entry into the `DC_JOB` table with the new job number, business object, and status of 1 (New).
3. Loads the staging table with the job number, form name, state of 2 (Ready), and action of 4 (Upsert).
4. Updates the `DC_JOB` table with the status of 2 (Ready).

Multiple type
When the integration definition runs, this type of DataConnect job follows the following process flow.

1. Creates a job number from the maximum `DC_JOB` number.
2. Inserts a new entry into the `DC_JOB` table with the new job number, business object, and status of 1 (New).
3. Loads the staging table with the job number, form name, state of 2 (Ready), and action of 4 (Upsert).
4. Queries for the dependent integration definition (**IntegrationObject Dependent**) and repeats steps 1 - 4 until an integration definition no longer has a dependent integration definition.
5. Updates the `DC_JOB` table with the status of 2 (Ready).

Error handling

If failures occur during an integration, the affected records are not saved. The integration instance summary displays the errors that occurred, and each record that failed is represented by an instance failure record. An instance failure record contains an instance record representation that you can manually edit and resubmit.

The **Resubmit Record** field in an instance failure (`triIntegrationInstanceFailure`) record is a note field that contains the key-value pairs that represent the record that you are trying to create or update. You can manually edit the data in the **Resubmit Record** field and then click the **ReSubmit** form action to resubmit the record.

When a resubmitted record is processed successfully, the following events occur:

- The representation of the failed record changes from "Failure" to "Successful".
- The text entries in the **Error Message** field and the **Resubmit Record** field are cleared.
- The integration instance (`triIntegrationInstance`) record counts are updated to reflect the correct numbers. Specifically, the number in the **Records Successful** field is increased by one and the number in the **Records Failed** field is decreased by one.

When the **Record Failed** count is equal to zero, you can complete the integration object record with the **Complete** action.

Integration elements

In addition to defining and executing the integration definition, the elements and processes that are common to integrations include outbound formats, object upgrades, and object migrations. Other common elements include standard workflows, standard queries, and standard lists.

Outbound formats

For outbound file integrations, you can select `.json`, `.xml`, or `.xslt` file formats. For outbound HTTP post integrations, you can select JSON, XML, or XSLT post formats and response formats.

File export types

The file export formats include flat, `.json`, `.xml`, and `.xslt`. If you select the flat option for standard delimited files, the **File Header** field becomes available.

HTTP post types

The HTTP post formats include parameter, query string, JSON, XML, and XSLT. The results of the outbound query are converted into the selected format.

Parameter
Each row is converted into parameter name-value pairs. Each record is submitted separately. No batch submission is available.

Parameters can also be specified in the **Parameters** field, which is only visible for this HTTP post type. This field accepts name-value pairs in the `name:value` format, where the name and value are separated by a colon. This field accepts multiple name-value pairs with one entry per line.

Query string
Each row is converted into parameter name-value pairs, which are appended to the URL as a query string, and submitted. Although the data is appended to the URL, instead of being included in the body of the request, the HTTP request method is POST, not GET.

JSON

You can select JSON as the post type for the HTTP post scheme or as the export type for the file scheme.

If you want to send a separate request for each record that is returned by the outbound query, specify the **Response Type** value. The simple response parameters must match up with the fields that are defined in the response map. To send all of the returned records as a single batch submission, do not specify a response type.

The JSON structure has two objects: data and header. The data object contains an array of objects with the labels and values from the outbound query columns, and if specified, also includes the business object ID and record ID. The following example shows the default JSON structure:

```
{
    "data": [
        {
            “[column_label]”:”[column_value]”,
            "boId": 10002100,
            "recId": 2999294
        },
    ...
    ],
    "header": [
        "[column_label]",
    ...
    ]
}
```

XML and XSLT

You can select XML as the post type for the HTTP post scheme or as the export type for the file scheme.

If you want to send a separate request for each record that is returned by the outbound query, specify the **Response Type** value. The simple response parameters must match up with the fields that are defined in the response map. To send all of the returned records as a single batch submission, do not specify a response type.

Default XML structure

The XML structure has three nodes for each outbound query column: field, label, and value. The following example shows the default XML structure:

```
<query>
   <continueToken/>
   <results total="13">
      <result recordId="11430080" associatedRecordId="null"
boId="106402">
         <columns>
            <column>
               <field>triIdTX</field>
               <label>HR_ID</label>
               <value>1000000</value>
            </column>
            ...
         </columns>
      </result>
      ...
   </results>
</query>
```

For date values, the outbound query results include the raw values that are stored in the database and the formatted display values that are shown to the user. The `displayValue` node is added to the query results. The following example shows the default XML structure:

```
<column>
    <field>Date</field>
    <label><![CDATA[Date]]></label>
    <value><![CDATA[1359964800000]]></value>
    <displayValue><![CDATA[02/04/2014]]></displayValue>
</column>
<column>
    <field>DateTime</field>
    <label><![CDATA[Date_Time]]></label>
    <value><![CDATA[1360009800000]]></value>
```

```
    <displayValue><![CDATA[02/04/2014 12:30:00]]></displayValue>
</column>
```

Labels as nodes

If you select the **Use Query Label As Element** option, the default XML structure changes. The label and value nodes are merged by using the label name as the value node. The following example shows the XML structure:

```
<query>
   <continueToken/>
   <results total="13">
      <result recordId="11430080" associatedRecordId="null"
boId="106402">
         <columns>
            <column>
               <field>triIdTX</field>
               <HR_ID>1000000</HR_ID>
            </column>
            ...
         </columns>
      </result>
      ...
   </results>
</query>
```

For date values, the merged *name*_display node is added to the query results. The following example shows the XML structure:

```
<column>
    <field>Date</field>
    <Date><![CDATA[1359964800000]]></Date>
    <Date_display><![CDATA[02/04/2014]]></Date_display>
</column>
<column>
    <field>DateTime</field>
    <Date_Time><![CDATA[1360009800000]]></Date_Time>
    <Date_Time_display><![CDATA[02/04/2014 12:30:00]]>
</Date_Time_display>
</column>
```

XSLT transformations

If you are applying an XSLT transformation to the XML structure, you can use the **XSLT** binary field to store the XSLT stylesheet. This field is not required. You can also transform the XML structure with or without the **Use Query Label As Element** option.

HTTP response types

The HTTP response formats include flat, JSON, and XML. The simple response parameters must match up with the fields that are defined in the response map. If you select the **Send As Batch** option, do not specify a **Response Type** value. After the integration process is finished, the records that are affected by the response are listed in the **Associations** tab of the integration definition with the association string of Sourced.

Flat

Retrieves the HTTP response in string format.

For example, if the response is "Success" or "Failure", you can map that string to a field that can be used as a decision point in a post-processing workflow.

JSON

Retrieves the HTTP response in JSON format.

In the response map, you define the **External** attribute with the JSONPath string. You must use simple response definitions only. If the JSON element that you want to map is in an array, you must

specify the index number of that array. For example, `candidates[0].location.x` returns `-12819744.7565` in the following JSON structure:

```
{
   "spatialReference" : {
     "wkid" : 102100
   },
   "candidates" : [
     {
       "address" : "6720 Via Austi Pky, Las Vegas, NV, 89119",
       "location" : {
         "x" : -12819744.7565,
         "y" : 4309920.3012000024
       },
       "score" : 100,
       "attributes" : {

       }
     },
     {
       "address" : "6721 Via Austi Pky, Las Vegas, NV, 89119",
       "location" : {
         "x" : -12819812.309700001,
         "y" : 4309994.186999999
       },
       "score" : 79,
       "attributes" : {

       }
     }
   ]
}
```

XML

Retrieves the HTTP response in XML format.

