TOC

About This Guide

This is an archived copy of the v23 documentation provided for Logi Info v23.3 and its service packs.

Notice: Archived Documentation

This documentation is provided as a courtesy reference for a version of our software that is no longer under active development or support. The information contained herein is offered without warranties of any kind, either expressed or implied, including but not limited to warranties of accuracy, completeness, or fitness for a particular purpose.

While this archived material may assist with understanding historical functionality, please be aware that the software described is no longer maintained at this version level and may contain outdated or inaccurate information. Images may not reflect currently supported modules, support sites, or third party products. This software may not be compatible with current versions of previously compatible third party products.

To access and upgrade to current software solutions and receive ongoing support, please contact our customer support team. They can assist you in migrating to the latest appropriate software version that meets your needs. Our support team is available to help ensure a smooth transition to actively maintained alternatives that provide the functionality and reliability you require.

Integrate Logi Reports with Other Applications

Developers frequently ask how Logi applications can be **integrated** with their other applications and this topic provides an overview.

Topics related to integrating Logi reports with other applications include:

About Logi's Zero Footprint

A major consideration when discussing the integration of two applications is what kind of **impact** a second application will have on the performance and operation of the main application. As free-standing web applications, Logi applications are designed to have a "zero footprint", which is to say *no impact at all*. Just think of a Logi application as a separate web site your main application connects to. Logi reports can be run in a way that entirely **isolates** them from the main application: no shared code, no supporting library dependencies, and no mutual versioning considerations. As free-standing web applications, Logi reports can also be run on **separate servers** so that there need be no performance or storage impacts on the main application.

No Common Code Required

From a development perspective, Logi reports do not need to be combined with the **code** of the main application. There are no function or object libraries**,** no compiled **API** that developers must add to their main application code in order to access the power

and flexibility of Logi reporting. Logi .NET applications are typical ASP.NET apps but none of their code is placed in the Global Assembly Cache. Logi Java applications use their own .jar/.java files.

Direct Datasource Connection

Logi applications connect directly to a wide variety of datasources, so they do not need to rely on, or go through, the data connectivity of the main application. So main application performance is not affected by data retrieval and processing activities in the reporting application.

Embedding in Desktop Applications

If developers want to **embed** Logi application output in their **desktop applications**, this can be easily accomplished. Their desktop application just needs the ability to make an **HTTP request** and to **render** the **HTTP response**. Many development languages include browser objects that can do this; for example Microsoft Visual Studio includes a Web Browser component. Once again, the formal interaction with a Logi application is a simple URL call rather than actual code integration.

Including in Main Applications

Developers may want to actually include their Logi report as part of their main ASP.NET or Java web application. It *is* possible to do this, however, it introduces dependencies and complications, so it's not a recommended approach. For example, the web.config file used by a Logi .NET reporting application is slightly different from that used by a traditional .NET application, and when the Logi app is part of the main app, these differences have to be resolved for both to run correctly. The flexibility and performance provided by running Logi applications as separate applications usually outweighs the convenience of packaging everything together within the main application.

Integrating Security

Our "Logi Security" features offer **secure access protection** at the report, object, data row, and even (in Logi Info) down to the data column level. Logi Security is very flexible and can operate in a stand-alone mode to independently retrieve security credentials for authentication and authorization from OS-based security systems, such as a **Windows Active Directory** domain, from LDAP directories, from custom security databases, and from a variety of other sources. When necessary, a standard login page is available; customized login pages are supported. For those interested in a "single sign-on" implementation, **Logi SecureKey Authentication** allows a main application to handle logins and authentication, and to securely pass user security authorization information into a Logi application. More information about security is available in our *Intro to Logi Security* .

Logi Plug-ins

Logi applications support the use of "Plug-ins", which are compiled .dll or .jar files, written with modern development languages such as Java, Visual Basic.NET, or C#. Plug-ins allows developers to dynamically change the data, the design, and the behavior of Logi reports at runtime, and provide a way to access system resources and other objects in the scope of the web server OS and platform. Plug-ins can also be used to let the Logi application interact directly with the main application. More information about plug-ins is available in *Logi Plug-ins*.

Scripting and Embedded Reports API

Logi applications support the use of JavaScript within them and can reference external scripting libraries, such as JQuery. Logi Info also includes our own *Embedded Reports API*, with makes it a breeze to embed active Logi reports into other HTML pages.

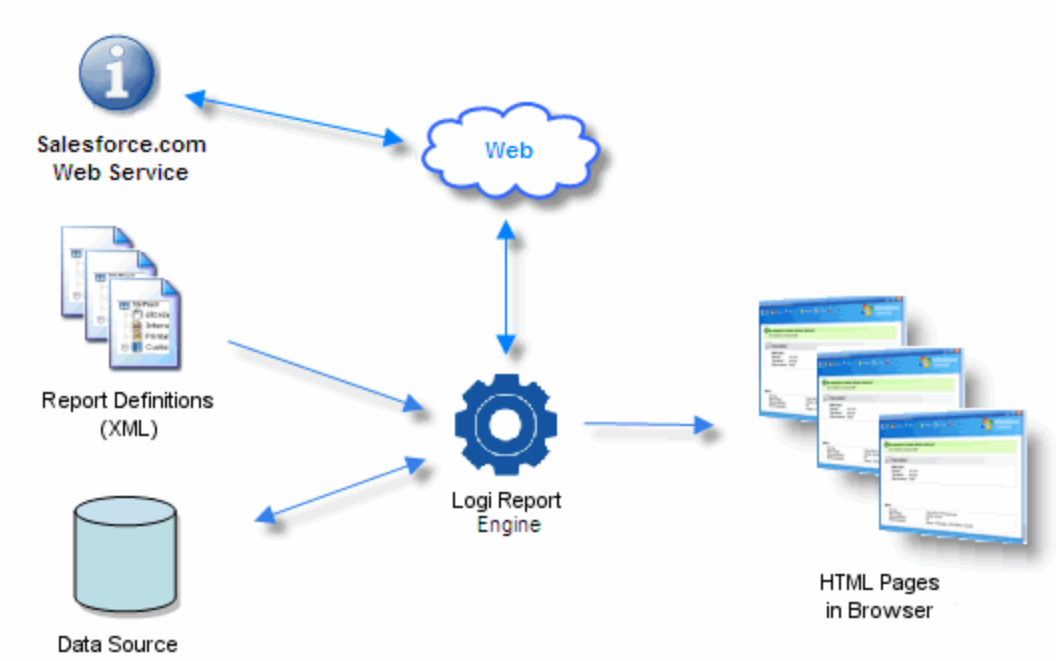
Work with Salesforce

Salesforce is a web-based Customer Relationship Management (CRM) service. Developers can use Logi Info with the Salesforce web service to provide reporting and business intelligence based on your Salesforce data.

The following topics discuss the use of Salesforce:

What's a Web Service?

The formal definition of a web service is "a software system designed to support interoperable, machine-to-machine interaction over a network". In practical terms, developers can think of a web service as a programming API that happens to be hosted on an external machine that your program reaches over the Internet. You send it *parameters* and the web service sends you *results*, with the benefit that the whole process is language neutral.



The diagram above illustrates how this fits into the Logi application architecture. Parameters and queries sent to the web service may rely on data retrieved from local data sources. The results returned from it are used in the HTML pages that are output to a browser. Logi Studio connection elementsmake the process of connecting to, and communicating with, the web service very easy.

Web Service providers, like Salesforce, make their web services available to Internet users for a fee. This means you must *sign up and pay*, usually on a transaction-by-transaction basis. However, Salesforce allows developers to access their services on a trial basis for free (the "Developer Edition"). Before you can use Logi Info with Salesforce, you need to sign up at www.Salesforce.com and get your security credentials (see below).

