ETLC_ApexBridge
Developer Guide

Table Of Contents

This document is the developer guide explaining all the different configuration options and methods available for you to use the ETLC_ApexBridge design pattern and companion library in your Lightning Components projects.

Note: This document assumes that you have read the blog article[1], and watched the companion presentation that explains the examples posted in the Github repository[2].

Client-Side: ETLC_ApexBridge Component

One of the first steps in implementing the ETLC_ApexBridge library in your components is to nest the `<c:ETLC_ApexBidge />` component inside your own component like this:

Sample Usage In Markup

```
1.   <aura:component>
2.     <c:ETLC_ApexBridge aura:id="ApexBridge" />
3.   </aura:component>
```

ETLC_ApexBridge's Attributes:

When you nest the ETLC_ApexBridge component into your own component, you can use these attributes to configure it further:

Attribute: debugClient

Value	Description
Name	debugClient
Data Type	Boolean
Default Value	False
Description	Allows you to see client-side logging messages generated in JavaScript using the `console.log()`; command

Attribute: debugServer

Value	Description
Name	debugServer
Data Type	Boolean
Default Value	False
Description	Allows you to see server-side logging messages generated in Apex using the `System.debug();` command.

[1] http://eltoro.it/ETLC_ApexBridge
[2] https://github.com/eltoroit/ETLC_ApexBridge

Attribute: forceRefresh

Value	Description
Name	forceRefresh
Data Type	Boolean
Default Value	True
Description	If true, the data will not be cached and a fresh copy will always be used. You can also override this for each request.

The way your controller will interact with ETLC_ApexBridge component is by invoking either of the two public methods: `callApex()` or `makeRecords()`.

ETLC_ApexBridge's Methods:

Method: callApex()

This is the main method of the ETLC_ApexBridge, because it's the one that makes the call to Apex. The logic in this method will also perform a preliminary check on the response before it's sent back to the invoking method.

Your code should look like this:

Sample Invocation From JavaScript

```
var apexBridge = component.find("ETLC_ApexBridge");
apexBridge.callApex({
   component: component,
   request: {
      controller: "ApexClassName",
      method: "methodName",
      input: {
         var1: value1,
         var2: value2,
         varTimestamp: new Date(valueDateTime).toJSON(),
         ...
      },
      records: [record1, record2, ...],
      doesCallout: false
   },
   forceRefresh: false,
   pleaseWait: {
      type: "None|Event|Toast|Full",
      message: "Please Wait..."
   },
   callBackMethod: function (serverResponse) {
      console.log(serverResponse.output);
   },
   errorHandler: function (serverResponse) {
      console.error(serverResponse);
   }
});
```

Sample Invocation Explained

Let's talk about each of the pieces of this code:

Line Number	Description
1	Finds the nested component. Assuming the ETLC_ApexBridge component was nested in your component with the property aura:id set to this value "ETLC_ApexBridge".
3	Your component that nested the ETLC_ApexBridge library.
4-15	Defines the request that you want to make to your server-side controller
5	Sets the name of the Apex class that will handle your request. This class must implement the ETLC_ApexBridge_Abstract class.
6	Specifies which methos in Apex will handle your request.
7-12	Specify the non-sObject data that you pass to the Apex controller,
13	Specify the list of sObject data that you pass to the Apex controller. This is a list of sObjects records (could be a list with only 1 record).
14	If your controller method is going to perform any web service callout, then this must be set to true.
16	If you want to ensure a fresh copy of the data, set this to true. But if you prefer speed, bytrying to get data from cache, set this to false.
17-20	Allows you to control the "Please Wait" message that you get while the server-side controller is busy.
18	Which type of "please wait" display do you prefer? The different options are: • **None**: No indication at all. With this option there is no message displayed, nor an application event raised. • **Event**: There is no message displayed, but an application event will be fired which you can handle and display your own messages or take any other action needed. • **Toast**: Displays a toast message, which is a small drop-down from the top of the page with the message displayed. • **Full**: The default value, which displays a modal pop-up window with the message.
19	You can customize the text message that is displayed. By default it will be a funny line.
21-23	Define the call back function that will handle your request if there were no errors when making the request
24-26	Define the call back function that will handle the errors found while executing your request. This is optional, and if you do not provide a callback function then ETLC_ApexBridge will throw a JavaScript exception.