In the response map, you define the **External** attribute with the XPath string. You must use simple response definitions only. If the XML element that you want to map is in an array, you must specify the index number of that array. For example, `//root/candidate/location[2]/x` returns 5 in the following XML structure:

```
<root>
  <candidate>
    <location>
      <x>1</x>
      <y>2</y>
    </location>
    <location>
      <x>5</x>
      <y>6</y>
    </location>
  </candidate>
</root>
```

The XML format can also retrieve and update multiple records in the response. In the response map, where you define the **External** attribute with the XPath string, you must represent the repeatable XML element with the token `[i]`. In the following example, to map the XPath to the `recordId` attribute in the XML structure, you set the XPath to `//result[i]/@recordId`. To map the XPath to the `Name` node, you set the XPath to `//result[i]/columns/column/Name`. You can also set the **isKey** (key) attribute in the map to update the record.

```
<?xml version="1.0" encoding="UTF-8"?>
<query>
  <continueToken/>
  <results total="6">
    <result recordId="11464082" associatedRecordId="null" boId="10025526">
      <columns>
        <column>
          <field>triNameTX</field>
          <Name><![CDATA[Default Map]]></Name>
          <Name_display><![CDATA[Default Map]]></Name_display>
        </column>
        <column>
          <field>triIdTX</field>
```

```
          <ID><![CDATA[001]]></ID>
          <ID_display><![CDATA[001]]></ID_display>
        </column>
      </columns>
    </result>
    <result recordId="11464531" associatedRecordId="null" boId="10025526">
      <columns>
        <column>
          <field>triNameTX</field>
          <Name><![CDATA[SecondMap]]></Name>
          <Name_display><![CDATA[SecondMap]]></Name_display>
        </column>
        <column>
          <field>triIdTX</field>
          <ID><![CDATA[002]]></ID>
          <ID_display><![CDATA[002]]></ID_display>
        </column>
      </columns>
    </result>
...
```

Object upgrades

Whenever a new IBM TRIRIGA Application Platform installer is available, you can specify whether the installer upgrades the TRIRIGA integration object during the platform upgrade.

If you do not want the installer to upgrade the TRIRIGA integration object, then before you run the installer, you create an integration object record that is named `IGNORE_UPGRADE`. Do not run this record. The record needs only to be present in the existing platform installation. Then, when the installer runs, it does not upgrade the TRIRIGA integration object even if the installer build version is newer than the currently installed platform version.

If you do not create this `IGNORE_UPGRADE` record, then the installer upgrades the TRIRIGA integration object if the installer build version is newer than the currently installed platform version.

Object migrations

After you run an object migration, but before you use an integration object record in the new environment, you must open any integration object record and run the **ReMap** action.

The **ReMap** action updates all of the IDs that are saved in your data maps and response maps. These maps contain IDs for modules, business objects, forms, and smart record data. You must run the **ReMap** action to be sure that the IDs are updated for the new environment.

Object elements

The `triIntegration` module contains several business objects such as the TRIRIGA integration object. In addition, the integration object record requires the use of several class loader elements.

The `triIntegration` module includes the following business objects:

`triIntegration`
The TRIRIGA integration object is the primary business object that defines the integration between IBM TRIRIGA and external systems. You create integration definitions or integration object records from this TRIRIGA integration object.

`triDataSource`
The data source business object contains the property settings that define the connection to a database. This object is referenced as a single-record smart section in the integration object record. This referencing allows the reuse of data source definitions across multiple integration definitions.

`triIntegrationInstance`
The integration instance business object contains the details about a specific instance of an integration run. This object contains the counts, the time it took to run, a collection of error messages if any, and a query section with details about each record that failed. This object is associated to records that are affected by the response. This object is referenced in the Execute History section of the integration object record.

triIntegrationInstanceFailure
The integration instance failure business object contains the details about a single record that failed for an inbound integration. This object contains the error message for the failure and a simplified representation of the submitted record that can be manually edited and resubmitted.

The TRIRIGA class loader record that is named `Integration` is required by the integration object record. This `Integration` class loader record includes the following elements in its Resource Files query section:

jtds-1.2.5.jar

The Java Device Test Suite (JDTS) `.jar` file is the database driver for Microsoft SQL Server.

Although IBM TRIRIGA is delivered with the Oracle Java Database Connectivity (JDBC) driver for Oracle Database, IBM TRIRIGA does not include a database driver for Microsoft SQL Server. If you are connecting to Microsoft SQL Server, you must add a Microsoft SQL Server driver such as the `jtds-1.2.5.jar` to the Resource Files query section of the `Integration` class loader record.

TRIRIGAIntegration.jar
The TRIRIGA integration `.jar` file contains all of the Java classes that are used by the TRIRIGA integration object components. The file includes workflow-related Custom task entry points and a communication layer that is used by the `TRIRIGAIntegration_Assets.zip` file elements.

TRIRIGAIntegration_Assets.zip
The TRIRIGA integration assets `.zip` file contains all of the `.html`, `.css`, JavaScript, and image files that are used by the data map for inbound integrations. The JavaScript communicates with the Java API of IBM TRIRIGA Connector for Business Applications by using URI-based servlet proxy handles. The server-side elements are contained within the `TRIRIGAIntegration.jar` file.

Object versions

If you want to determine which version of the TRIRIGA integration object is installed in your IBM TRIRIGA environment, you can go to the following web address: `http://`*`[localhost:8001/context]`*`/html/en/default/rest/Integration`. The result displays the build date, number, and version number of the integration object that you are currently running.

Standard workflows

The TRIRIGA integration object is delivered with several IBM TRIRIGA workflows.

The following workflows are triggered from the integration object record or its supporting records.

triIntegration - Execute

This asynchronous workflow is triggered when the user selects the **Execute** action. This workflow is the primary workflow that supports the integration between IBM TRIRIGA and external systems. This workflow controls the record status that is displayed on the form. The `Trigger Integration` task is a Custom task that calls `Integration:com.tririga.custom.integration.Integration` and is the primary entry point for all integration object records. The information about the processes to run during the integration is defined in the integration object record and is passed to the Custom task through the record ID.