Salesforce Requirements

In order to use a Logi application with Salesforce you will need:

1. An account and credentials for Salesforce. Developers who want to use their free test service can get credentials at: http://developer.force.com/gettingstarted.

   Connection.Salesforce element uses the **Salesforce API**, which is not available for all versions of Salesforce. For example, it is not enabled for the Professional version. *Ensure that your organization is using a version of Salesforce that makes the API available before proceeding.*

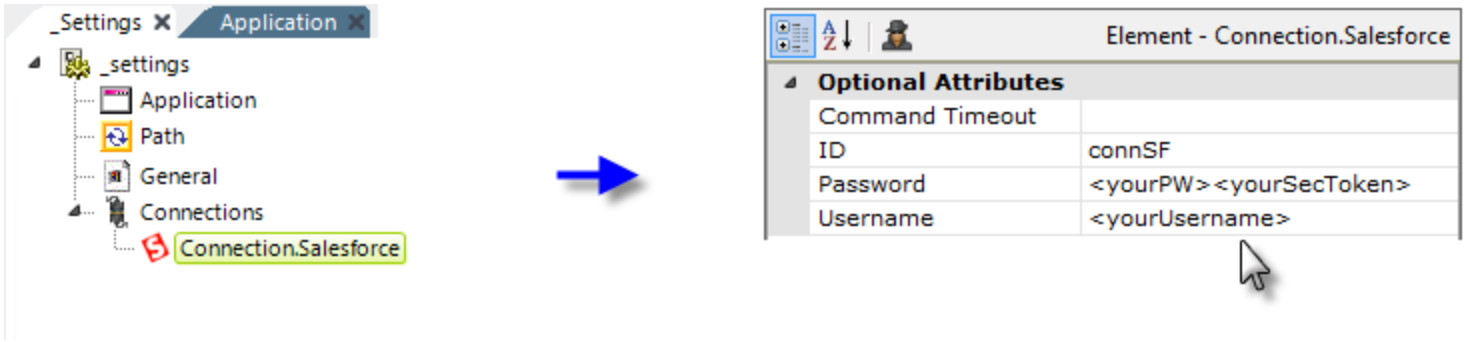
   Your credentials will include a "security token", which is concatenated with your Salesforce password, when configuring the Connection.Salesforce element. This token can be generated by logging into Salesforce, clicking **Setup**, and then going to the My Personal Information section. Click **Reset Security Token** to have a new token emailed to you. This token is machine address-dependent and may need to be reset if you move your application to another machine.

3. When your finished application runs, it obviously needs to be able to access the Internet in order to interact with the web service. There may be firewall and security implications for you to consider.
4. Logi reports built using Salesforce data can be *exported* to other formats, such as PDF, Excel, and Word.

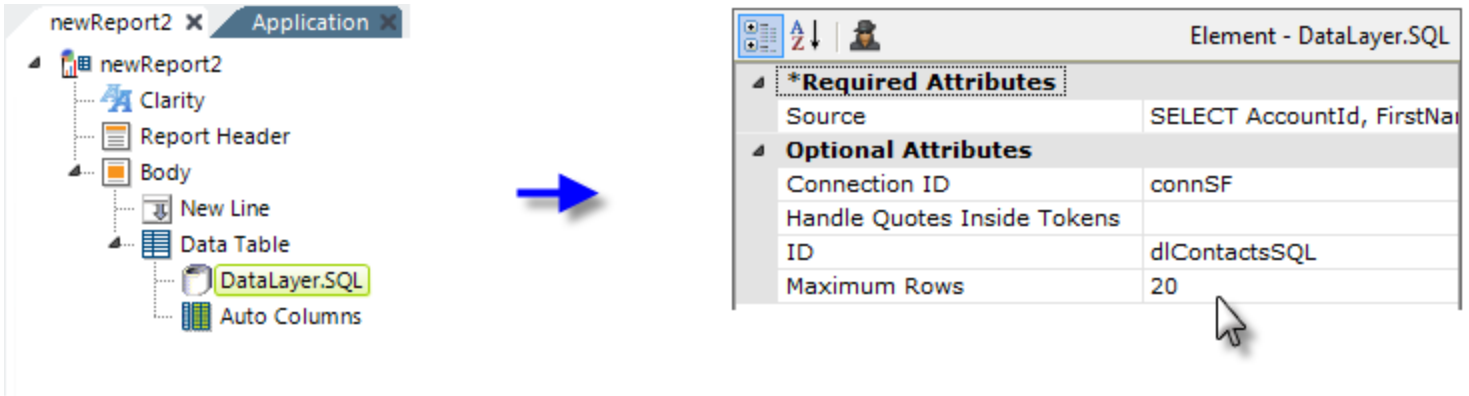
Salesforce.com has discontinued support for communications using the TLS 1.0 protocol. If you're using **Connection.Salesforce**, or REST or SOAP API connections to Salesforce, you need to upgrade to Logi Info v12.2-SP4 or later, which supports the TLS 1.1 and 1.2 protocols, in order to continue to connect when your Salesforce instance is changed. In addition, Java applications must use Oracle JDK 8+ or OpenJDK 8+ to use the appropriate protocol version.

Using Salesforce Data in a Report

Using data retrieved from Salesforce is very easy and straightforward. In the following example, we'll create a simple report that displays data from the sample Contact database on Salesforce:



1. In the Definition Editor, in the _Settings definition, add a **Connection.Salesforce** element, as shown above.
2. Set its attributes as shown, providing your Salesforce account credentials as required. Your Salesforce password and Salesforce security token are entered as a concatenated string. For example, if `XXXXX` is your password and `FiSg46Rkrp3FmbVzl` is your token, the Password attribute value is: `XXXXXFiSg46Rkrp3FmbVzl`.



3. In your report definition, add a **Data Table** and **DataLayer.SQL** element, as shown above. Set their attributes as shown.
4. Set the DataLayer element's **Source** attribute as follows:

```
SELECT AccountId, FirstName, LastName, Salutation, Phone, Email, Title FROM Contact
 WHERE AccountId <> '' and LastName <> 'Unknown'
```

Salesforce uses a special variant of SQL, the Salesforce Object Query Language (SOQL), which has some important differences from standard SQL (for example, it doesn't include DISTINCT). More information about it can be found in the *SOQL Language Reference* at the Salesforce website.

 Because SOQL is sufficiently different from standard SQL, Logi Studio's **SQL Query Builder** tool *does not* support it.

5. Set the **Connection ID** attribute to the ID of the Connection.Salesforce element from Step 1, as shown.
6. In this example, we'll save a few keystrokes by adding an **Auto Columns** element instead of all of the necessary Data Table Column and Label elements. In other report definitions that don't use Auto Columns, the Salesforce data in the datalayer is available, as usual, using @Data tokens.
7. Preview the report.