Method: makeRecords()

When you want to insert records in Salesforce database you have two options:

The first option is pass the values for each field from the JavaScript controller and have Apex assemble the record before it's inserted. In Apex you would so something like this:

```
1.   Account a = new Account();
2.   a.Name = <Retrive value using data passed from JavaScript>;
3.   insert a;
```

The second option is to assemble the record in the client-side controller and pass that record, as a single unit, to Apex to be inserted. In this later case, the Apex server-side controller would look something like this:

```
1.   Account a = <Retrive record using data passed from JavaScript>;
2.   insert a;
```

If you want to use the second option, you must pass a properly formatted record created in JavaScript. You can use the `makeRecords()` method to create the data with the required format, and send that to the server-side Apex controller.

Note:
If you want to send some fields to the server side controller, and the record structure will be created in Apex, then this method is not required. It's only required if you want to send new records whose structure have been created in the client-side controllers.

Also, if you have retrieved one or more records from Apex, to either update or delete, for example, then those records already have the desired format and you do not need this method either.

Sample Invocation From JavaScript

```
1.   var apexBridge = component.find("ETLC_ApexBridge");
2.   apexBridge.makeRecords({
3.      sObjectType: "Account",
4.      records: [
5.         {
6.            Name: "Acme Inc. #1",
7.            Description: "Account created in JavaScript"
8.         },
9.         {
10.           Name: "Acme Inc. #2",
11.           Description: "Another account created in JavaScript"
12.        }
13.     ],
14.     callBackMethod: function(records) {
15.        // Send records to Apex.
16.     }
17.  });
```

Sample Invocation Explained

Let's talk about each of the pieces of this code:

Line Number	Description
1	Similarly to previous methods, this finds the nested component.
2-17	Defines the parameters for the records to be created
3	The API name of the records to be created.
4-13	A JSON list of objects, for each record define the field and the value to be set in the new records. Note: if you only need one record, create a list with only one record.
14-16	The callback function that will process the records that were created. You could invoke the `callApex()` method to send the records to the controller.

Client-Side: ETLC_PleaseWait Component

As described in the `callApex()` method, there is a "Please Wait" message that gets displayed while the call to Apex is being processed.

If you specify the `type` as "None" there will be no message and if you set the `type` to "Event" you will have to handle this yourself… but if you do not change the default message or set the `type` to "Full" or "Toast" the `ETLC_PleaseWait` component will display such message.

Although `ETLC_ApexBridge` manages this component by properly setting the properties and invoking the methods as needed, you could also use this component whenever you need a "Please Wait" message, so let me walk you through the public attributes and methods available

ETLC_PleaseWait's Attributes:

Attribute: showFullMessage

Value	Description
Name	showFullMessage
Data Type	Boolean
Default Value	true
Description	When true, the component will display a modal pop-up message. If false, it will display a toast message

Attribute: useDefaultMessage

Value	Description
Name	useDefaultMessage
Data Type	Boolean
Default Value	True
Description	When true the messages will display a random funny message, if you want to customize the message set this to false.

Attribute: customMessage

Value	Description
Name	customMessage
Data Type	String
Default Value	null
Description	If you want to use a custom message, you can set it in this attribute with that value.

ETLC_PleaseWait's Methods:

Method: showMessage()

Value	Description
Name	showMessage
Description	Call this method to show/hide the message.

Sample Invocation From JavaScript

```
1.   var isShow = true;
2.   var cmp = component.find("ETLC_PleaseWait");
3.   if (isShow) {
4.      cmp.set("v.showFullMessage", true);
5.      cmp.set("v.useDefaultMessage", false);
6.      cmp.set("v.customMessage", "This is the new message");
7.   }
8.   cmp.showMessage(true);
```

Sample Invocation Explained

Let's talk about each of the pieces of this code:

Line Number	Description
2	Similarly to previous methods, this finds the nested component.
3-7	These attributes should be set only when displaying the message, not necessarily when hiding it.
4	True will display a modal pop-up message. False will display a toast message
5	Do you want a custom messge?
6	Set the custom message to be displayed
8	Show (true) or hides (false) the message.

Server-Side: ETLC_ApexBridge_Request Class

This class represents the request made by JavaScript, and it's also the way Apex sends the data back to JavaScript. It's therefore a very important class and you will use it very often.

Apex Properties

Although the instances of this class will have properties that control the request, some of these properties are managed either internally (by the library), or JavaScript (by your client-side controller):

Property: controller

Value	Description
Name	controller
Data Type	String
Managed By	JavaScript
Description	Name of the class that handles this request.

Property: useDefaultMessage

Value	Description
Name	method
Data Type	String
Managed By	JavaScript
Description	Name of the method that handles this request.

Property: input

Value	Description
Name	input
Data Type	String
Managed By	JavaScript
Description	JSON string passed from JavaScript. Do not use this directly, but go though the getInput*() methods

Property: output

Value	Description
Name	output
Data Type	String
Managed By	Internal
Description	JSON string passed back to JavaScript. This property is manged internally and it's populated with the data returned in your server-side controller[3]

Property: records

Value	Description
Name	records
Data Type	List<SObject>

[3] As described in the companion materials

Managed By	JavaScript
Description	List of records (List<SObject>) passed from JavaScript

Property: debug

Value	Description
Name	debug
Data Type	Boolean
Managed By	JavaScript
Description	Helps you determine if the client is expecting to see debug statements in the server logs. May want to check this attribute before writing debug logs using System.debug();

Property: isSuccess

Value	Description
Name	isSuccess
Data Type	Boolean
Managed By	Internal
Description	Do not modify this property. The library will assign this value when there has been errors found.

Property: messages

Value	Description
Name	messages
Data Type	Map
Managed By	Internal
Description	This map (Map<MessageType, List<String>>) contains the messages passed to your client-side controller. The messages are grouped in 3 categories: Errors, Warnings, Information. You will add messages when needed using the method addMessage().

Property: doesCallout

Value	Description
Name	doesCallout
Data Type	Boolean
Managed By	JavaScript
Description	Does this request make a callout?[4]

[4] Internaly, when the library receives the request from JavaScript, it will first make a SavePoint so that the data processed by the request can be rolled back and the exceptions can be proplery managed. Creating this SavePoint will prevent you from making a HTTP request.

Apex Methods

Method: addMessage()

Value	Description
Name	addMessage
Input	
Output	
Description	This method adds the message to the collection of messages, based on the type (Errors, Warnings, Information). If the type of message is Error, the property isSuccess is set to false.

Method: getInputValue()

Value	Description
Name	getInputValue
Input	String: Key for the input value you want to obtain
Output	Object: Value obtained from input parameters

Method: getInputDate()

Value	Description
Name	getInputDate
Input	String: Key for the input value you want to obtain
Output	Date: Value obtained from input parameters

Method: getInputDateTime()

Value	Description
Name	getInputDateTime
Input	String: Key for the input value you want to obtain
Output	DateTime: Value obtained from input parameters

Method: dateToJSON()

Value	Description
Name	dateToJSON
Input	Date: Value to convert
Output	String: String representing value converted

Method: dateTimeToJSON()

Value	Description
Name	dateTimeToJSON
Input	DateTime: Value to convert
Output	String: String representing value converted

As you can see from the last few methods listed, this class allows you to receive and send date and date/time input values. The date/time information in JavaScript can be a bit tricky because of the different locales and time zones. This library has taken into account those issues, so you do not have to worry about the details. The only thing you need to do when sending date or date/time information from your JavaScript client-side controller to the Apex server-side controller is to pass the data like this:

Passing Date/Time From JavaScript

```
1.   input: {
2.    variableName: new Date(component.get("v.demoWhen")).toJSON()
3.   }
```

Server-Side: ETLC_SecuredDB Class

This class is part of a GitHub repository I had posted previously but I have included in here because it's **very** relevant while developing code on the Lightning Components Frameworks because of the unsecure nature of Apex server-side controllers.

Before going any further, let's very quickly review the data security model in Salesforce. As you may already know, Salesforce security model is based on three equally important pieces: CRUD (CRED), FLS, Record Access. The first two can be controlled via Profiles and Permission Sets. The third part is a bit more complex and is controlled via things like: Role Hierarchy, Sharing Rules, OWD, etc.

Apex runs in what is called "System Mode" meaning it will ignore any restrictions imposed by the security model and therefore will assume full CRUD, FLS and Record Access. You can control the record Access in your classes by ensuring they are written as "With Sharing" but this will still full CRED and FLS.

These blog articles help younunderstand the For more information on the security model[5], and how to write secure Apex code[6].

If you have worked with Visualforce, you may be familiar with the fact that it protects you from showing sensitive data when you work with tags like `<apex:inputField>` or `<apex:outputField>`. Unfortunately, that is not the case when working with Lightning Components and you must ensure that the security will not be violated when your data is accessed. You could write secure code to ensure this, by writing code that uses Dynamic Apex to check for the user's permissions (see previous blog article for more information). These same techniques are used in this Apex class, so you do not have to worry about the details.

---

[5] http://eltoro.it/SalesforceSecurityWhatEveryDeveloperMustKnow
[6] https://eltoro.it/DynamicApexToCheckCrudAndFls

This class will validate the user has the required permissions in the objects and fields involved before attempting submitting the request (query or DML) to the database. This code has three main modes:

Validate Security When Querying Data

You can use this class to validate if the user is able to fetch the records/fields by providing a SOQL query to the `query()` method or, a you can pass either a single record (or a list of records) to the `validateQuery()` method. When you provide the SOQL, the system will perform the query from the database.

Validate Security When Performing a DML operation

You can also use this class to validate if the user is able to perform a DML operation on the records/fields by providing a record (or list of records) to the `performDML()` method. When invoking this method, you must indicate the operation desired (using the enum variable `ETLC_SecuredDB.Operation`).

Just Validating Security

This class inspects the data to find the objects and fields involved before validating the access the user has on the data. This may be a slow operation, especially if your database action (query/DML) has a big dataset with may records, many fields, and in the case of the queries many related records.

In order to speed things up, and not have the library discover all the objects/fields in the database operation (query/DML), you may optionally handle the database requests yourself and use this class just to validate your data. There is no reason to always find the objects/fields at runtime if the data structure is not changing ;-)

In order to take advantage of not having to validate the data structure every time you access you would want to get the field structure as a string (`getFieldsForPlainValidator`) and pass that structure to the validator (`plainValidator`), indicating the desired operation (`ETLC_SecuredDB.Operation`).

Apex Properties

Before I explain the different methods in this class, let's talk about the properties:

Property: findFieldsUsingJSON

Value	Description
Name	findFieldsUsingJSON
Data Type	Boolean
Description	This attribute was created to test performance on discovering the fields. You may want to experiment with this property. For more information, you can check this GitHub issue

Property: findFieldsUsingJSON

Value	Description
Name	showDebugMessages
Data Type	Boolean
Description	If you need to debug this class, set this flag to true and Apex debug messages will be printed in the debug log.

Property: Operation

Value	Description
Name	showDebugMessages
Data Type	Enum
Description	This is one of the most important attributes in this class. In controls the operation that you want to perform. The operations available are: • Querying • Inserting • Updating • Deleting • Upserting

Apex Methods

Let's take a look at the different methods available in this class…

Method: query()

Value	Description
Name	query
Input	SOQL (String): This is the SOQL query to be executed.
Output	List<sObject> or an exception if the user does not have the right permissions.
Description	Performs a SOQL query and returns the results, but if the user executing the query does not have right permissions (CRUD/FLS) an exception is thrown.

Method: validateQuery() for a single record

Value	Description
Name	validateQuery
Input	dbRecord (sObject). The record you want to validate the user's permissions.
Output	Nothing or an exception if the user does not have permissions on this record.
Description	Allows you to validate the permission for the user after you have queried a SOQL query.

Method: validateQuery() for multiple records

Value	Description
Name	validateQuery
Input	dbRecords (List<sObject>). The list of records you want to validate the user's permissions.
Output	Nothing or an exception if the user does not have permissions on this record.
Description	Allows you to validate the permission for the user after you have queried a SOQL query.

Method: performDML() for a single record

Value	Description
Name	performDML
Input	• Op (Operation): The desired DML operation to perform. • dbRecord (sObject): The record to perform the DML operation on.
Output	Nothing or an exception if the user does not have permissions on this record.
Description	Performs the desired DML operation, but if the user executing the operation does not have right permissions (CRUD/FLS) an exception is thrown. Note: For upserts, you must pass a list of records.

Method: performDML() for multiple records

Value	Description
Name	performDML
Input	• Op (Operation): The desired DML operation to perform. • dbRecords (List<sObject>): The records to perform the DML operation on.
Output	Nothing or an exception if the user does not have permissions on this record.
Description	Performs the desired DML operation, but if the user executing the operation does not have right permissions (CRUD/FLS) an exception is thrown. Note: For upserts, you must pass a list of records.

Method: getFieldsForPlainValidator() for a single record

Value	Description
Name	getFieldsForPlainValidator
Input	dbRecord (sObject): The record that should be discovered for fields.
Output	String: A JSON serialization of a Map representing the objects and the fields based on the input.
Description	The previous methods (query and performDML) use a discovery method to find the objects/fields that are being checked. This can be a slow process and could be avoided if you already have the information required.

Method: getFieldsForPlainValidator() for multiple records

Value	Description
Name	getFieldsForPlainValidator
Input	dbRecords (List<sObject>): The records that should be discovered for fields.
Output	String: A JSON serialization of a Map representing the objects and the fields based on the input.
Description	The previous methods (query and performDML) use a discovery method to find the objects/fields that are being checked. This can be a slow process and could be avoided if you already have the information required.

Method: plainValidator() using JSON Data

Value	Description
Name	plainValidator
Input	• Op (Operation): The desired operation that should be checked for. • fieldsJSON (String): A JSON serialization of a Map representing the objects and the fields to be checked.
Output	None or an exception if the user does not have the right permissions.
Description	This method is used by the previous methods (query and performDML) after the fields have been discovered to validate the security.

Method: plainValidator() using a single object

Value	Description
Name	plainValidator
Input	• Op (Operation): The desired operation that should be checked for. • sObjName (String): Object containing the fields to be checked. • fieldNames (Set<String>): Names for the fields belonging to single object to be checked.
Output	None or an exception if the user does not have the right permissions.
Description	This method is used by the previous methods (query and performDML) after the fields have been discovered to validate the security.

Method: plainValidator() using a pap of fields

Value	Description
Name	plainValidator
Input	• Op (Operation): The desired operation that should be checked for. • mapFieldNames (Map<String, Set<String>>): Map of sObject names and the fields for each sObject that to be checked.
Output	None or an exception if the user does not have the right permissions.
Description	This method is used by the previous methods (query and performDML) after the fields have been discovered to validate the security.