The standard workflow contains the following workflow tasks:

- Start task
- Modify Records task that is named `Mark Status as Processing`
- Custom task that is named `Trigger Integration`
- Switch task
  - If the Switch condition is true, run the following tasks:
    - Trigger Action task
    - Modify Records task that is named `Mark Status as Ready`

- If the Switch condition is false, run the following tasks:
  - Modify Records task that is named `Mark Status as Failed`
- End task

triIntegration - Generate SQL for Table
This synchronous workflow is triggered from the Database section action of an integration object record to generate the SQL for use with your staging tables.

triIntegration - Generate Test Data
This synchronous workflow is triggered from the Database section action of an integration object record to generate randomized test data in your staging tables. This test data can be used for functional testing or load testing.

triIntegration - HideShow Data Sections
This synchronous workflow is triggered from the initial loading of a new integration object record, and from various form elements such as a field-action `OnChange` workflow. This workflow shows and hides the form elements as needed to suit the integration that you are defining.

triIntegration - PreLoad
This synchronous workflow is called when a new integration object record is opened. In turn, this workflow calls the `triIntegration – ResetMetaData` and `triIntegration – HideShow Data Sections` workflows.

triIntegration - Reset MetaData
This synchronous workflow resets the form elements that are cleared. This workflow is called from the `triIntegration – PreLoad` workflow.

triIntegration – Resubmit
This asynchronous workflow is triggered when you resubmit a failed record from the instance failure (`triIntegrationInstanceFailure`) record.

triIntegration - Test DB Connection
This synchronous workflow is triggered from the Database section action of an integration object record to call `select count(1) from [table_name]` on the defined database. This call validates that the integration definition can communicate and run SQL commands on that database.

triIntegration - Test File Access
This synchronous workflow is triggered from the File section action of an integration object record to validate the create, retrieve, update, and delete permissions to the specified file location.

triDataSource - Reset Metadata
This asynchronous workflow resets the data source (`triDataSource`) record.

triDataSource - Test DB Connection
This synchronous workflow is triggered from the data source (`triDataSource`) record to call `select count(1) from [table_name]` on the defined database. This call validates that the data source definition can communicate and run SQL commands on that database.

Standard queries

The TRIRIGA integration object is delivered with several IBM TRIRIGA queries.

The following application areas and corresponding queries are included in the `triIntegration` module.

Datasource Name field
`triDatasource - getIntegrationObject datasource`

Execute History section
`triIntegration - get Instances`

Failures section
`triIntegrationFailures - Get all failures`

IntegrationObject Dependent field
`Manager Default - Integration Objects`

My Reports portal
`triIntegration - getIntegrationObject`

Standard lists

The TRIRIGA integration object is delivered with several IBM TRIRIGA lists.

The following fields and corresponding lists are included in the `triIntegration` module.

DataConnect Type
`triDataConnectTypeLI`

DB Driver Name
`triDriverNameLI`

Delimiter
`triFileDelimiterLI`

Direction
`triIntegrationDirectionLI`

Export Type
`triFileExportTypeLI`

Integration Type
`triIntegrationTypeLI`

Post Type
`triHttpPostTypeLI`

Chapter 3. Integrating data with the TRIRIGA Connector for Esri Geographic Information System (GIS)

You can retrieve data from your IBM TRIRIGA database and data from an Esri ArcGIS server, and display that data in GIS maps. The GIS maps can be in portal sections and form tabs.

Overview of GIS maps

When you add maps to your IBM TRIRIGA portals and forms, users see data in graphic form, with features you customize to enhance their understanding of the data. To show maps in the IBM TRIRIGA application, you must create a GIS map record for each map, and you must specify the location of your GIS map server.

A geographic information system (GIS) is a system for capturing, storing, analyzing, and managing data and associated attributes that are spatially referenced to Earth. In the strictest sense, it is an information system capable of integrating, storing, editing, analyzing, sharing, and showing geographically referenced information. With GIS, users can create interactive queries, analyze spatial information, edit data, integrate maps, and present the results of these tasks.

Geographic information system technology is used for resource management, asset management, environmental impact assessment, urban planning, sales, marketing, logistics, and many other activities. For example, GIS makes it possible for facility managers to easily assess impacts on facility assets in the event of a natural disaster. Or, GIS can be used to find a site for a new business to take advantage of a previously underserved market.

Esri data services and IBM TRIRIGA tools

To display GIS maps in IBM TRIRIGA forms, you need both Esri data services and IBM TRIRIGA tools.

The following list shows the services and tools that enable the display of GIS maps in IBM TRIRIGA forms.

Esri data services

- The geographical and geospatial data (Data Services). The IBM TRIRIGA system obtains this data through REST API services on Esri servers.
- The actual map view. The ArcGIS server, whether the services offered online or the services hosted on a proprietary server, renders the maps and handles your geoprocessing.
- Geoprocessing for drive time or distance radii. Geoprocessing is provided by the ArcGIS server.
- Geocoding for gathering the latitude and longitude coordinates of addressable objects or identifiers that represent the features. Geocoding is provided by the ArcGIS server.
- The Esri JavaScript API. It renders the viewer to provide data from the server and to provide basic interaction with the map data. The standard configuration is defined in the IBM TRIRIGA GIS Map object.

IBM TRIRIGA tools

- An initial set of building data that is used to query the Esri ArcGIS server.
- A ClassLoader object that contains the logic to render the Esri JavaScript viewer.
- Tools to configure the basemaps, layers, spatial references, widgets, and queries that are used in the rendered GIS Map areas within IBM TRIRIGA.

Esri offers a number of data services, including the Spatial Query data service. These services are used by IBM TRIRIGA to determine the geographic areas that the user sees within the current map view.

A GIS map can be displayed in an IBM TRIRIGA portal section or form tab. The same full-featured function is available each time a map is displayed.

The versions of the IBM TRIRIGA Application Platform and the IBM TRIRIGA applications that support the GIS features are defined in the "Compatibility Matrix for IBM TRIRIGA Products" IBM Support page.

Esri integration points

To display maps, IBM TRIRIGA communicates with a mapping service that is provided by the Esri ArcGIS server.

IBM TRIRIGA communicates with the Esri ArcGIS server through Java™, JavaScript, and HTML. The Esri JavaScript API makes the service calls to the Esri ArcGIS server and to IBM TRIRIGA with its native HTTP service protocol. Actions that are provided on IBM TRIRIGA records, such as the **Geocode Address** action, run an integration object record via a workflow to make a call to the service.

The following fields in each of the as-delivered location business objects support this integration object record:

GIS Latitude (`triGisLatitudeNU`)
Contains the latitude of the geocode point.

GIS Longitude (`triGisLongitudeNU`)
Contains the longitude of the geocode point.

Esri authentication credentials

For the **Geocode Address** action in the location records and forms to correctly retrieve and store these latitude and longitude values, Esri OAuth credentials are required. These credentials can also be used to access and connect to secured online basemaps and layers that are hosted by Esri services at GIS runtime. Your Esri OAuth credentials must be provided by one of the following methods.

Esri authentication record

If you use this authentication method, go to **Tools** > **System Setup** > **GIS** > **GIS Esri Authentication**. Your Esri OAuth credentials must be provided in the Esri authentication record:

client_id
Contains the actual value of your Client ID.

client_secret
Contains the actual value of your Client Secret.

To verify your values, select **Validate Credentials**.

Integration object record

If you use this integration object method, go to **Tools** > **System Setup** > **Integration** > **Integration Object**. Your Esri OAuth credentials must be provided in the Http Post section of the `Geocode Address - Esri - Authentication` integration object record:

UserName Parameter (`triPostUserNameParameterTX`)
Contains the parameter value of `client_id`.

UserName (`triPostUserNameTX`)
Contains the actual value of your Client ID.

Password Parameter (`triPostPasswordParameterTX`)
Contains the parameter value of `client_secret`.

Password (`triPostPasswordPA`)
Contains the actual value of your Client Secret.

Your Client ID and Client Secret credentials are required to generate the Esri authentication token that is contained in the query string of the HTTP request. For more information, see Accessing ArcGIS Online Services (https://developers.arcgis.com/authentication/accessing-arcgis-online-services/) and

Authenticate a request to the World Geocoding Service (https://developers.arcgis.com/rest/geocode/api-reference/geocoding-authenticate-a-request.htm).