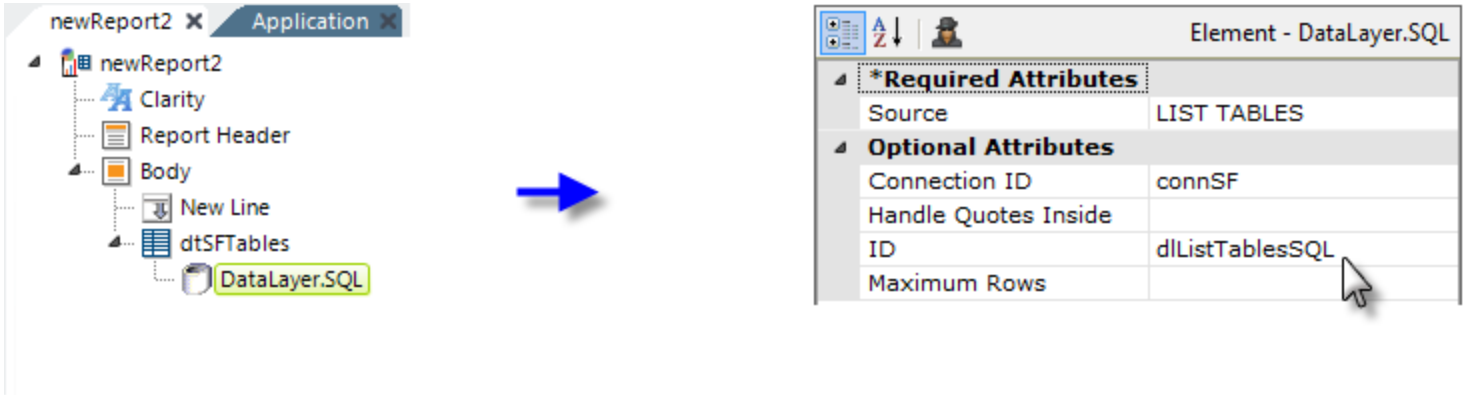
AccountId	FirstName	LastName	Phone	Title	Salutation
001C0000014oZbUIAU	Sally	Grant	(843) 413-8183	Senior Project Manager, R&D	
0018000000Pi6eOAAR	Avi	Zacha	97237667822	Dir Dev	
001C00000106aZXIAY	Mike	Forrest	516-809-7912	Network - works with Vito	Mr.
0018000000PK0bFAAT	Huy	Trante	(312) 279-8612	Manager, Product Development	
001C000001317sWIAQ	Manda	Deshande	5386873464		
0018000000OAIQ9AAL	Pascale	BelMonde	(450) 978-0792	Project Manager	
001C000000xXYHyIAO	Lakshmi	Dinrasha	267-840-1112	Application Developer	
001C000000xFrrkIAC	Dennis	Crenshaw	(215) 326-7800	Sr. Web Developer	
001C000000s1muBIAQ	Bonny	Rogers	7609298121	PM	Ms.
001C000000zrHvTIAU	Praka	Bhuvan	3145537822	Technical Services Manager	
0018000000NjTyNAAV	James	Ward	2407520045	CTO	
0018000000MTeP2AAL	Peggy	Hymann	8044161808	Chief Applications Officer	Ms.
0018000000PLB03AAH	Chris	Sidlow	(516) 981-2700	Co-Founder	Mr.
0018000000PLB03AAH	Jennifer	Ditson	212.337.2700 x21	Media Analyst	Ms.
0018000000NjRUaAAN	Jason	Kramer	61433784003	Consultant	Mr.
001C00000106aZXIAY	Tony	Willson	(516) 803-6100	Sr. Buyer	Mr.
001C00000106aZXIAY	Arthur	Panket	516809XXXX	Acounts Payable email address	
001C000000xFNeOIAW	Mike	Baldini	2128793135	VP, Director Mgmt Engineering	
001C000000xCvuxIAC	Venkat	Tewari	17244416400	Product Mgt and Marketing	
001C000000xEA0pIAG	Jane	Middleton	408-765-9235		

The preview should look something like the example shown above.

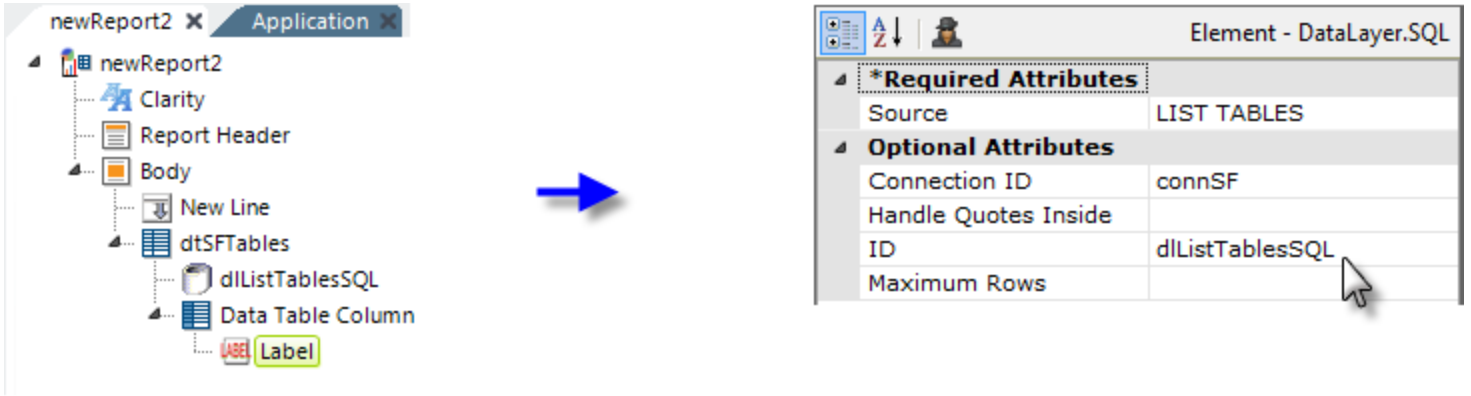
Getting a List of Available Salesforce Objects

How do you determine what objects, specifically tables and columns, Salesforce makes available?

To get a list of tables, you use a special SOQL query using the command **LIST TABLES**. This query will return a list of table names; these names, in turn, can be subsequently used with LIST to retrieve the columns for each table. Here are some examples of the code to put this in action:

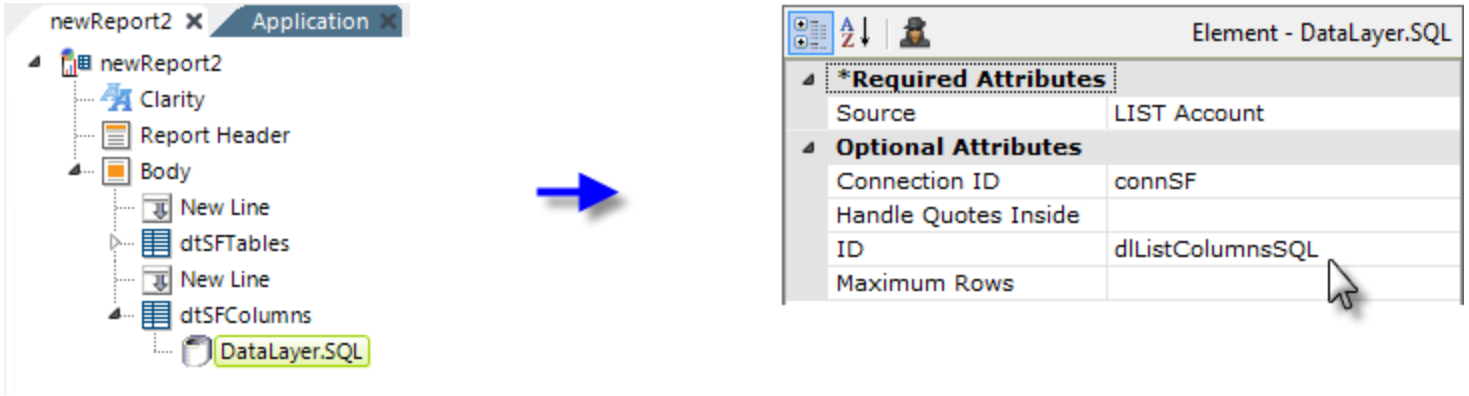


As shown above, a simple Data Table and DataLayer.SQL element are used to retrieve a list of tables from Salesforce. The special SQL query syntax "LIST TABLES" is used in the datalayer's Source attribute.