You define the URLs and ports that are used for creating the GIS map. You define basemaps, layers, and widget services such as the geometry service. The only exception is the Esri JavaScript API sourced from the Esri CDN. The offline API is self-contained and can be used behind your firewall. For more information about the services that are used with that API, see the website by Esri (www.esri.com).

Maps in GIS portal sections

A GIS portal section points to an in-application web page that contains a GIS map. The GIS map is associated to a GIS map record that defines the data to render.

The GIS portal sections with the in-application web pages include the following portal sections:

- `triURL - GIS - Environmental Manager/Planner`, or
- `triURL - GIS - Environmental Manager/Planner (US Govt)`

When a user signs in to the application, the URL is called to load the web page that contains the map. Based on the map that is defined in the URL parameter, a list of executable reports is returned. The first report in that list runs, and the results are returned to the map.

Reports can be metric queries or standard queries. The reports define what a user sees in the bubble markers for the locations or features. The queries also populate the table of data in the viewer. There is a one-to-one correlation between the bubble markers on the map and the data in the table. The table contains the same fields that the bubble marker contains because the source data for both is the same report.

With map widgets, the user can pan or zoom or find locations. The user can also create and edit features on the map and assign those features to an IBM TRIRIGA object.

The pinpoints on the map show the locations that are returned from the query. Each location has a bubble marker (hover text) that displays information that is related to the location. If the query is a metric query, the bubble markers display the metric results. The colors of the location pinpoint icons match the thresholds that are represented by the metric results. If the query is a standard query, the location pinpoint icons are colored blue. The default colors are blue, red, green, and yellow. You can use your own icons to represent the location pinpoints on the map.

The query that is associated to the map in a portal section does not affect or react to any other data on the portal. GIS is a stand-alone application inside of a portal section.

A GIS section includes a **Save Preferences** button. This feature saves the current map extent and view. The next time the user signs in, the saved settings override the default in the URL parameter of the portal section URL. If the user did not save preferences, the section shows the default view as defined in the URL parameter string. The user preferences are stored per user, per map. When the user changes to a different map, the preferences for the original map are not applied to the second map. Instead, the second map is displayed with the default settings for that map, unless the user previously saved preferences for the second map.

Maps in GIS form tabs

A GIS tab contains a map that pinpoints the location of the record.

When a user has appropriate licenses, the as-delivered location, geography, and real estate transaction plan forms include a GIS tab. The map extent of the map in the GIS tab is the localized area. A default query renders the map. The query is specified in the URL as a parameter string that determines which queries to run. The URL for the GIS tab is defined in the Form Builder.

Overview of GIS map elements

When you define a new GIS map definition, you must configure the various map elements. Map elements include the extents, queries, basemaps, layers, icons, and widgets.

GIS map records

To define a new GIS map definition, you must create a GIS map record from the GIS map business object. Existing GIS map records are listed in **Tools** > **System Setup** > **GIS** > **GIS Map**.

Extents

The initial extents refer to the spatial reference and the boundaries of the map when it first opens in a portal section or form tab.

If the well-known ID (WKID) is not appropriate for the basemaps and layers, your map section is blank and does not contain a map. Two examples of WKID values are 4326 and 102100.

For more information, see the WKID values for the geographic coordinate system and the projected coordinate system.

Queries

The queries define the IBM TRIRIGA building location data that is displayed in the map. Any query that is supported by the IBM TRIRIGA report manager can be applied to the map. Both standard queries and metric queries are supported.

The query with the lowest value in the **Display Order** field is displayed by default. To select another query, you can click the **Show Details** tab above the map and select a query from the list.

In GIS sections, query data is filtered and grouped by the geographical data that is displayed in the current map view. Queries must include display columns that are labeled **Longitude** and **Latitude**. The **Latitude** and **Longitude** fields pinpoint the item on the map.

For metric queries, the structure of the query must be tabular. For metric queries, in order for a record link to appear in a pin's mouse-over dialog, the `triRecordSY` field must be included as one of the report columns. For standard queries, this step is not necessary since the record link appears in the pin's mouse-over dialog by default.

If the results of the query are in more than one map system, the GIS software does the conversion and displays the results in the basemap.

When a query runs and the query filter includes fields `triGisLatitudeNU` and `triGisLongitudeNU`, the Esri JavaScript viewer automatically adds filters to the query. The filters restrict results to within the extents of the map. The `triGisLatitudeNU` and `triGisLongitudeNU` field names must be used. Otherwise, filters cannot be added.

A map can return a maximum of 1000 records.

Basemaps

The basemaps identify the maps that are available for display.

The basemap with the lowest value in the **Display Order** field displays by default.

You can select another basemap by clicking the **Show Details** tab above the map and selecting from the **Switch Basemap** list. The **Show Details** tab also displays information about the map that is provided by the map vendor.

The basemap record requires that you specify the REST URL of the Esri server that is providing the map service. If the URL is correct and valid, the basemap service description that is provided by the Esri server renders on the bottom of the GIS Base Map form. If instead of the basemap service description, nothing displays or you see an error, confirm that your Esri server enabled the basemap service with REST endpoints.

You can change the icon that shows when the basemap shows in the **Switch Basemap** list in the **Show Details** tab of the map. When the **Thumbnail URL** field points to a valid image, it is displayed in the **Switch Basemap** list. If you do not specify a value, the default image appears.

Layers

The layers identify the legend layers that are available for display on top of the basemap.

When the **Default** check box is selected, the layer displays when the map renders. You can add a layer by clicking the **Show Details** tab above the map and selecting the check box next to the name of the layer. Clearing a check box removes that layer from the display.

When you select a layer with associated legend information, a column on the right of the map shows the legend. When multiple layers are displayed, the legends are listed in display order. As you modify the extents on the map by zooming, the data in the legends updates to reflect the correct level of detail. If a layer displays by default, the legend is not displayed until you click either the **Show Details** tab or the **Show Table** tab. Delaying the display conserves space on the map, which is relevant when a map is viewed in smaller spaces, such as in a portal. The column that displays the legends disappears after the last layer is removed.

The layer record requires that you specify the REST URL of the Esri server that is providing the map service. If the URL is correct and valid, the basemap service description that is provided by the Esri server renders on the bottom of the GIS Base Map form. If instead of the basemap service description, nothing displays or you see an error, confirm that your Esri server enabled the layer service with REST endpoints.

Icons

The pin colors or icon colors can be defined to represent a particular range of data values.

When a metric query determines the points on the map, the thresholds that are defined in the metric query control the colors of the pins. For example, the threshold in a metric query defines a value of 1 through 3 as low and the color red represents a negative result. If the query returns a value of 2, the pin is displayed with the icon file contained in the **Red Icon** field. The **Blue Icon** field is used for a value that is returned by a standard query.

You can change the icons that are displayed by uploading your files into the **Icons** section. An icon file can be in any format that is used to render on the web, for example, a `.jpg` file or a `.png` file. If you do not change the icon files, the as-delivered icons are used.

Widgets

The widgets can be defined to add a geoprocessor, an editor, or other Esri functionality via custom widgets.

Most TRIRIGA sample widgets become available when you click the **Show Details** tab above the map.

Sample widgets

The IBM TRIRIGA Connector for Esri Geographic Information System (GIS) contains sample widgets from Esri. These as-delivered widgets are only included for purposes of illustration. The sample widgets have the following characteristics:

Geocoder widget
With the as-delivered geocoder widget, you can direct the map to latitude and longitude coordinates or to an address.

Editor widget

With the as-delivered editor widget, you can draw a line, polygon, or point on a map. The entity is saved on the Esri map if you associate the feature to one of the records in the query. The as-delivered editor widget is named `sampleEditor.js`. You can use that file as an example of how to create custom editor widgets.

In the as-delivered sample editor widget, a feature can be associated with an IBM TRIRIGA ID. To associate a feature with an ID, enter a value in one of the number fields or text fields in the information pane. In the sample, the **Issue Id** field is used. You can use this same method in another editor widget to associate a feature to a value.

You can draw a feature and associate it to an IBM TRIRIGA record by clicking the **Show Table** tab and selecting the **Associate to selected feature** icon for the IBM TRIRIGA record in the table. The next time that the record is queried, the feature displays on the map. The IBM TRIRIGA ID is the common ID between IBM TRIRIGA and Esri.

When enabled, the as-delivered editor widget interferes with the bubble marker information provided by the pins. At the time of this release, there is no known workaround.

Custom widgets

Custom widgets are made up of compartmentalized JavaScript code specific to Esri widgets. With this feature, you can add a custom Esri widget that is based on the Esri JavaScript API version 4.15 custom widget might be something as simple as a map overlay or as complex as polygon editing. To define a widget, upload the JavaScript code for the widget into the **Code** field. After you add the custom JavaScript code to the binary field, that code is injected into the Esri JavaScript viewer and appended to the rendered map at run time.

For more information, see the white paper about EsriJS Widget Construction.

Widget groups

The widget groups identify the security groups that are authorized to access widgets in addition to members of the admin group.

The users in any of the listed security groups or in the admin group can see a particular widget when its **Add Security** option in the Widgets section is selected.

However, regardless of the selected widget groups, all non-admin users must be members of the TRIRIGA GIS Widget View Mode security group to view widgets on the GIS map.

Geometry services

The geometry service identifies the service that is used to process the transformation of projections for the Esri JavaScript viewer. You can reference that service in widgets and use that service for point projection.

Preview preferences

You can preview the map that is configured in the GIS map record and also save map preferences on the **Preview** tab of the form.

You can save the basemap and extents of your current view by clicking the **Show Details** tab and then the **Save Preferences** button. Selecting the **Clear Preferences** button returns the basemap and extents to the default values that are defined in the GIS map record. The preferences are saved by an association from the GIS map record to the GIS user preferences object. You can store preferences for each map.

You can see the data that is represented on the map in a table by selecting the **Show Table** tab. When you select a row in the table, the map zooms to the point on the map corresponding to that record and centers there. Selecting the **Export** link in the upper-right corner of the table downloads the table as a tab-delimited text file.

The value in the **Constructed URL** field on the **Preview** tab is the URL for the map that is displayed in the **Preview** tab. You can copy the value of the **Constructed URL** field and paste it into a location record to tie the location to the map.

Configuring GIS maps

You use GIS map records to display GIS maps in your IBM TRIRIGA portals and forms. The GIS map record defines the connection between the IBM TRIRIGA application and the ArcGIS server that provides the Esri maps. The GIS map record contains the metadata that defines which maps, layers, and widgets to display.

Procedure

1. Go to **Tools** > **System Setup** > **GIS** > **GIS Map**.
2. Add a GIS map record.
3. Specify the map name, the minimum and maximum extents, and the well-known ID (WKID) for the map.
4. Identify the queries to define the data results that are shown on the map.
5. Identify the basemaps and their URLs to the Esri server that is providing the map service.
6. Identify any layers to provide the legend information that is shown over the map.
7. Replace any icons to customize the pin images that are shown on the map.
8. Identify any widgets to provide more map tools or customize new map tools.
9. Identify any widget groups to receive security access to widgets, in addition to members of the admin group.
10. Specify any geometry service that transforms projections for the Esri JavaScript viewer.
11. Create and save the record.

Adding custom GIS portal sections

A GIS portal section points to an in-application web page that contains a GIS map. The GIS map is associated to a GIS map record that defines the data to render. A GIS portal section has a portal section type of **External**, where you specify the URL of the map.

About this task

You can use one of the following as-delivered GIS portal sections with the in-application web pages for reference:

- `triURL - GIS - Environmental Manager/Planner`, or
- `triURL - GIS - Environmental Manager/Planner (US Govt)`

Procedure

1. Configure a GIS map record. Define the extents, queries, basemaps, layers, and other map elements. For this example, the name of the map is `My First Map`.
2. From the portal builder, create a new portal section with a portal section type of **External**.
3. Enter the URL of the map.
   For this example, the URL is `/html/en/default/rest/EsriJS?map=My First Map`.
   The map parameter value of `My First Map` tells the map viewer to render that GIS map record.

Configuring ArcGIS servers behind a firewall

Your ArcGIS server can be an external online ArcGIS server or an internal hosted ArcGIS server that is typically behind a firewall. By default, EsriJS is configured for the online ArcGIS server. So, if your IBM TRIRIGA server is not behind a firewall, no configuration is necessary. However, if your IBM TRIRIGA server is behind a firewall, which is common when you internally host your ArcGIS server, a resource file that contains the Esri JavaScript API must be added in the EsriJS ClassLoader record. This resource file is

necessary if your environment cannot access the online version of the Esri JavaScript API, because this API handles the logic to render GIS maps.

About this task

The resource file for an internal hosted ArcGIS server that is behind a firewall, must be named `EsriJS_API_4.15.zip`.

The first time that any user loads any map, the software connects to the Esri JavaScript API that is defined by the resource file in the EsriJS ClassLoader record. If `EsriJS_API_4.15.zip` is not present, the external online version of the Esri JavaScript API is used.

Existing EsriJS ClassLoader records are listed in **Tools** > **System Setup** > **System** > **Class Loader**.

Configuring Esri JavaScript API behind a firewall

If your IBM TRIRIGA server is behind a firewall and cannot access the online version of the Esri JavaScript API, which is common when you internally host your ArcGIS server, a resource file that contains the Esri JavaScript API must be added in the EsriJS ClassLoader record.

About this task

The `[HOSTNAME_AND_PATH_TO_JSAPI]` must be modified to refer to the BASE_URL variable. The resource file that you add to the EsriJS ClassLoader record must be named `EsriJS_API_4.15.zip`. After you add your `EsriJS_API_4.15.zip` file, the next time that any user accesses any GIS map, that map and all future maps are rendered from the Esri JavaScript API in this `.zip` file.

Procedure

1. Go to Esri Downloads (https://developers.arcgis.com/en/downloads/).
   a) Download the ArcGIS API for JavaScript v4.15 API (`arcgis_js_v415_api.zip`).
   b) Extract the files from that compressed file.
2. Locate `arcgis_js_v415_api\arcgis_js_api\library\4.15\init.js` and open it with a text editor.
   a) Find `[HOSTNAME_AND_PATH_TO_JSAPI]` in this line: `"https://[HOSTNAME_AND_PATH_TO_JSAPI]dojo"`
   b) Change that line to the following code: `location.protocol + '//' + BASE_URL + "dojo"`
   c) But do not remove the BASE_URL variable or replace it with your own URL.
   d) Pay attention to the location of the quotation marks.
   e) Save the `init.js` file.
3. Locate `arcgis_js_v415_api\arcgis_js_api\library\4.15\dojo\dojo.js` and open it with a text editor.
   a) Find `[HOSTNAME_AND_PATH_TO_JSAPI]` in this line: `"https://[HOSTNAME_AND_PATH_TO_JSAPI]dojo"`
   b) Change that line to the following code: `location.protocol + '//' + BASE_URL + "dojo"`
   c) But do not remove the BASE_URL variable or replace it with your own URL.
   d) Pay attention to the location of the quotation marks.
   e) Save the `dojo.js` file.
4. Locate `arcgis_js_v415_api\arcgis_js_api\library` and select the `4.15` folder. Create a compressed file of that folder.
5. Rename the compressed file to `EsriJS_API_4.15.zip`.
6. Open the EsriJS ClassLoader record.
7. In the record, add a resource file so you can upload your `EsriJS_API_4.15.zip` file. If the `EsriJS_arcgis.js` file is present, you must remove it from the resource files.