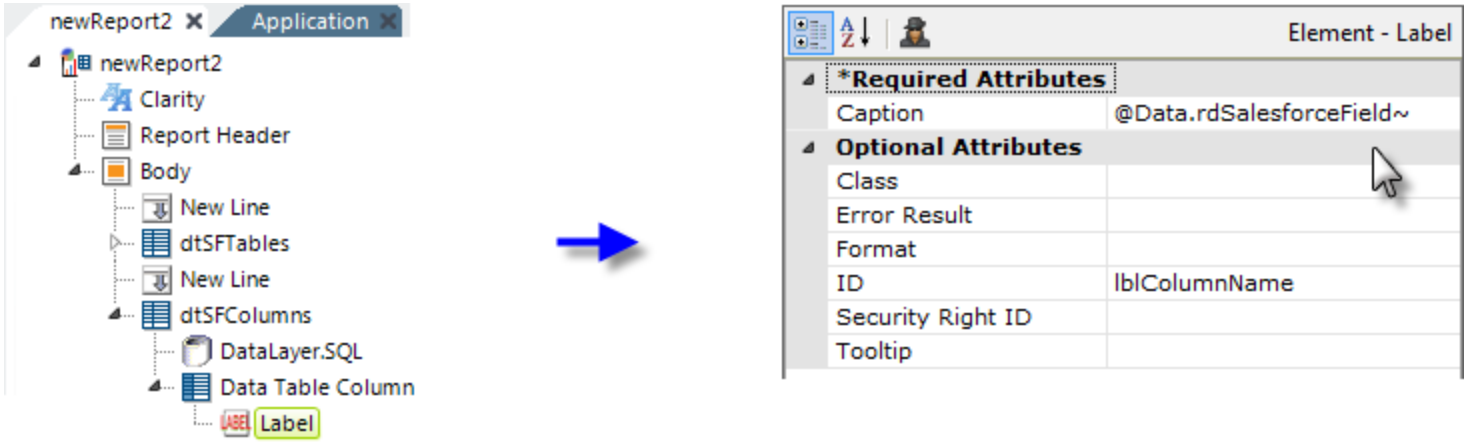


The results returned from Salesforce can be accessed using the **special Data token** @Data.rdSalesforceTable~, as shown above,

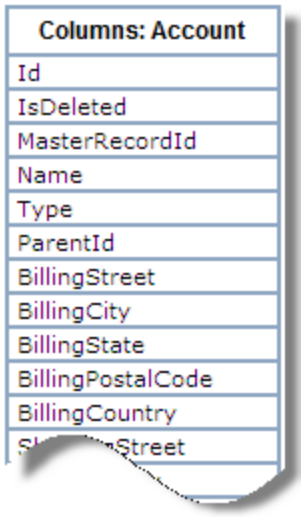
and the output will look something like the example above. Once the table names have been acquired, they can be used in a query to retrieve a list of the columns for a particular table.



In the example above, a second Data Table is used to list the columns associated with one of the Salesforce tables. The SOQL query syntax is LIST <table name>, and in the example, a table name has been hard-coded into the query. A more sophisticated solution might use a table name selected from the displayed results of the previous (LIST TABLES) query to drive the second query, using a request token: LIST @Request.TableName~, for example.



And finally, as shown above, another **special Data token** @Data.rdSalesforceField~ is used to display the Salesforce table column information the query retrieved.



The output would be a list of columns, as shown above.

By combining the techniques shown here, developers can access data stored on Salesforce and incorporate it in their Logi applications.

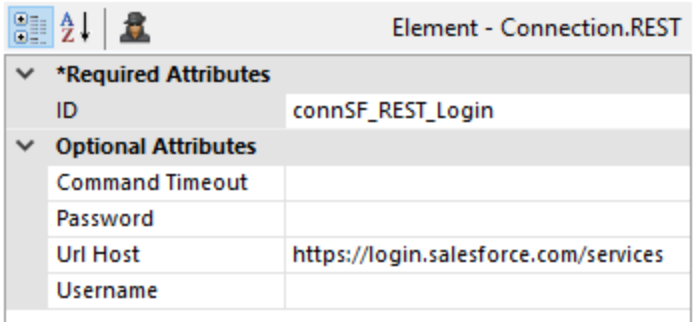
Reading and Writing to Salesforce

Beginning with Logi Info v12.2 SP5, it's possible to use Salesforce's REST API to "write back" (Insert, Update, Delete) data to your Salesforce instance. This is a complex development task, however, and requires in-depth knowledge of both Logi Info and Salesforce.

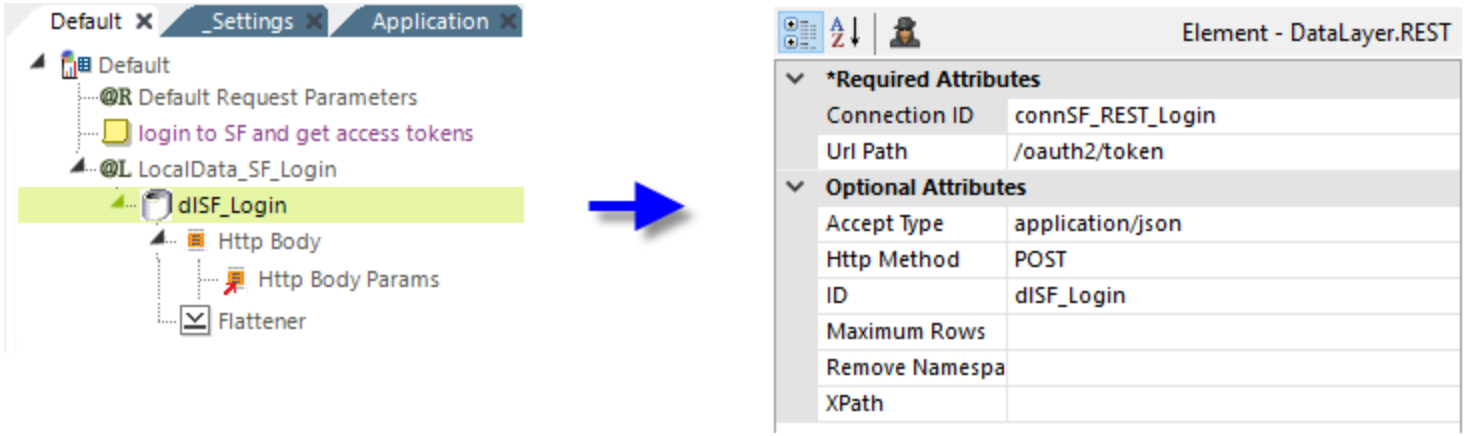
The following discussion of operations using the Salesforce REST API assumes a security model in which a single Salesforce account (the "service" account) will be used to access Salesforce on behalf of all users of your Logi application. This account will require a standard Salesforce User License and should be restricted to limit its interactions to just the Salesforce objects to be modified. *Do not use an Administrator account for this*. You'll need to know the account's Salesforce password and security token value. Salesforce uses OAuth 2 security, which includes a Username-Password authentication flow mode, which will be used in this example.

The general steps for developing a Logi application that uses the API are:

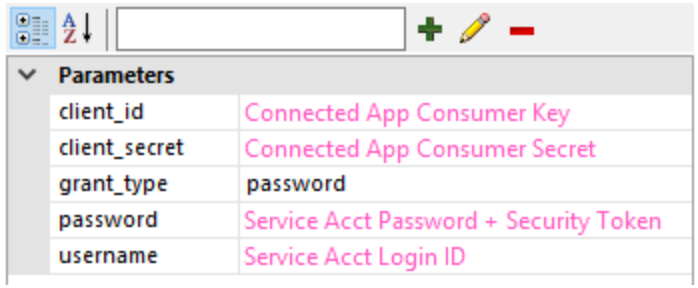
1. Register your Logi application as a "Connected App" in Salesforce, and get its "Consumer Key" and "Consumer Secret" values. See this Salesforce document for more information.
2. You may need to create a Salesforce *Custom Profile* and *Permission Set* for the service account, in order to ensure access to the Salesforce objects you want your Logi application to interact with.