8. Save the record.

Switching from resource file to online Esri JavaScript API

You can change from the IBM TRIRIGA resource file that contains the Esri JavaScript API to the external online version of the Esri JavaScript API, by removing the `EsriJS_API_4.15.zip` resource file from the EsriJS ClassLoader record.

About this task

After you change the Esri JavaScript API configuration, the next time that any user accesses any GIS map, that map and all future maps are sourced from the new Esri JavaScript API configuration.

Procedure

1. Open the EsriJS ClassLoader record.
2. Remove the old resource file that specifies the Esri JavaScript API.
3. If you are not using the external online version of the Esri JavaScript API, you can find or create another resource file that specifies the Esri JavaScript API and add the resource file to the record.
4. Save the record.

Troubleshooting GIS maps

To help in resolving any performance or display issues with your GIS portal sections and form tabs, you can refer to the following issues, reasons, and remedies.

Common issues with GIS maps

Common GIS issues include slow response, missing elements, and duplicate records. Some of their common explanations might help to resolve the issue.

Map is slow to respond

The rate at which the map refreshes when you zoom in and out depends on the speed of your Internet connection. Each time that you move the map, a call is made to respond and redraw the map. Your Esri JavaScript viewer handles these actions. The time that it takes to redraw the map depends on the speed of the network that is used. If you use an online service such as Esri, the commands are communicating over HTTP. If you use an in-house Esri server, your intranet determines the latency.

If your Esri JavaScript viewer is operating in Microsoft Internet Explorer, verify that the following XML HTTP setting is enabled in the browser. Select **Tools** > **Internet Options** > **Advanced** > **Enable native XMLHTTP support**.

Labels in the bubble markers and the table do not appear

The labels within the bubble markers and in the table are defined by IBM TRIRIGA queries. To put the marker on the map, the data must contain and the query must display fields that are labeled **Latitude** and **Longitude**. To display a location's image in the bubble markers, the location data must contain, and the query must display, a field that is labeled **Image**.

Missing locations

- The locations are not geocoded. In order for a location to display on the map, the record must be geocoded.
- Your query does not return results.
- Your ArcGIS server is down or not responding.

Data does not appear

- All queries must resolve to a list of locations or entities.

- The system is designed to have an initial query and a basemap. A standard query can bypass these requirements and display any geocoded results on the map. The query must return columns with the latitude and longitude in the results.

Duplicate records

Duplicate records in the table are caused by a data issue with your hierarchy structures. GIS uses a flattened hierarchy table. You can rebuild the flattened hierarchy structure.

GIS section is blank

If your GIS section is a blank white screen instead of a map, check the following conditions:

- The GIS configuration pages are set up properly.
- The ArcGIS server is up.
- The ArcGIS ClassLoader is not set to debug mode.
- You have a license to use the GIS map.

Missing license message

If a message states that you do not have a GIS license, verify that your IBM TRIRIGA license file or files are up to date.

Missing resource files

The Resource File section of the EsriJS ClassLoader record must contain the following four files: `TRIRIGAIntegration.jar`, `EsriJS.jar`, `EsriJS_Assets.zip`, and `EsriJS_LanguagePack.zip`. If any of these files are not present, click **Find** and add the missing files to the record.

Missing menu items

If the **GIS Map** menu item does not appear under **Tools** > **System Setup**, then in the Navigation Builder, add the menu item as follows. Edit the existing menu collection, go to the **Navigation Items Library**, search for the **GIS Map Manager Query** item, and add the item to the navigation collection under **Tools** > **System Setup**.

If the **GIS Esri Authentication** menu item does not appear under **Tools** > **System Setup**, then in the Navigation Builder, add the menu item as follows. Edit the existing menu collection, go to the **Navigation Items Library**, search for the **GIS Esri Authentication** item, and add the item to the navigation collection under **Tools** > **System Setup**.

Removing duplicate records from GIS maps

To remove duplicate records from the table of your GIS map display, you can rebuild your flattened hierarchy structure with the IBM TRIRIGA data modeler.

Procedure

1. Select **Tools** > **Builder Tools** > **Data Modeler**.
2. Select **Utilities** > **Hierarchy Structures**.
3. Select **All Geographies** then the **Generate Data** link.
4. Select **Building Spaces** then the **Generate Data** link.
5. Select **Buildings and Land** then the **Generate Data** link.

Chapter 4. Extending connector functions

You can write extended functions for many IBM TRIRIGA connectors. These extended functions use class loaders, resource files, custom workflow components, servlet proxies, the Java programming language, and the IBM TRIRIGA Connector for Business Applications web service interface.

Overview of extended functions

IBM TRIRIGA connectors use the ClassLoader business object and resource files. The servlet proxy is an extension of class loaders.

When you write extended functions, you can distribute those functions in an object migration package. Connectors use the ClassLoader business object and resource files, and custom workflow components such as `CustomTask`, `CustomParameters`, and `CustomTransitions`. The servlet proxy is an extension of class loaders that gives a handle to the Java API for the IBM TRIRIGA Connector for Business Applications (CBA). CBA uses Java servlet-style programming for integration into external systems with custom form components.

To create IBM TRIRIGA connectors, you must be familiar with the IBM TRIRIGA Application Platform builder tools, IBM TRIRIGA Connector for Business Applications web service interface, and Java programming language. A connector can be implemented with the Java programming language only.

Custom class loaders

The IBM TRIRIGA custom class loader components work together with the system class loaders to deploy Java classes into the application server domain.

The following figure illustrates the interaction between the IBM TRIRIGA custom class loader and the system class loaders.