3. As shown above, a **Connection.REST** element is used in the _Settings definition to connect to Salesforce in order to login. No login credentials are configured in this element.

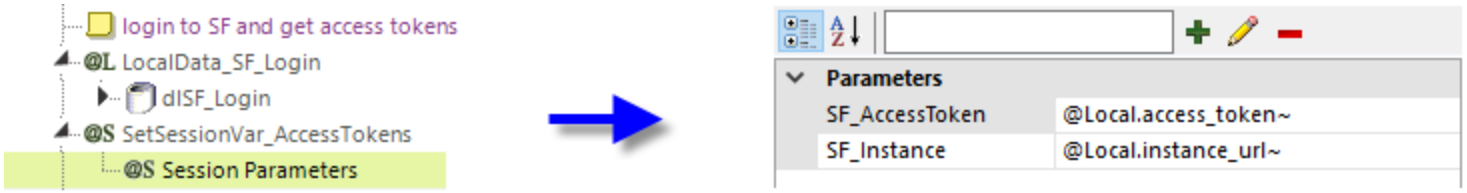


4. In your application's Default report definition, add a **Local Data** element and **DataLayer.REST** to perform the login. Set the Local Data element's Condition attribute to "@Session.SF_AccessToken~" = ""to ensure that the access token will only be requested once per session (it has a default 4-hour lifetime).

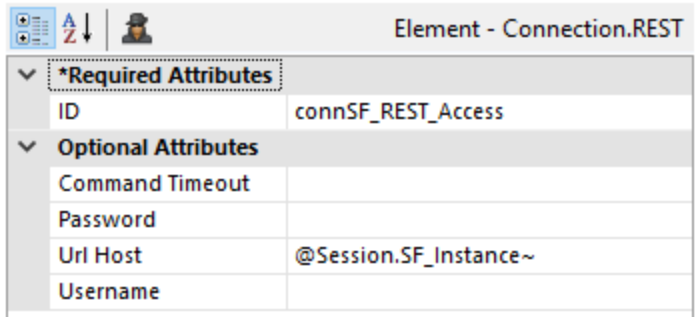


An **Http Body** element is used to pass parameters during the login. The **Http Body Params** name-value pairs that need to be passed are shown above. The client_id and client_secret parameter values are, respectively, the Consumer Key and Consumer Secret values you noted when you created the Salesforce Connected App. The grant_type value must be *password*. The password parameter value is the Salesforce service account's password, concatenated with its security token. The username is the account's Salesforce login ID.

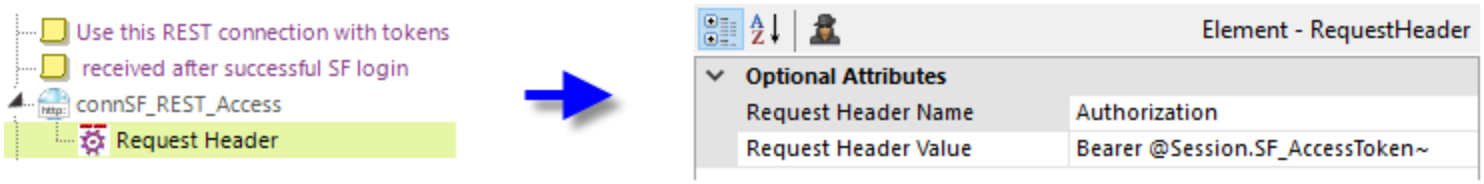
5. Finally a **Flattener** element, with no attributes set, is used to make the returned values usable as @Local tokens.



6. Next, add a **Set Session Variables** element and configure its parameters as shown above. This makes the values returned from a successful login available for later use as session variables.



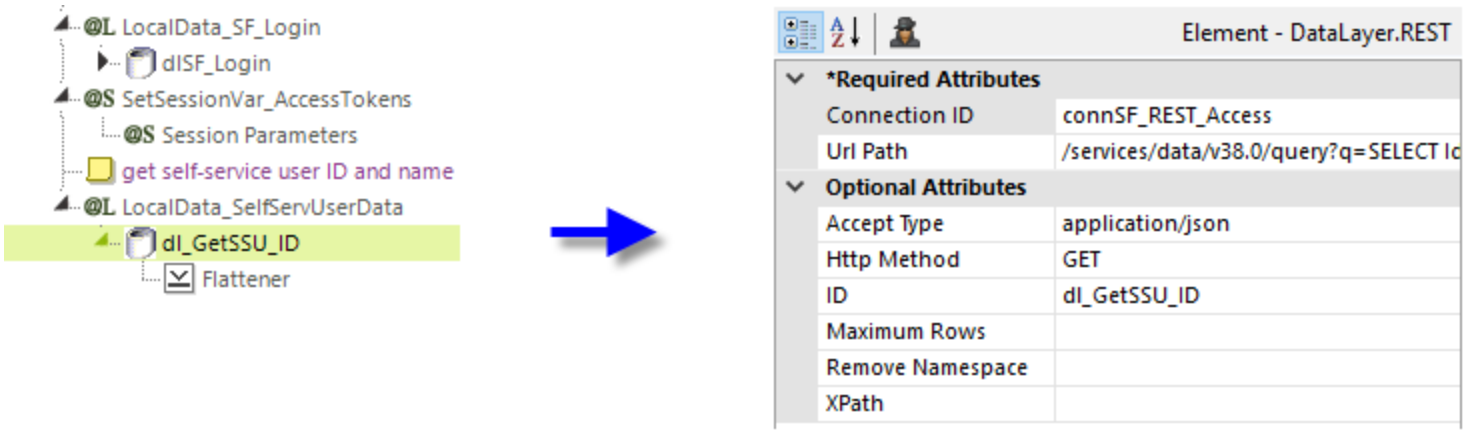
7. Back in the _Settings definition, add another Connection.REST element, to be used when making requests using the Salesforce REST API. Configure is as shown above, with one of the session variables created in the previous step.



8. Add a **Request Header** element beneath the Connection element, as shown above, and configure it as shown, using the second session variable.

Making Salesforce API Queries

Now we're ready to make a call to the Salesforce REST API, using **DataLayer.REST**, as shown below:

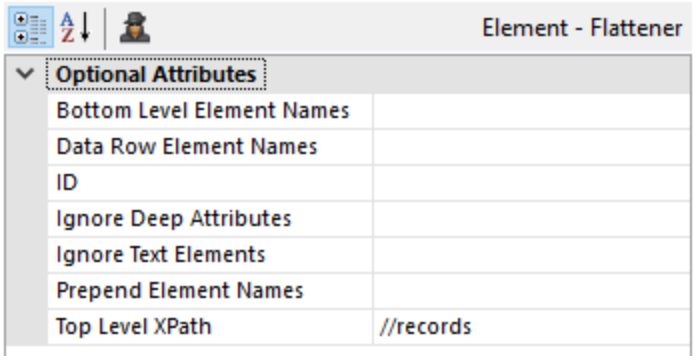


The Accept Type value is *application/json* once again, and the Http Method this time is *GET*. The Url Path attribute contains the Salesforce Object Query Language (SOQL) query and here's an example:

```
/services/data/v38.0/query?q=SELECT Id, FirstName, LastName, Email, SuperUser FROM SelfServiceUser WHERE ContactId = '@Request.someID~'
```

You'll recall that, at execution, the datalayer's Url Path attribute value will be appended to the Connection element's Url Host value to make a complete URL.