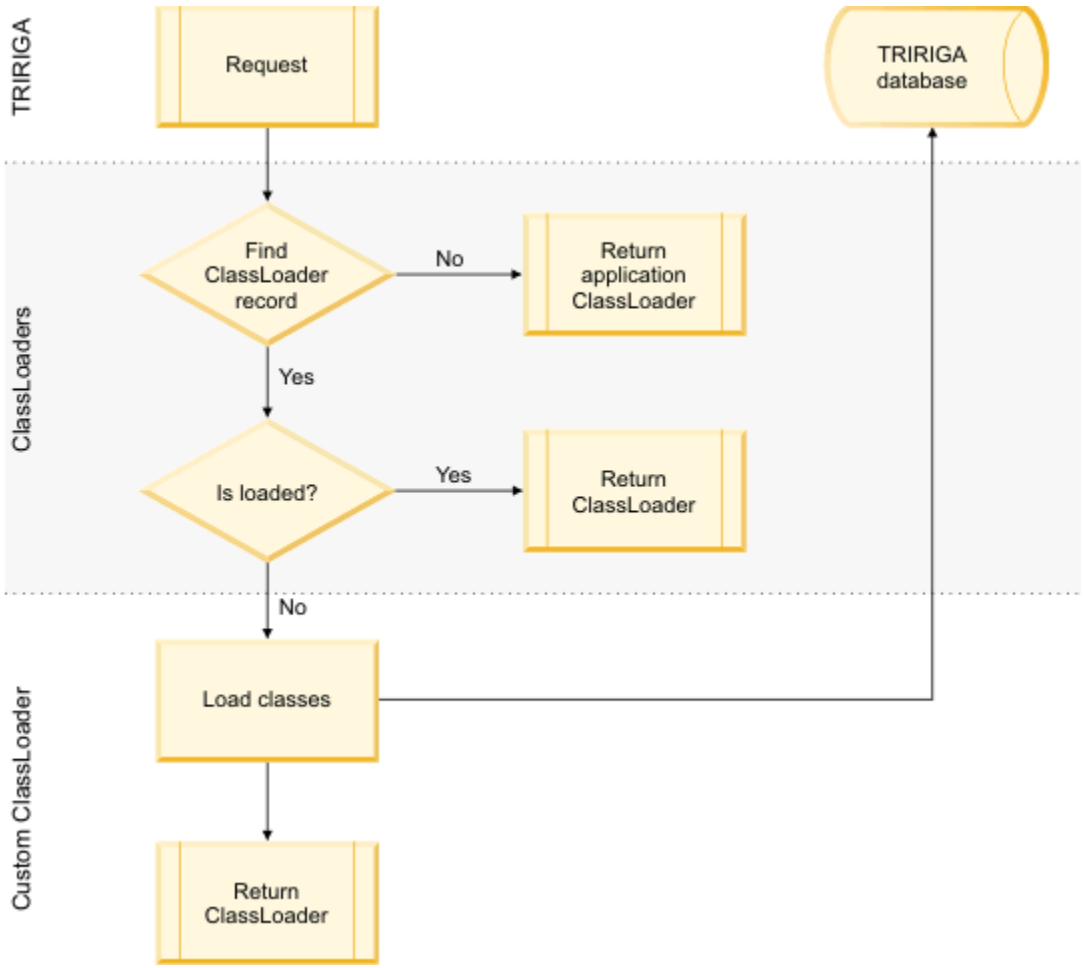

Figure 1. Custom ClassLoader

Custom classes and custom tasks

When you create a ClassLoader record, you can add custom Java classes to the class path of the application server. These classes can be accessed from a Custom task in a workflow.

You can add custom Java classes to the class path of the application server in one of three ways: Parent First, Parent Last, and Isolated. These classes can be accessed with a Custom task in a workflow for programmatic interaction or can be extended to work as a Java servlet. Either method extends the form capabilities for portal sections and form tabs.

To access your classes from a Custom task that is loaded through the ClassLoader, you must observe the following rules:

- Start your class packages with one of the following three structures: `com.tririga.ps`, `com.tririga.appdev`, or `com.tririga.custom`. Any other structure is blocked.
- Specify the ClassLoader name followed by a colon in the **ClassName** field in the workflow Custom task. For example, if you have a ClassLoader record that is named `MyClassLoader` and your entry class is `com.tririga.custom.myclassloader.Hello`, then the value in your **ClassName** field is `MyClassLoader:com.tririga.custom.myclassloader.Hello`. When you use this naming convention, workflow can search for your class within the context of the specified class loader.
- Implement one of the `com.tririga.workflow.pub.CustomTask` objects available from the as-delivered `TririgaCustomTask.jar` file in the *`InstallationDirectory`*`/tools/BusinessConnect` folder. For development, you must include the `TririgaBusinessConnect.jar` file and the `TririgaCustomTask.jar` file in your IDE class path. If you include these libraries in the ClassLoader business object, they are ignored.

For information about how a Custom task is implemented and what it offers you, see *Application Building for the IBM TRIRIGA Application Platform 3*. The ClassLoader object provides an easy handle to conduct a hot deployment and safely manage your Custom task implementations. It does not add to or change the function of a Custom task.

Application Building for the IBM TRIRIGA Application Platform 3 instructs you to put your files into the application server `lib` directory. You can skip this step, which can get complicated when you have multiple servers. Instead, add the class loader to the database and have the container intelligently extract and use the classes.

Class loader development mode

In development mode, you can change files and see your changes by refreshing the page and without uploading files to the ClassLoader record.

Normally, when you modify, add, or remove a resource file from a ClassLoader record, a workflow runs that increments the revision number. A change to this revision number tells the IBM TRIRIGA Application Platform to reload this ClassLoader record.

However, if you have access to the IBM TRIRIGA *`InstallationDirectory`*`/userfiles/`*`ClassLoaderName`* folder, you can select the **Development Mode** option in the ClassLoader record. When this option is selected, the class loader ignores the revision number and pulls the files from the *`InstallationDirectory`*`/userfiles/`*`ClassLoaderName`* folder instead.

The file types that you can change include web page (`.html`), JavaScript (`.js`), Adobe Flash, and images.

Attention: If you clear the **Development Mode** check box, the IBM TRIRIGA Application Platform pulls the latest files from the ClassLoader record and can overwrite your work.

Servlet proxies

The servlet proxy is an extension of class loaders. You use it to develop servlet-style classes that can render, redirect, and communicate with form assets within the ClassLoader record. The assets must be uploaded to the ClassLoader record as resource files.

The following figure illustrates the interaction between the servlet proxy and the IBM TRIRIGA custom class loader.