Once again, we'll need a **Flattener** element to shape the returned JSON data. In the case of SOQL queries, most of the time we need to tell the Flattener where to begin, by using its Top Level XPath attribute:

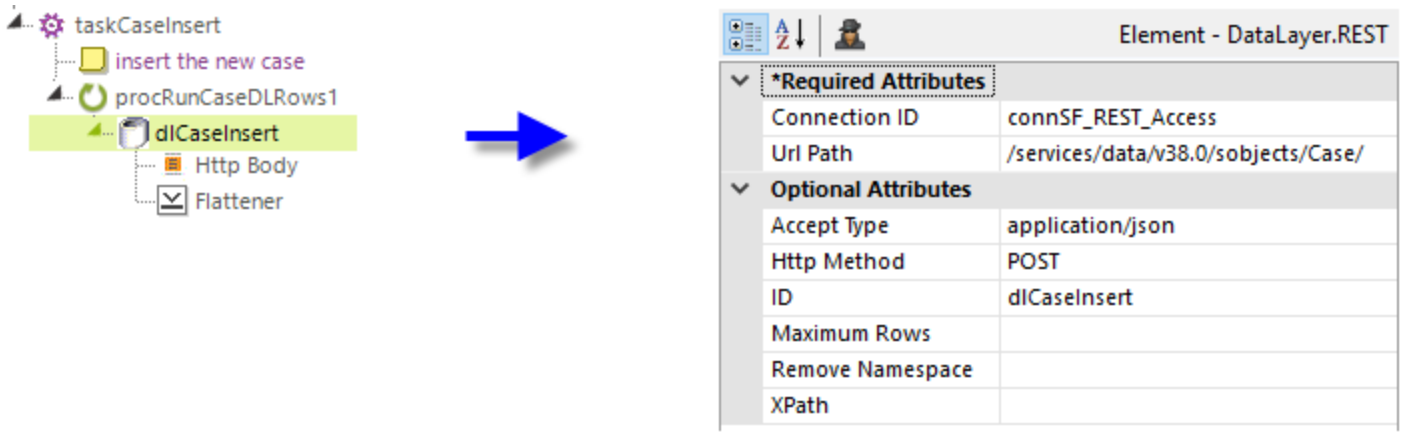


And in most cases, the *records* node, as shown above, is the correct one.

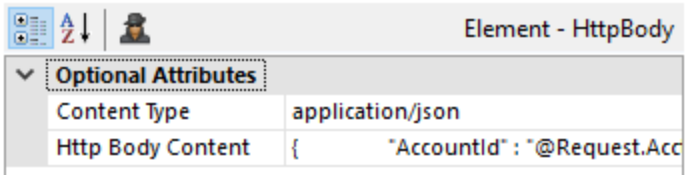
Writing to Salesforce via the API

Writing to Salesforce is somewhat similar to making queries, in that the same Connection.REST element is used and important information is placed in the URL Path attribute. However, there are some significant differences. The suggested approach is to use Process tasks, and you might think to use the **Procedure.REST** element to call the API.

However, when you make an API call you may want to use data it returns, but the Procedure.REST element is *not* a parent of the Flattener element, so you can't get the returned data into the proper format. The solution to this is to use a **Procedure.Run DataLayer Rows** element with a child DataLayer.REST element.



As shown above, the configured datalayer's attribute values are similar to those provided in the earlier Query example, except the Http Method is now *POST*. The Url Path is not a SOQL query but a direct reference to a Salesforce object endpoint, in this example, the Case object. This is the way the API expects to receive a command to insert that object.



In addition, an **Http Body** child element, configured as shown above, is used to pass the values to be inserted. Here's an example of the JSON data array in the Http Body Content attribute:

```
{
"AccountId" : "@Request.Acct_ID~",
"Description" : "@Request!Json.hdnDesc~",
```

```
"Product__c" : "Logi Info",
"Product_Version__c" : "@Request.inpVersion~",
"Service_Pack_Version__c" : "@Request.inpSPNo~",
"Subject" : "@Request!Json.inpSubject~"
}
```

Note the use of literal and @Request token values. Also, notice the use of the highlighted !Json token modifier. It's used when the data is free-form text and it encodes any special characters, such as double-quotes and line-feeds, so that Salesforce won't reject the Json data stream as invalid.

Once again, a Flattener is used to shape the returned data in the example but, in this case, no Top Level XPath attribute needs to be specified.

 Please remember that the Salesforce API may not enforce referential integrity and therefore it may be up to you to ensure that all related tables are updated and other appropriate actions aretaken for each operation, in order to ensure a complete transaction.

Work with Twitter

The social networking site **Twitter.com** allows its members to post brief public messages ("Tweets") and private email-like messages ("Direct Messages"). Elements available in Logi Info allow developers to interact with Twitter in their reports and applications.

The following topics discuss these interactions with Twitter:

- Connecting to Twitter
- Searching for and Retrieving Twitter Posts
- Posting Tweets and Messages


Twitter updated its API to v1.1 and Logi Info applications built with a version earlier than 11.4.046 will no longer be able to connect to Twitter. You must upgrade in order to restore connectivity.

About the Twitter API

Logi Info's Twitter-related elements make use of the Twitter v1.1 REST API. This is a collection of web service methods that allow the developer to interact with Twitter at a programmatic level. There is no fee or license required to use the "Standard" API; all that's needed are the credentials for a valid Twitter account and application registration. More functionality is available, using other levels of the API, for a fee. There are also several different Twitter APIs available, depending on what you want to do.

The Twitter API is maintained by Twitter and is subject to change. This may affect Logi applications if a new version of the API does not maintain backward-compatibility with earlier versions. For example, this might cause a problem if the names of the data columns returned to a Logi application change. Developers who use the Logi's Twitter-related elements are encouraged to keep abreast of API updates.

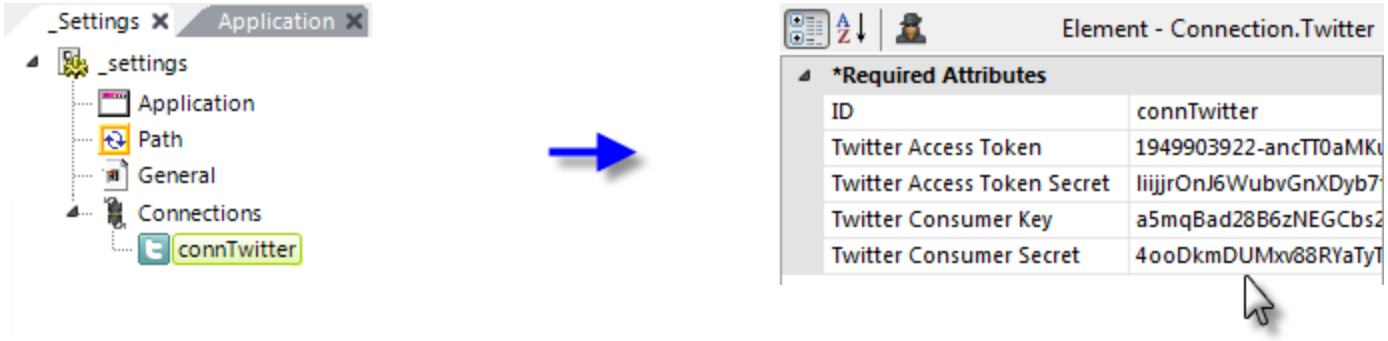
Information about the Twitter v1.1 API can be found in Twitter's *API Documentation*.

 In order to manage their resources effectively, Twitter imposes limits on the number and frequency of queries from custom applications, as described in *Rate Limiting*. Refer to this document to see how these limits may affect you.

Connecting to Twitter

You will need a Twitter regular account, of course, and you'll also need a Twitter "developer account". Then you'll have to register your Logi application, which is a complicated and time-consuming exercise. At the end of the process, you'll get *four values* from Twitter that you'll use when configuring your connection: Access Token, Access Token Secret, Consumer Key, and Consumer Secret.

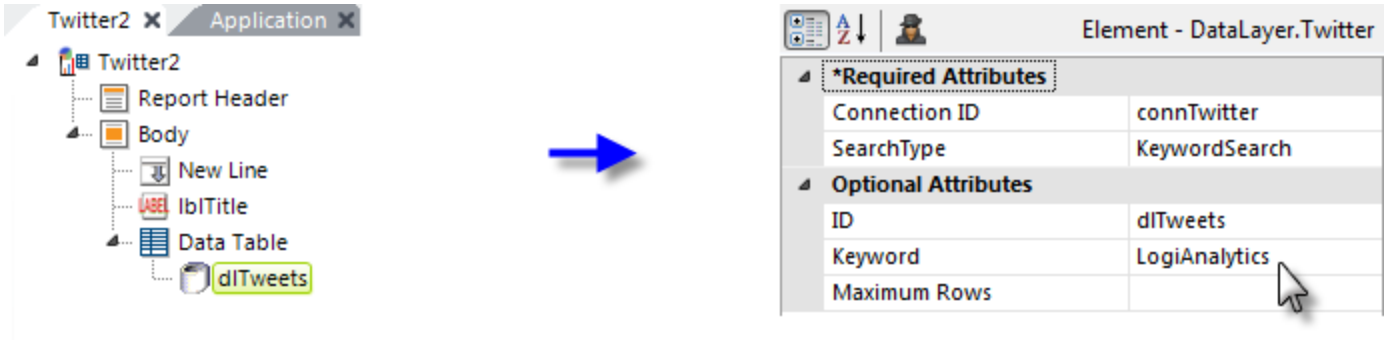
Once that's done, you can begin to work in your _Settings definition with the **Connection.Twitter** element, which manages authentication with, and connection to, Twitter for posting or retrieving data.



As shown above, the Connection.Twitter element is added beneath the Connections element in the _Settings definition and its attributes are configured using the four values generated when you registered your application with Twitter.

Searching for and Retrieving Twitter Posts

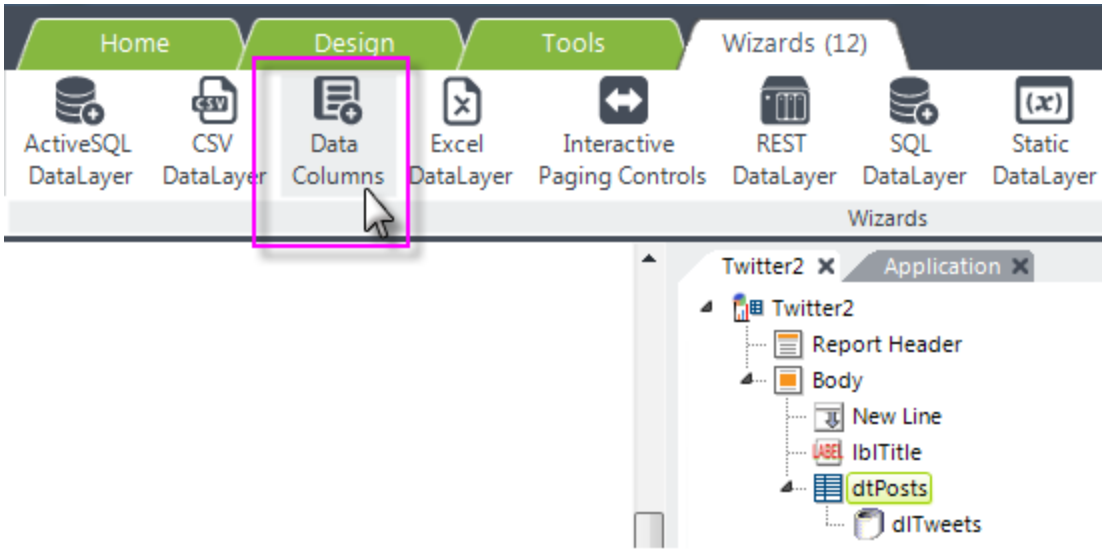
Once the connection has been configured, interaction with Twitter can occur. First, let's examine how posts can be located and retrieved for use in a Logi application.



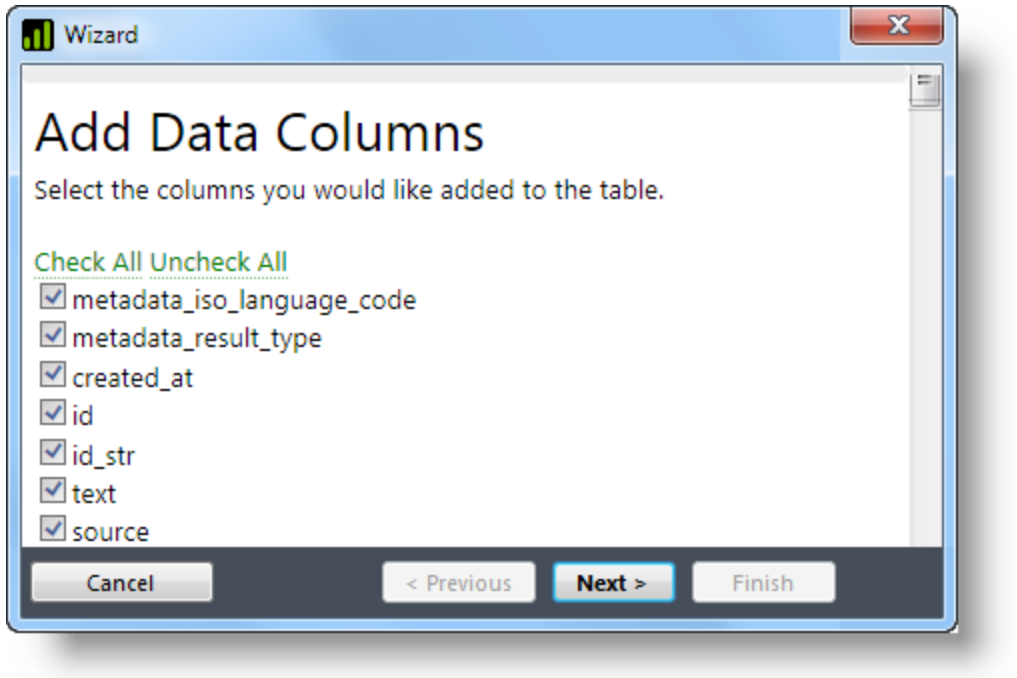
In the example above, a typical report definition is shown, including a Data Table. A **DataLayer.Twitter** element has been added beneath the table. Its attributes are configured as follows:

1. **Connection ID** - (Required) The ID of the Connection.Twitter element configured in the _Settings definition.
2. **Search Type** - (Required) This attribute controls which subset of data will be searched. The related pull-down list of options includes: *User's Statuses*, *Friend's Statuses*, *User's Inbox*, *User's Sent Items*, and *Keyword Search*. Keyword Search searches through statuses only, but for *all* users on Twitter, not just the account specified in the connection element.
3. **ID** - A unique element ID.
4. **Keyword** - The word or phrase to be used when Keyword Search has been selected as the Search Type. This value can be parameterized by using a token here, such as `@Request.Keyword~`. The valid formats and operators that can be used in the keyword are documented in the Twitter *Search Tweets API* document.
5. **Maximum Rows** - This attribute limits the number of rows in the result set returned to the datalayer. If left blank, defaults to 100 rows. Any value between 1-100 may be used, but above 100, the values must be in increments of 100.

The number and names of the columns returned into the datalayer will vary depending on the Search Type selected. The API documentation mentioned earlier provides detailed information about this.



You can, of course, add columns to the Data Table manually, but this is an excellent opportunity to use the **Add Data Columns Wizard**. To do so, select the Data Table element and click the main menu Wizards tab. Then click the Data Columns icon, as shown above.



As shown above, the wizard will connect to Twitter and then display a complete list of the result columns available for the specified Search Type, and you'll be able to exclude those you're not interested in seeing.

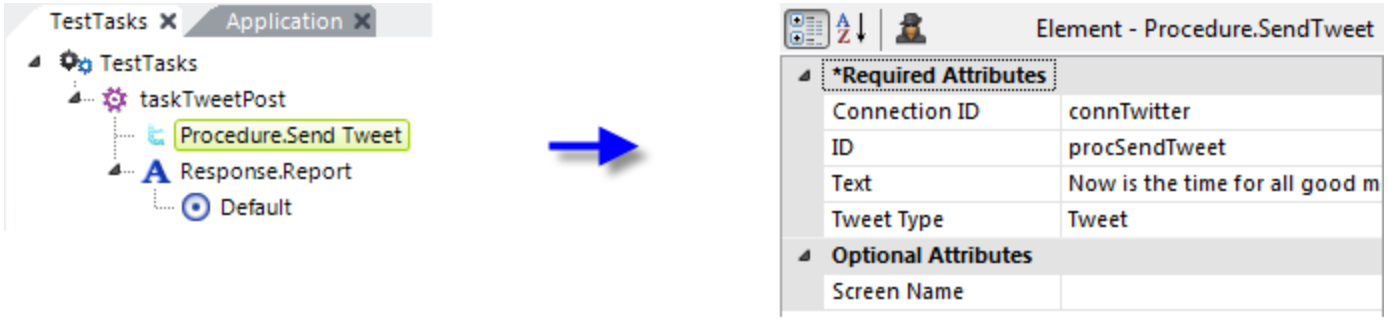
Created	Tweet	Source	User	Followers
11/4/2014	RT @LogiAnalytics: Logi ranked #1 for customer experience, product quality & support in the 2014 Gartner BI Vendor Customer Survey: http://...	RoundTeam	Penny Crown Limited	52
11/4/2014	Logi ranked #1 for customer experience, product quality & support in the 2014 Gartner BI Vendor Customer Survey: http://t.co/9HkxHEjirm	Hootsuite	Logi Analytics	4525
11/4/2014	Has #bigdata gone mainstream? @NVPBigData's survey found 67% of execs have big data initiatives in production: http://t.co/JGnIFV3gFM	Hootsuite	Logi Analytics	4525

The example above shows five of the 30+ columns returned for the Search Type "KeywordSearch".

Once the data has been retrieved into the datalayer, you are then free to manipulate it in any of the usual ways, such as applying filters, conditions, aggregations, etc. or even joining it with other data.

Posting Tweets and Messages

Posting *to* Twitter involves the use of a **Task** in a Process definition:



As shown above, a **Procedure.SendTweet** element is added beneath a **Task** element in the Process definition. Its attributes are set as follows:

1. **Connection ID** - (Required) The ID of the Connection.Twitter element configured in the _Settings definition.
2. **ID** - (Required) A unique element ID.
3. **Text** - (Required) The actual text to be posted; limited to 140 characters. This value can be parameterized using a token, such as `@Request.myText~`.
4. **Tweet Type** - (Required) The type of post: either *Tweet* or *Direct Message*.
5. **Screen Name** - The name of the recipient. This attribute is unnecessary when the Tweet Type attribute is set to *Tweet*, but is required when it's set to *Direct Message*. This value can be parameterized using a token, such as `@Request.ScreenName~`.

Running this task will cause the message to be posted directly to Twitter.com.

The **If Error** element can be used beneath Procedure.SendTweet to capture error messages and send them as a request parameter to a report definition for display.

Glossary

A

API

API, short for Application Program Interface, is a set of routines, protocols, and tools for building software applications. In business intelligence, APIs may be used to enable end-users to directly update source systems.

Authentication

Authentication is the verification of a user's identity.

Authorization

After a user's identity has been authenticated, authorization grants or denies access to reports, columns, and records to selected users or user-groups.

B

Big Data

Refers to both the ever-growing volumes of data in use today and also to services that are specifically engineered to provide and manipulate very large data volumes.

Business Analytics

Business analytics, or business intelligence (BI), gives customers the ability to rapidly create scalable, interactive data analysis applications, and self-service capabilities users can access from anywhere and on any device.

C

Columnar Data Store

Columnar data store is a type of big data repository containing structured data in columns and rows. The main benefits are that the data can be highly compressed and is easily searchable.

CRM

A Customer Relationship Management (CRM) system is a database-based system that records a company’s daily customer-related transactions. CRMs can help customer representatives to provide better service, close more deals, and increase revenue.

CSS

Cascading Style Sheets (CSS) is a technology that allows the presentation aspects of web pages to be separated from the page content. It can be used to add "styling" (e.g. apply fonts, colors, alignment, spacing, and more) to web pages.

D

Data Discovery

Data discovery is the capability to analyze data on-the-fly and uncover insights from it.

Data Enrichment

Data enrichment is a method of preparing data to make it ready for analysis and exploitation, and can include formatting, adding calculations, joining with other data, and more.

DevNet

The Logi Developer Network website.

Drill Down

Drill Down is a capability that allows the user to get a view of the underlying or supporting data used in an analysis.

Drill Through

Drill Through is similar to Drill Down but takes it one step further by applying analysis to the underlying or supporting data.

E

Elemental Development

A development approach used in Logi Info that lets developers build feature-rich applications by using reusable, pre-built elements, rather than by writing low-level code.

F

Forecasting

A technique involving data mining and analysis leading to predictions about what will happen in the future.

G

Geo Mapping

The combination of geographic and other data to produce map visualizations, such as Google or Leaflet maps.

H

Heatmap

A Heatmap chart, sometimes called a "tree map", which uses a unique arrangement of rectangles to represent data and relationships, using color and size.

I

Interpolation

The process of evaluating a literal value match containing one or more placeholders, yielding a result in which the placeholders are replaced with their corresponding values.

J

JavaScript

JavaScript is a programming language supported by the majority of modern web browsers and used by many websites.

JDBC

Java Database Connectivity (JDBC) is an API used to access relational databases. Open Database Connectivity (ODBC) is a similar API designed for use with Java.

JSON

JavaScript Object Notation (JSON) is a lightweight data-interchange format that's easy for humans to read and write, and easy for computers to parse and generate.

K

KPI

Key Performance Indicators (KPIs) are visual indicators, in the form of color-coded shapes, which are tied to a pre-defined, critical threshold.

L

LDAP

The Lightweight Directory Access Protocol (LDAP) is an Internet protocol applications use to look up information from a server and is frequently used for containing user login information.

M

My Term

My definition

N

NoSQL

"Not only SQL" (NoSQL) is an alternative to traditional relational databases, and doesn't rely on tables and a pre-determined schema. NoSQL databases are especially useful for working with large sets of distributed data.

O

ODBC

Open Database Connectivity (ODBC) is an API used to access relational databases. Java Database Connectivity (JDBC) is a similar API designed for use with Java.

OLAP

Online Analytical Processing (OLAP) is the process of analyzing data stored in multi-dimensional "cubes".

R

REST

Representational State Transfer (REST) is a type of API used to provide interoperability between computer systems on the Internet.

S

SSM

The Self-Service Module (SSM) is a package that includes Logi Info + SSRM + Discovery or Logi Platform Services.

SSRM

The Self-Service Reporting Module (SSRM) is a Logi Info add-on module that adds special elements to Info and includes the InfoGo application.

W

Write-Back

The ability to update data sources, typically by adding, editing, or deleting data.