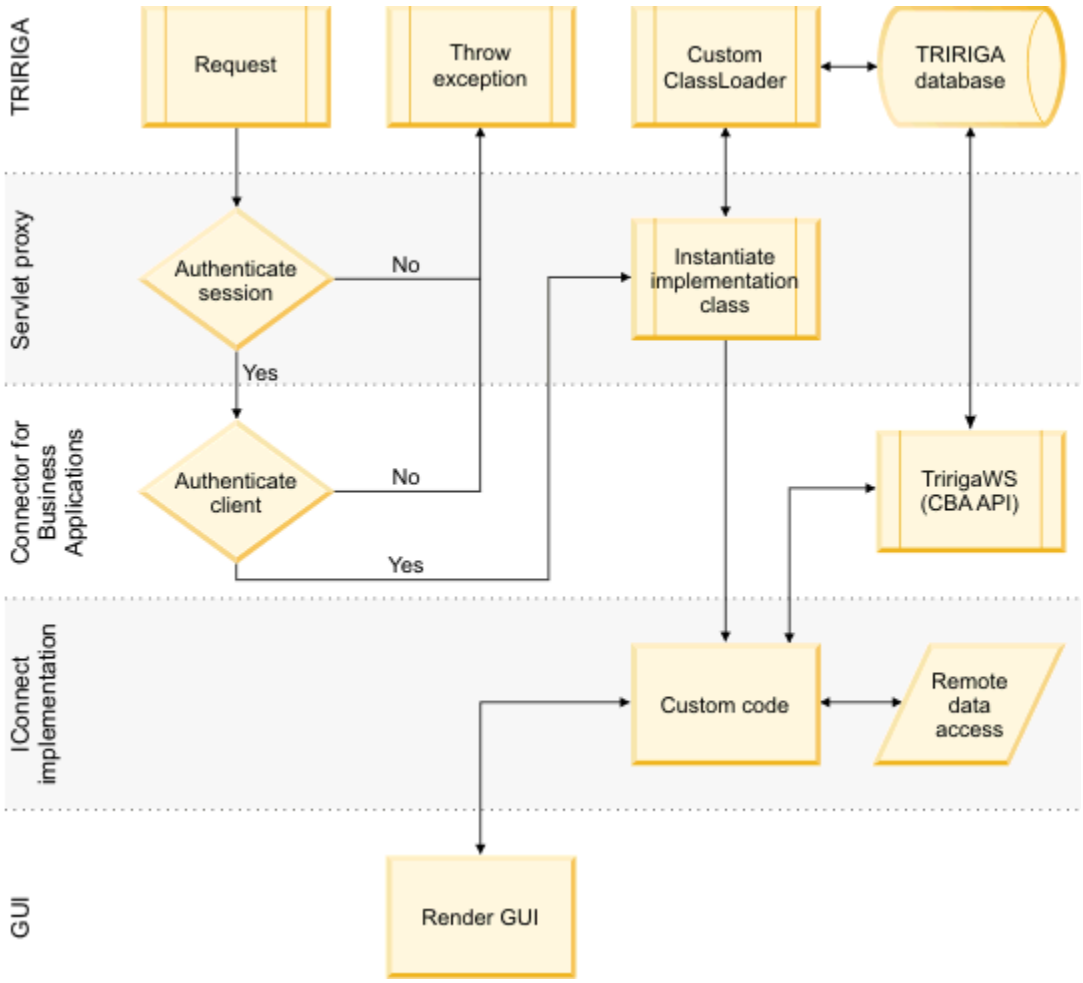

Figure 2. Servlet proxy

Servlet proxy access

A servlet proxy is available at the following URL: `http://`*`YourServer`*`/html/en/default/rest/`*`ConnectorName`*`.` If your servlet proxy is named `MyFirstConnector` and properly configured, it is available at the following URL: `http://`*`YourServer`*`/html/en/default/rest/MyFirstConnector`.

You must have a valid username and password to access this URL. You can add this URL as an external link within a portal section or within a custom form tab.

All access to your servlet proxy starts from this base URL. To access the files within your class loader resource files, append the word `resource` to the base URL, followed by the path to the resource that you want to load. For example, if you have an image as a resource file named `helloWorld.jpg`, you can load this image dynamically with the following URL: `http://`*`YourServer`*`/html/en/default/rest/MyFirstConnector/resource/helloWorld.jpg`.

The server checks whether this resource file is loaded. If the file is not loaded, the server pulls the file from the **Resource File** binary field in the resource file record and places it into the *`InstallationDirectory`*`/userfiles/`*`ClassLoaderName`* folder. When a request is made, the server checks the cache and matches its revision number to the revision number in the class loader. If the revision numbers are different, the server reloads all files that are not part of a `.jar` file into this directory. Then, the server refers to this location for each subsequent request.

Your resource files are automatically refreshed each time that a change is made. You do not need to restart the server to refresh the class loader.

Adding resource files to class loaders

You add resource files to your class loader to provide your classes and form assets, such as web page (`.html`), JavaScript (`.js`), Flash, image, and property files. For example, a JavaScript (`.js`) resource file in the EsriJS ClassLoader record specifies from where the GIS maps are sourced.

About this task

A resource file can be used by more than one class loader.

For resource files, a good rule of thumb is to prefix the library names with an abbreviation of the class loader and the real name of the library.

If you are uploading a `.jar` file, then only the `.class` files are loaded into the class path. If you have many assets such as web page (`.html`), JavaScript (`.js`), and image files, you can collect them into a compressed file and upload them as a single file. You also can upload a file individually, such as a configuration file, so that you can more easily modify it.

When you modify, add, or remove a resource file from a ClassLoader record, a workflow runs that increments the revision number. A change to this revision number tells the IBM TRIRIGA Application Platform to reload this ClassLoader record.

Procedure

1. Go to **Tools** > **System Setup** > **System** > **Class Loader**.
2. Add or open the ClassLoader record.
3. In the record, find or add your resource files.
4. Save the record.

Configuring servlet proxies

For your servlet proxy to correctly render and pass through your code, you implement a Java interface class that is contained in the `TririgaCustomTask.jar` file.

About this task

The `com.tririga.custom` package is the only package that you can use to create an implementation class. The class must be unique. You can have only one `IConnect` implementation class per ClassLoader record. The ClassLoader record name and the Java class that implements `IConnect` must have the same name. For example, if your class is named `MyFirstConnector`, then you must also identify your ClassLoader record with the name `MyFirstConnector`.

After you configure your servlet proxy, you can access the connector.

Procedure

1. Create a Java class in the `com.tririga.custom` package.
   For example, you create a class that is named `MyFirstConnector`.
2. Implement the `com.tririga.pub.adapter.IConnect` Java interface through the `execute` method in the `IConnect` class.
   For example, if your class is named `MyFirstConnector`, you create the following code:

```
public class MyFirstConnector implements IConnect {
    public void execute(TririgaWS tws, HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        //your code goes here
    }
}
```

This example has a handle to a `TririgaWS` interface class. This class is the Java interface for the IBM TRIRIGA Connector for Business Applications API. The example also shows a basic request and response that you would normally have in a Java servlet.

3. Add your code to the `execute` method where it says `//your code goes here`.
   Continuing with the example, to print a web page (`.html`) with the words "Hello World", you insert the following code:

```
PrintWriter out = response.getWriter();
try{
    response.setContentType("text/html");
    out.println("<html><head></head><body marginwidth='0' marginheight='0'
        style='margin:0;padding:0;border:0;'>");
    out.print("Hello World");
    out.println("</body></html>");
    out.flush();
} finally {
    if(out!=null)out.close();
}
```

4. Compile your Java class and add it to a `.jar` file.
   For example, you compile your `MyFirstConnector` class and add it to a `.jar` file that is named `MyFirstConnector.jar`.
5. Go to **Tools** > **System Setup** > **System** > **Class Loader**.
6. Add your ClassLoader record.
   For example, add your ClassLoader record with the name `MyFirstConnector`.
7. In the record, add a resource file so you can upload your `MyFirstConnector.jar` file.
8. Save the record.

Troubleshooting extended functions

To help in resolving any issues with your class loaders, servlet proxies, or custom connectors, you can refer to the following issues, reasons, and remedies.

Common issues with extended functions

Common issues with class loaders, servlet proxies, and custom connectors can be resolved by activating or customizing the platform logs.

To start platform logging

In the administrator console, in the **Platform Logging** managed object, turn on debugging for the **Class Loader** and **Servlet Proxy** categories. These verbose-mode logs can give you a good understanding of what the server is doing.

For information about how to access and use the administrator console, see the *IBM TRIRIGA Application Platform 3 Administrator Console User Guide*.

To simplify the debugging of class loaders and servlet proxies

In the administrator console, to add a custom category to the **Platform Logging** managed object, add the custom category to the `CustomLogCategories.xml` file and restart the server. The `CustomLogCategories.xml` file is in the *`InstallationDirectory`*`/config` folder.

This method is preferred because you set it up one time. If the server is restarted, you can turn **DEBUG** back on by selecting the option for your custom category.

The `.jar` files do not deploy as expected

Do not add multiple instances of the same `.jar` file to a ClassLoader record. For example, one that is added directly and one that is contained within a compressed file. When this occurs, the instance of the `.jar` file that is loaded is not predictable.

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: