**Introducing Integration Procedures**

So what exactly are OmniStudio Integration Procedures? They are applications used to read and write data from Salesforce and from external systems. An Integration Procedure can be called from an OmniStudio component such as an OmniScript or FlexCard, an API, or even from an Apex method.

Think of it this way: When you need to access and transform data from third-party sources and no user interaction is required, and when moving the workload from client to server is desirable, then that's when you want to use an OmniStudio Integration Procedure.

OmniStudio Integration Procedures are declarative, server-side processes that execute multiple actions in a single server call. Let’s break this down to better understand what it means.

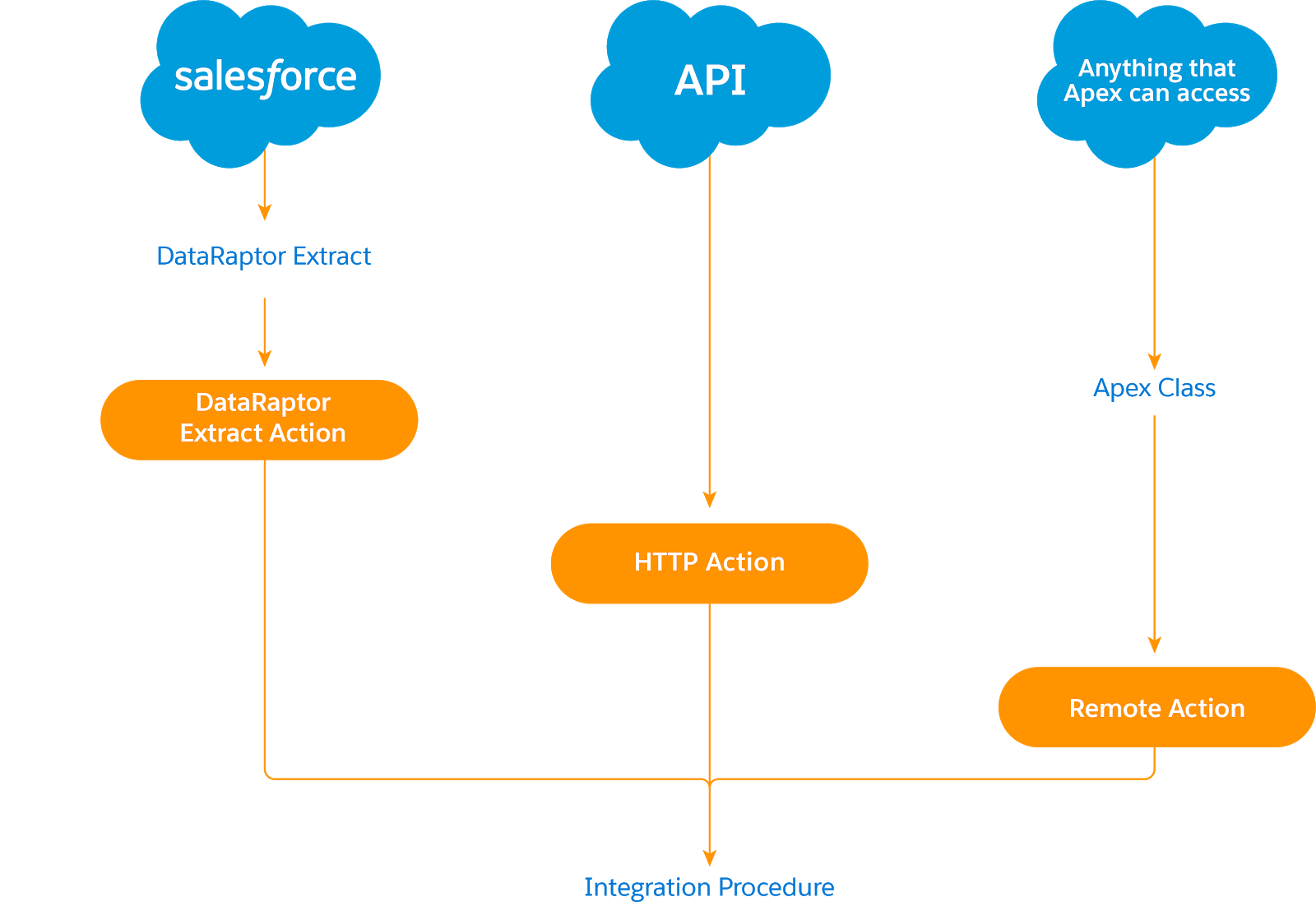
* **Declarative:**Use drag-and-drop elements in the Integration Procedure Designer to build the process structure.
* **Server-side processing:** This enables faster performance, because the server, in most cases, is faster than the client at processing data.
* **Multiple actions in a single server call:** This prevents round trips to the server. Minimizing client/server calls is beneficial, as more round trips mean slower performance.

More simply, Integration Procedures are a way to get, save, and manipulate data behind the scenes. They’re especially useful in the following scenarios.

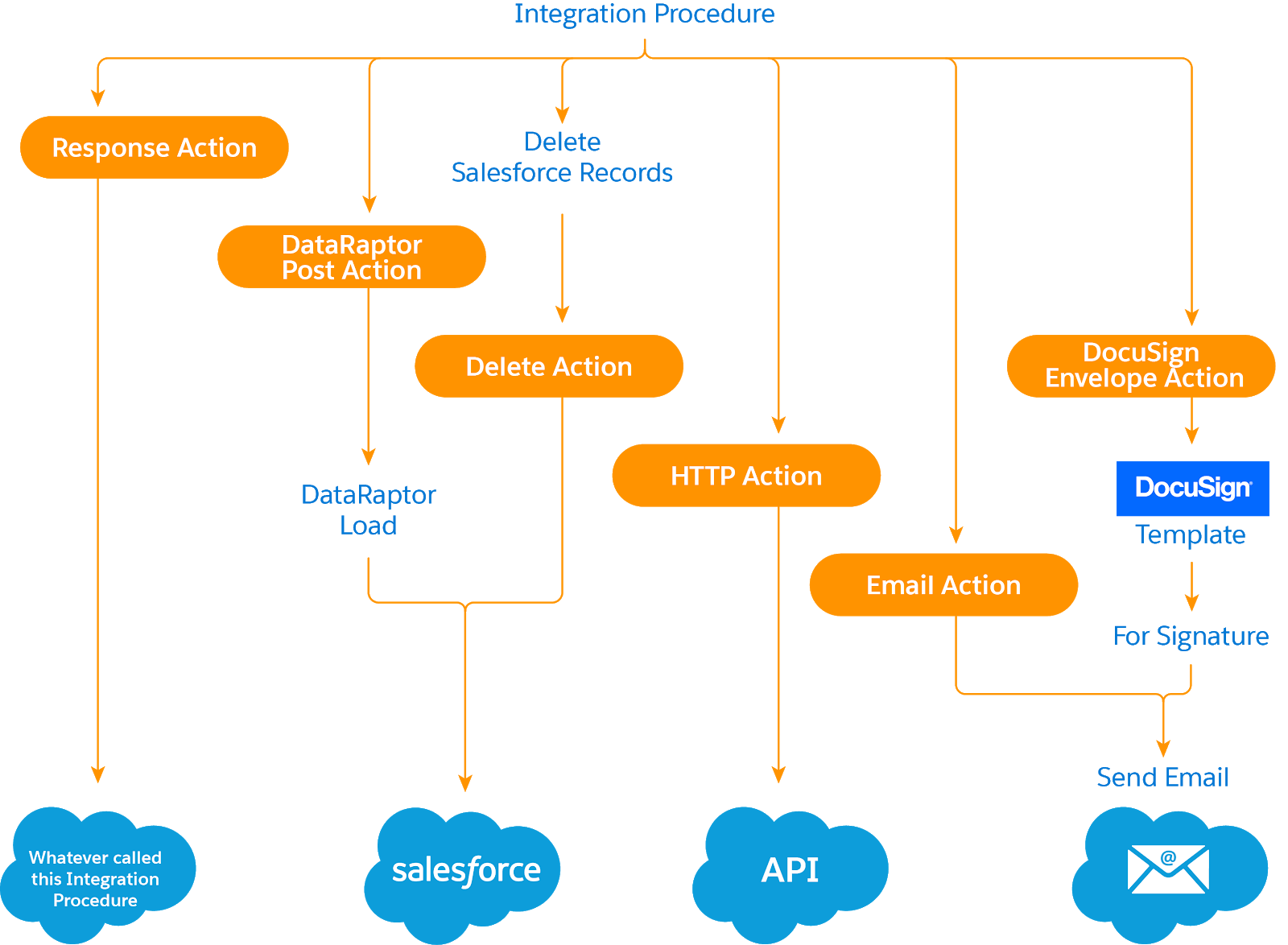
* You need to access and transform data from third-party sources.
* No user interaction is required.
* Moving the workload from client to server is preferable.

Integration Procedures do what they do thanks to a host of awesome capabilities.

**Integration Procedures handle multiple data sources.** They read data from Salesforce, external systems (using representational state transfer [REST]/application programming interface [API] calls), and Apex classes. It looks a little something like this:



**Integration Procedures serve as data sources for multiple technologies.** They are called from and return data to OmniScripts and OmniStudio FlexCards. They also serve as data sources for APIs and Apex code. Check it out:



**Integration Procedures are portable.** This means you build an Integration Procedure once and use it everywhere. You can use the same one in a FlexCard as well as in an OmniScript.

**Integration Procedures only send and receive the data you need.** An often overlooked performance factor is the amount of data being sent between the browser and the server. The Integration Procedure’s Response action allows you to trim the data returning to the browser from the server. This minimizes client/server data transfer, which is a key factor if you’re on a slow network or using a mobile connection.

**Integration Procedures perform batch processing.** They process large amounts of data without causing a Salesforce timeout.

As you can see, Integration Procedures have some fantastic capabilities. Next, let's examine how they make working with data management easier.

**Integration Procedure Advantages**

We recommend using Integration Procedures as much as possible as a data source. Why? Because they have a streamlined structure that’s efficient and consistent. They’re adaptable to use with all your data sources and easy to navigate.

But that’s not all—they give developers a lot more control over the data going back and forth and:

* Provide optimal flexibility.
* Make implementations easier.
* Greatly improve the performance of FlexCards and OmniScripts.

Another huge advantage of using Integration Procedures is that they future-proof your design.

For example, when you design a FlexCard, you need some data from the server. However, you might not yet be entirely sure what that data looks like or how to get it. Here’s how Integration Procedures provide a solution.

* You create your Integration Procedure with sample (mock) data, and call the Integration Procedure from the FlexCard. You move forward in your card design, even if the backend system is not yet ready.
* When that backend system becomes ready, you simply make the changes to the Integration Procedure, and you’re up and running. You no longer have to touch your FlexCard. This approach is a good way to separate your development work on the front end from your development work on the back end.

In addition to these benefits, replacing Apex classes with Integration Procedures has several perks. For example, Integration Procedures:

* Require no Apex test code coverage, which is a time- and cost-saving technical advantage.
* Are much easier to maintain and update.
* Take up to 97% less development time.

Let’s take a closer look at that last one. Here’s a comparison of development times with OmniStudio implementations when using custom Apex classes versus using Integration Procedures.

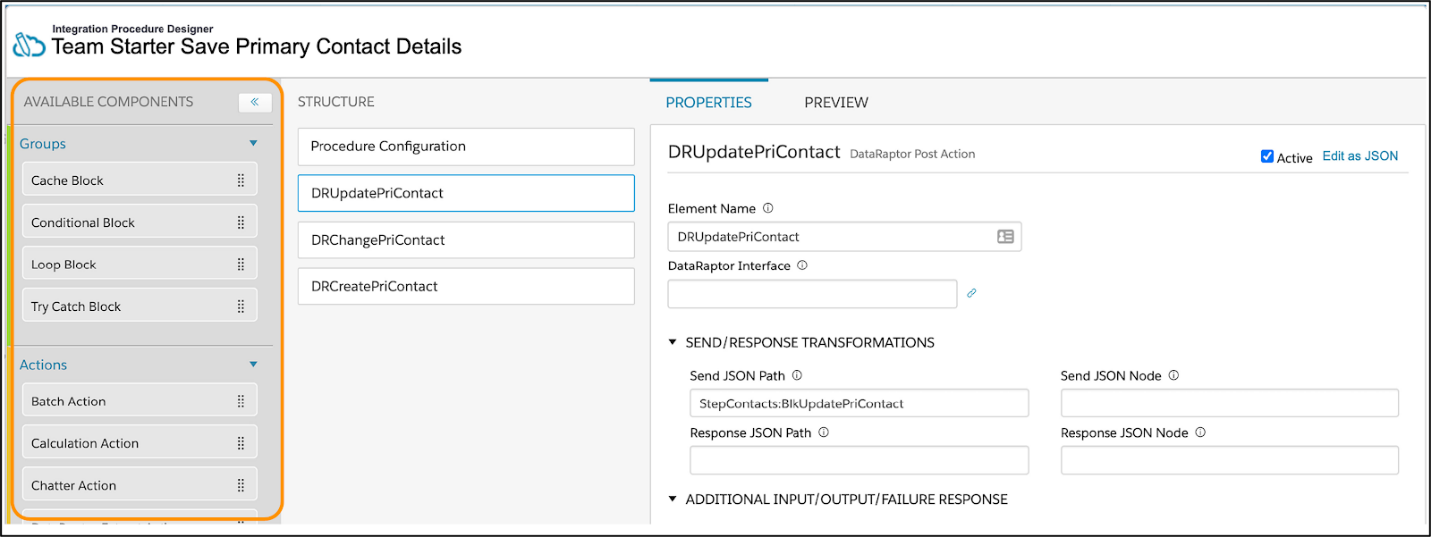
|  |  |  |  |
| --- | --- | --- | --- |
| **Backend Service Complexity** | **Build Time Apex** | **Build Time Integration Procedures** | **% Reduction in Level of Effort and Duration Using Integration Procedures** |
|  |  |  |  |
| Easy | 2 hours | 30 minutes | 75% |
| Difficult | 6 weeks | 1 day | 97% |

## Meet the Integration Procedure Designer

The OmniStudio Integration Procedure Designer allows you to quickly configure your Integration Procedure by dragging different elements into the Structure panel and configuring the properties for that element on the same screen.

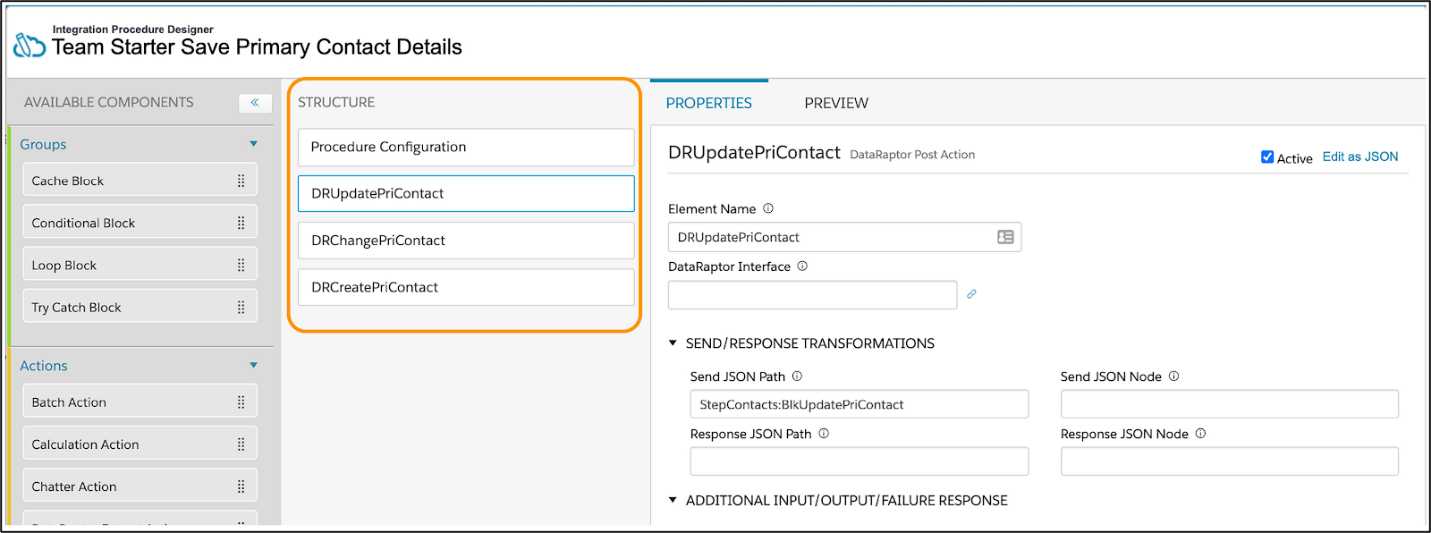
**Available Components Panel**

The Available Components panel contains Groups and Actions elements.



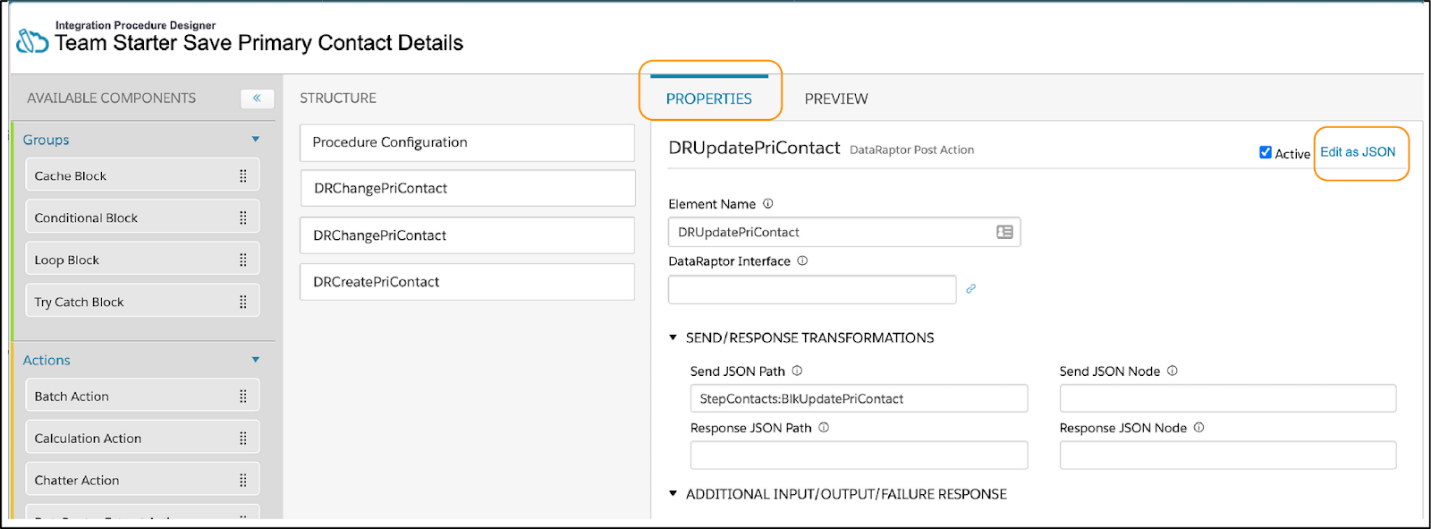
**Structure Panel**

Drag and drop elements from the Available Components panel to the Structure panel.



**Element Properties Pane**

Each element has a Properties pane with a link to edit the JavaScript object notation (JSON) of each element.



**Preview Pane**

The Preview pane allows you to input parameters and execute the Integration Procedure to confirm it has been configured correctly. You can use the Errors/Debug Output to troubleshoot.

**Integration Procedure Elements**

You might be familiar with using elements in OmniScripts and OmniStudio FlexCards. Integration Procedure elements are a little different. They work behind the scenes—there are no elements for user interaction.

Let’s see what they do instead. Integration Procedure elements are either Groups or Actions.

**Groups**

Integration Procedures allow you to group related steps as a unit inside a block. Integration Procedures provide the following block types.

| **What It Is** | **What It Does** | **Examples** |
| --- | --- | --- |
| **Cache Block** | * Saves the output of the steps within it to a session or org cache for quick retrieval * Stores frequently accessed and infrequently updated data, which saves round trips to the database and improves performance * Allows data updates without caching * Allows different cached data to expire at different times | An Integration Procedure caches weekly sales numbers but does not cache stock prices. |
| **Conditional Block** (This is the most basic block type.) | * Executes if a specified condition is true or treats the steps within it as a series of mutually exclusive alternatives * Supports branching workflows, which function like conditional views in an OmniScript | Based on a price and a state code, an Integration Procedure calculates the sales tax and reports the total price. |
| **Loop Block** | * Iterates over the items in a data array, enabling the Actions within it to repeat for each item rather than requiring separate Action elements for each item | In a scenario where a user is adding products to a cart, one Remote Action within a Loop Block can add all four products. Without a Loop Block, adding four products to a cart would require running four separate Remote Actions. |
| **Try-Catch Block** | * Lets you “try” running the steps inside the block and then “catch” the error if a step fails * Returns specified output or calls an Apex class if a step within it fails | An Integration Procedure creates and deletes a Contact with a specified LastName and returns an error message if the LastName field is blank. |

You can nest blocks within other blocks. For example, you can nest a Loop Block within a Try Catch Block or a Cache Block.

All blocks have one property in common—an **Execution Conditional Formula**.

* If this formula evaluates to true or is not defined, the block is executed.
* If it evaluates to false, the block is skipped.

**Actions**

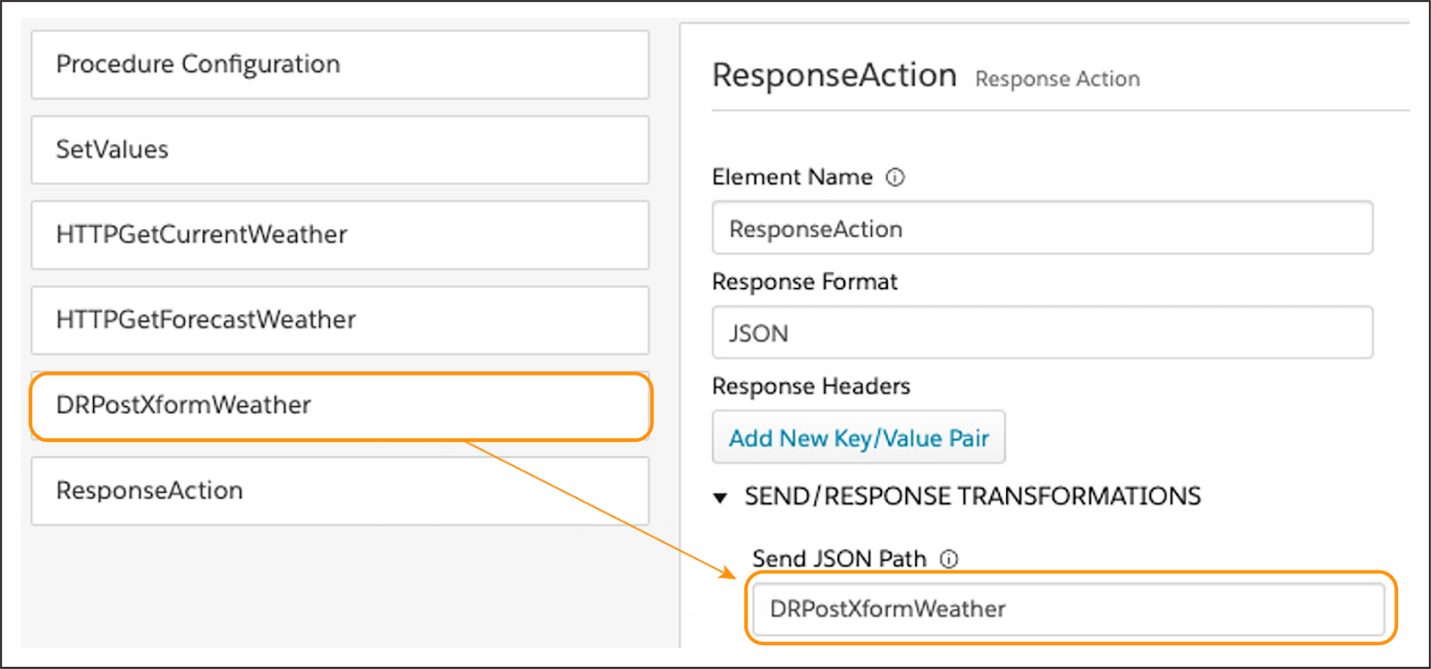
Next, let’s see what Action elements are all about.

Action elements are essential building blocks of every Integration Procedure. These actions can set data values, perform functions, call DataRaptors, invoke Apex classes, send emails, invoke REST endpoints, run other Integration Procedures, and more.

You can also use blocks to group actions for conditional execution, caching, list processing, and error handling. To add an action, drag it from the palette (shown below) into the Integration Procedure Structure panel.

The data source determines which element you use, and you should always use a Response Action at the end if needed. The Response Action element tells the Integration Procedure what data to pass back to the tool that called it.

In the following example, the Response Action limits what is sent back, only sending what is in the DataRaptor Transform Action’s node.



## Data Input and Data Output

**Data Input**

Remember, the data source determines which element you use.

| **For This Data Input Source** | **Use This Integration Procedure Action Element** |
| --- | --- |
| Salesforce Org | DataRaptor Extract Action or DataRaptor Turbo Action |
| API | HTTP Action |
| Anything that Apex can access | Remote Action |

**Data Output**

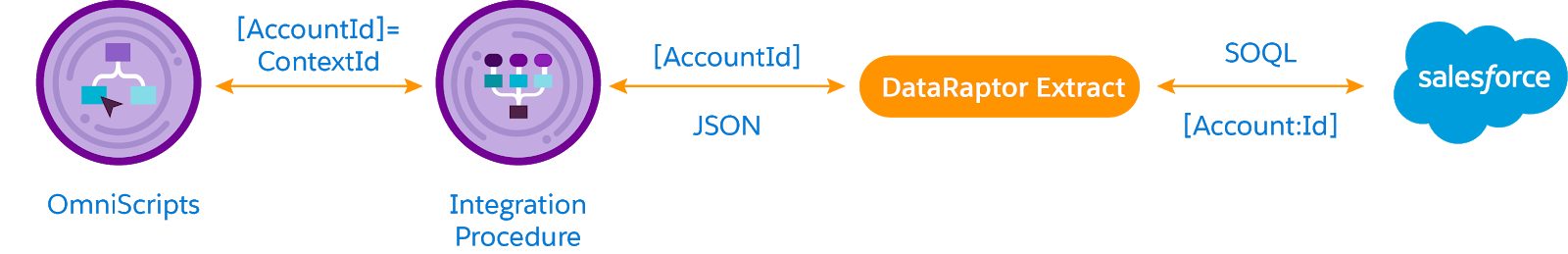
The Response Action element is unique to the Integration Procedure. It passes data back to the tool that launched the procedure. In addition to getting and processing data, you can use an Integration Procedure to output data.

| **For This Data Output Source** | **Use This Integration Procedure Action Element** |
| --- | --- |
| FlexCards or OmniScripts | Response Action |
| Salesforce Org | DataRaptor Post Action (To save data to Salesforce Records) |
| Salesforce Org | Delete Action (To delete data from Salesforce Records ) |
| API | HTTP Action |
| Send Email | Email Action |
| DocuSign Template | DocuSign Envelope Action |

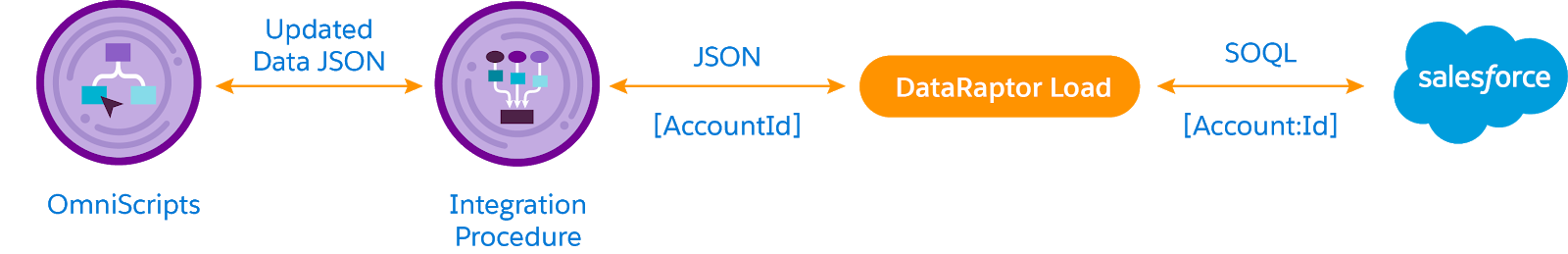
## Where Integration Procedures Fit Within the Data Flow

First, let’s see where an Integration Procedure fits in the data flow for this scenario.

* When the Edit Account OmniScript launches from an action on a FlexCard, it passes the RecordId for the account (called the AccountId) from a JSON node to a DataRaptor Extract.
* The Edit Account OmniScript saves this RecordId in a variable called ContextId. AccountId is an arbitrary name for the RecordId, which means you can call the RecordId anything you like; however, the name must be the same in each of the elements for the data to flow properly between them and Salesforce. It’s best practice to label the ContextId with a description of what kind of ID it is.
* The OmniScript passes a variable called AccountId to an Integration Procedure, which passes the AccountId to a DataRaptor Extract.
* The DataRaptor Extract uses the AccountId in Salesforce Object Query Language (SOQL) to retrieve data from Salesforce. This includes not only the AccountId but also data from the Account record. The DataRaptor Extract sends the data, in JavaScript Object Notation (JSON) format, back to the Integration Procedure and then to the OmniScript.

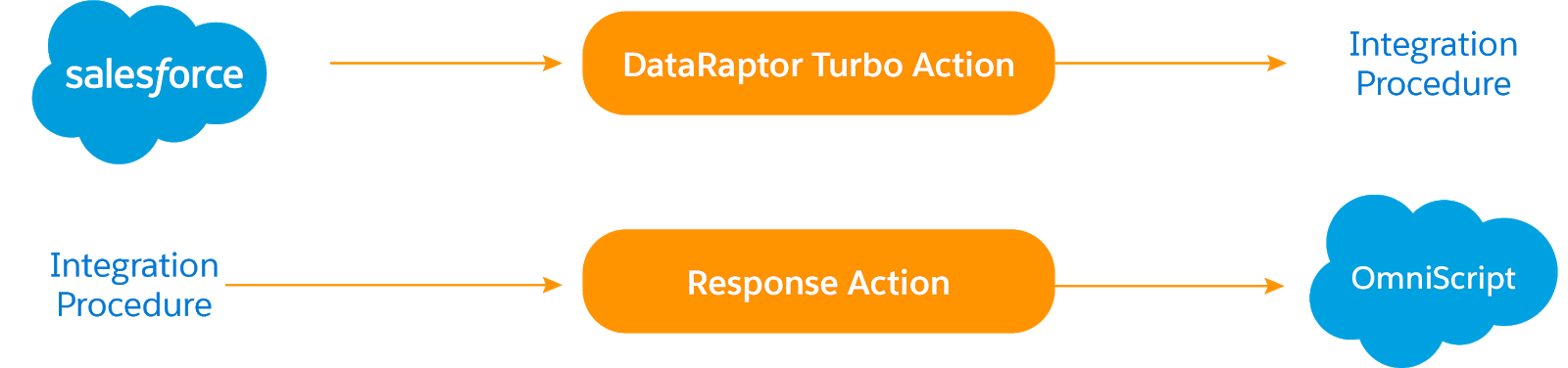


* The user and other OmniScript actions manipulate the data.
* An updated data JSON, including the AccountId, is then passed to an Integration Procedure, which passes it to a DataRaptor Load.
* The DataRaptor Load uses the AccountId to identify the original account record and updates the data in Salesforce.

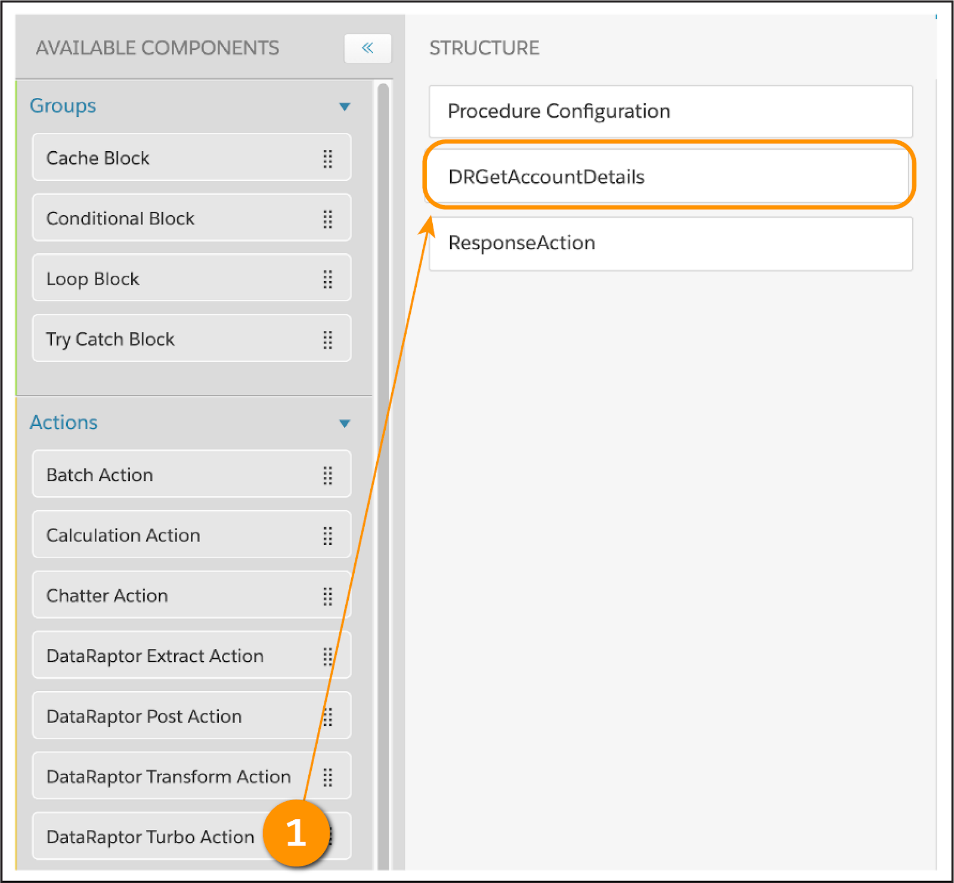


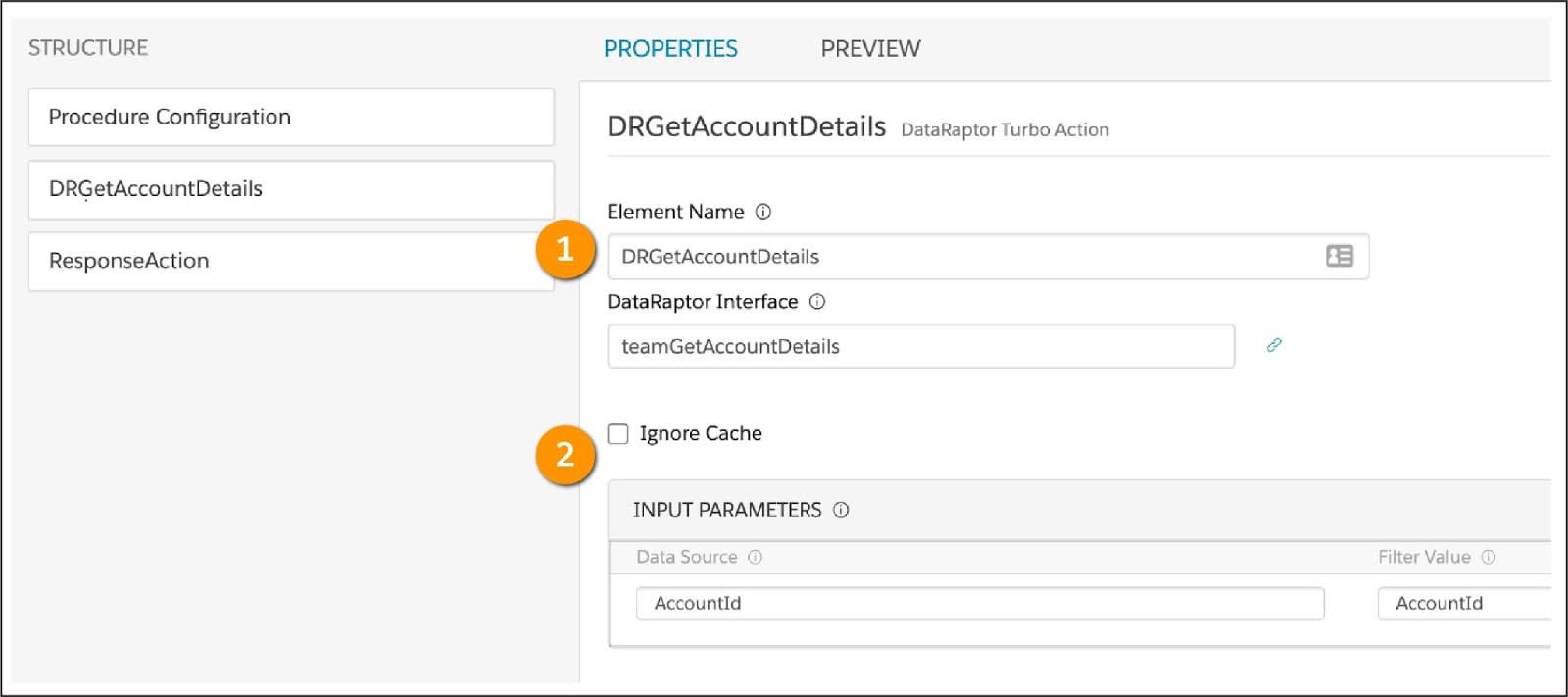
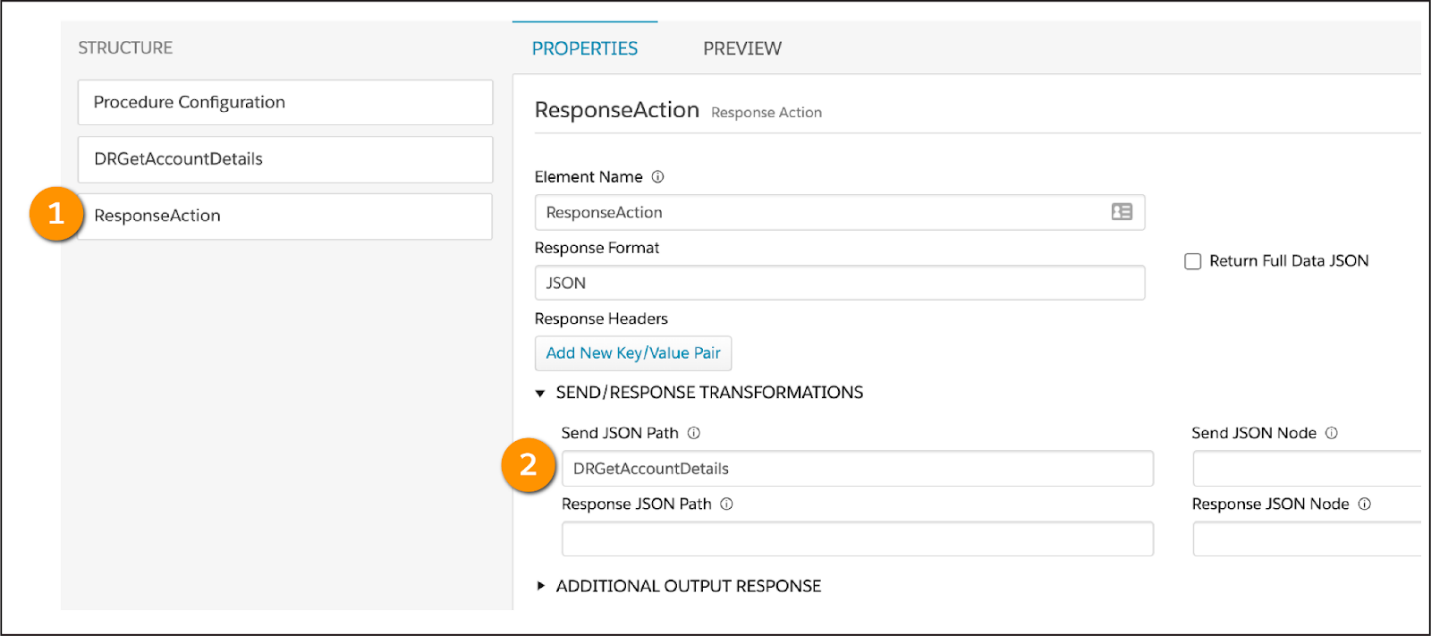
**Build an Integration Procedure to Get Data**

In scenario 1 in the previous unit, you designed the Integration Procedure to get data for the Edit Account OmniScript. You chose the DataRaptor Turbo Extract Action to call a DataRaptor Turbo Extract to get Account data from Salesforce. Then you chose the Response Action to send that data back to the Edit Account OmniScript from the Integration Procedure. That process looks like this.



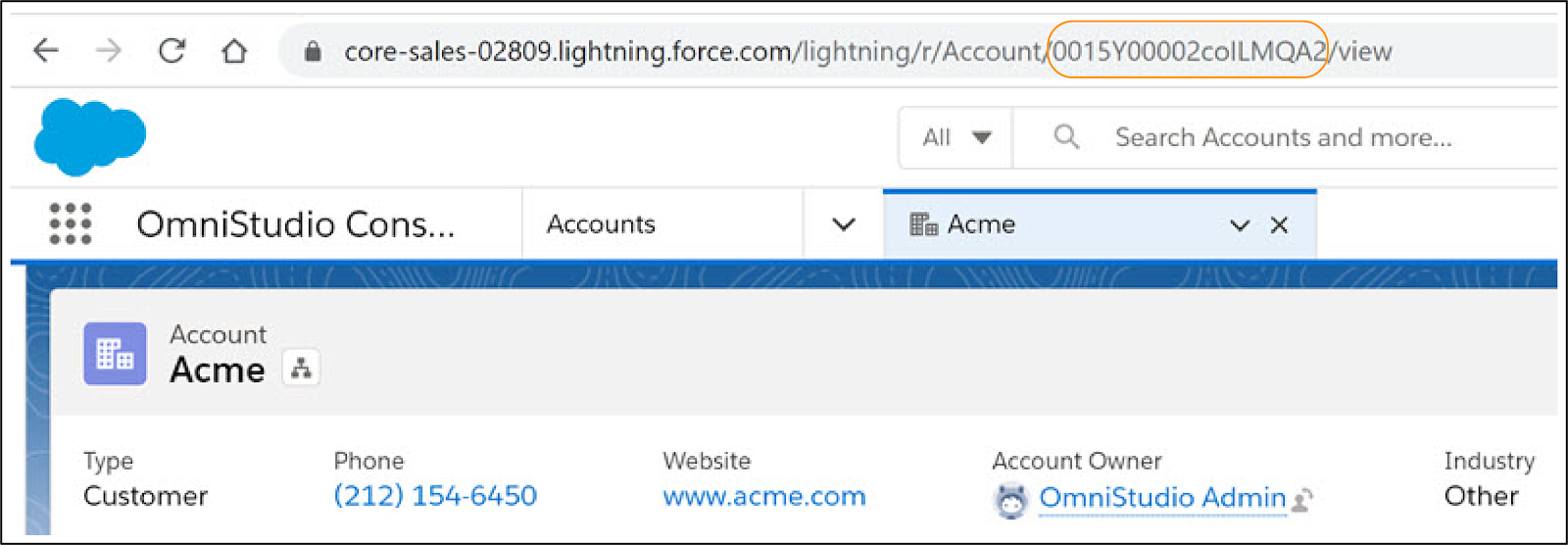
How do you configure all this in the Integration Procedure itself?



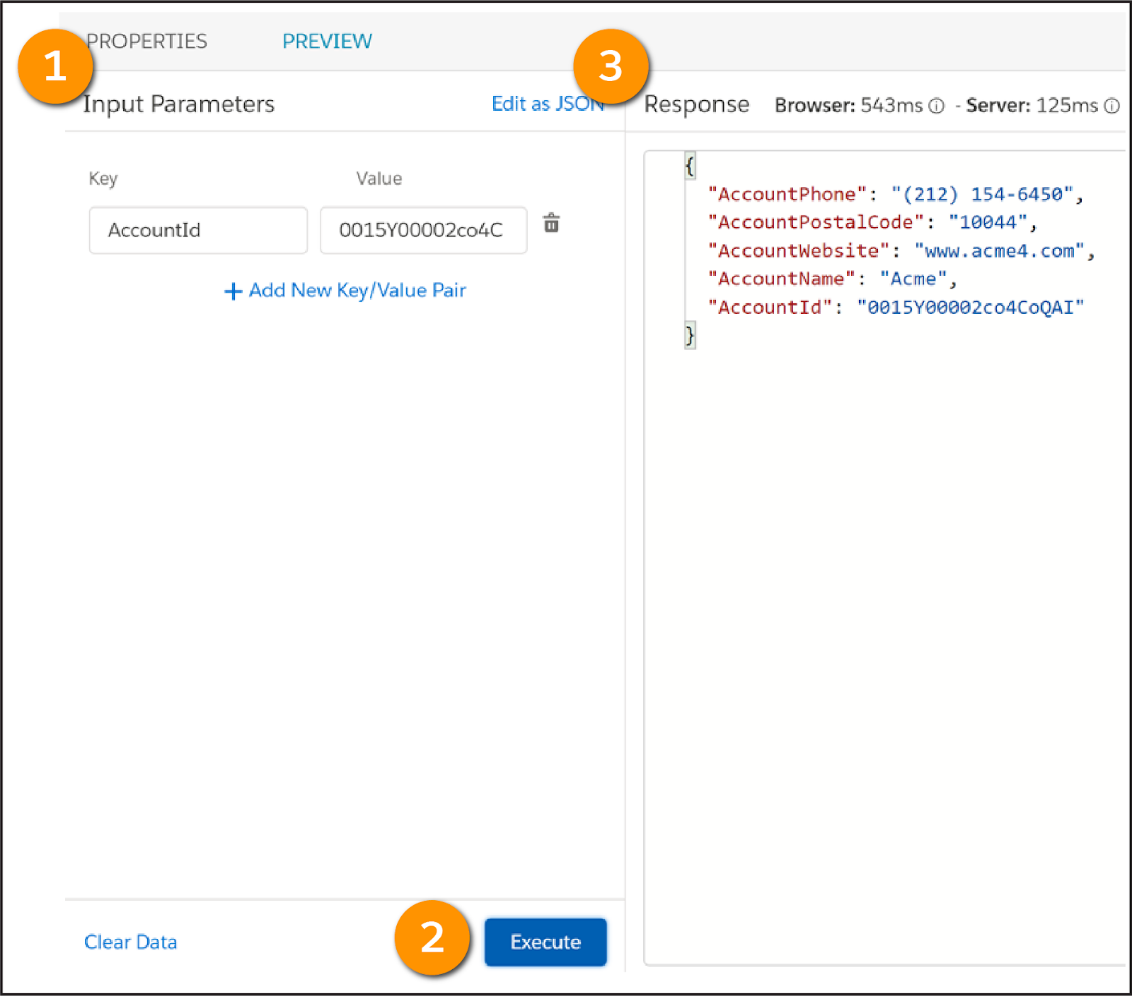
1. Drag a **DataRaptor Turbo Action** (1) from the Available Components panel to the Structure panel.
2. Give it a name that describes its purpose, such as DRGetAccountDetails.
3. In the DataRaptor Turbo Action’s Properties panel, select the DataRaptor Turbo Extract you want to use in the DataRaptor Interface (1) field.
4. Under Input Parameters (2), enter AccountId for both the Data Source and Filter Value. Remember, AccountId is an arbitrary name for the RecordId, but it must be the same in each of the elements for the data to flow properly between them and Salesforce.
5. Configure the Response Action (1) to tell the Integration Procedure what data to send back to the Edit Account OmniScript. To trim the data and return only what the OmniScript needs, enter DRGetAccountDetails in the Send JSON Path (2) field.

Way to go! You’ve configured the Integration Procedure, but now you want to test if it’s working. You need the RecordId of an account to do this, so let’s take a moment to explain what this is and how you find it.

All records in Salesforce have a unique RecordId. To find a RecordId for an account, simply open any Account record in your org and copy the RecordId from the URL.

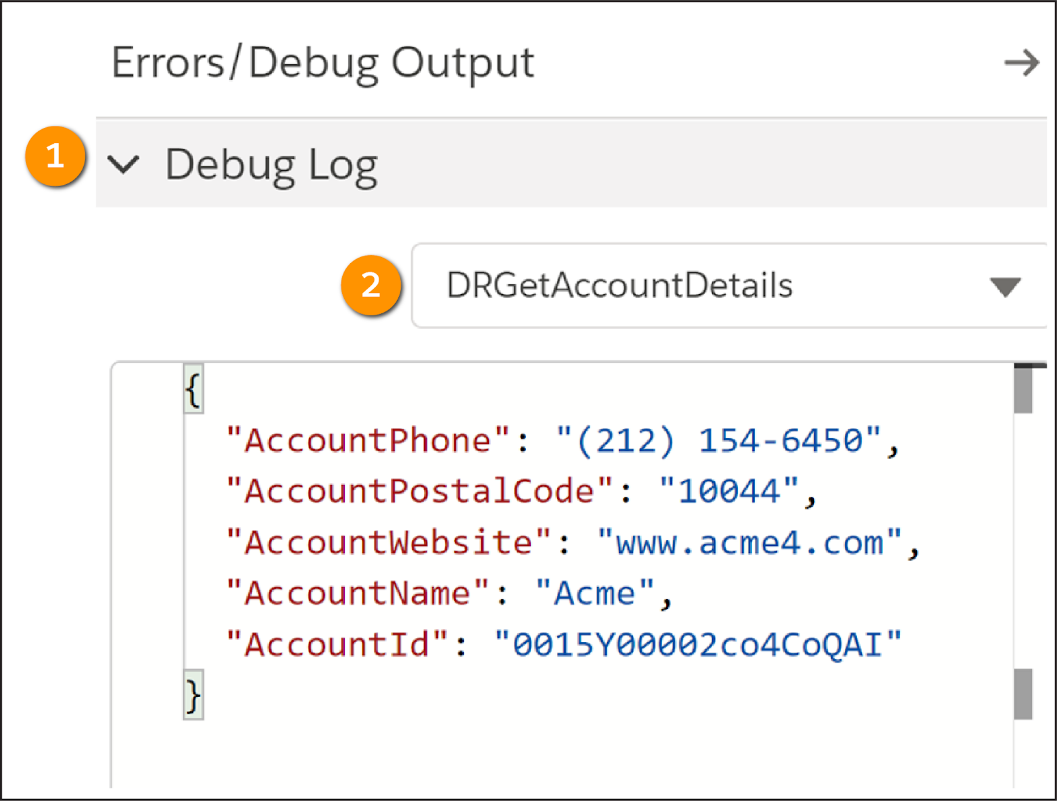


Now that you have an AccountId, you’re ready to test.



On the Integration Procedure’s Preview tab, enter a Key/Value pair in the Input Parameters panel (1) such as AccountId for the Key and an Account’s RecordId for the Value.

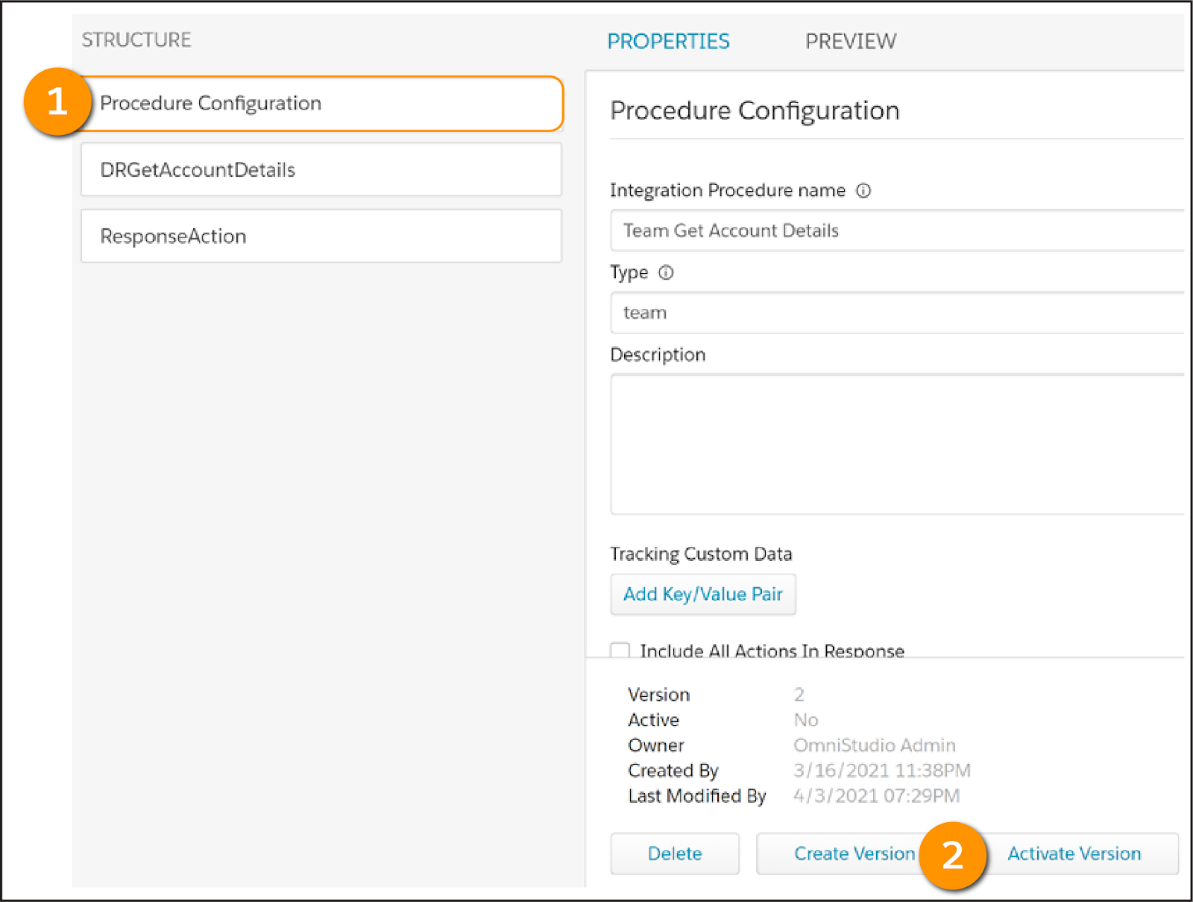
Click **Execute** (2) to run the Integration Procedure. The Response pane (3) returns results to confirm it is extracting data correctly.



The Preview tab includes a Debug Log (1), which helps you to work through and identify any issues with the output and the input for each element in the Integration Procedure. To ensure the DataRaptor is connected to the Integration Procedure, select **DRGetAccountDetails** (2) and confirm that the Account information displays.

In the Debug Log dropdown, Original Input is what you expect to pass to the Integration Procedure. The Debug Log includes two entries for each element. NAME displays the output and NAMEDebug shows the input for the step. If you don't see expected outputs, review the debug entry to find out what is wrong with the input data.

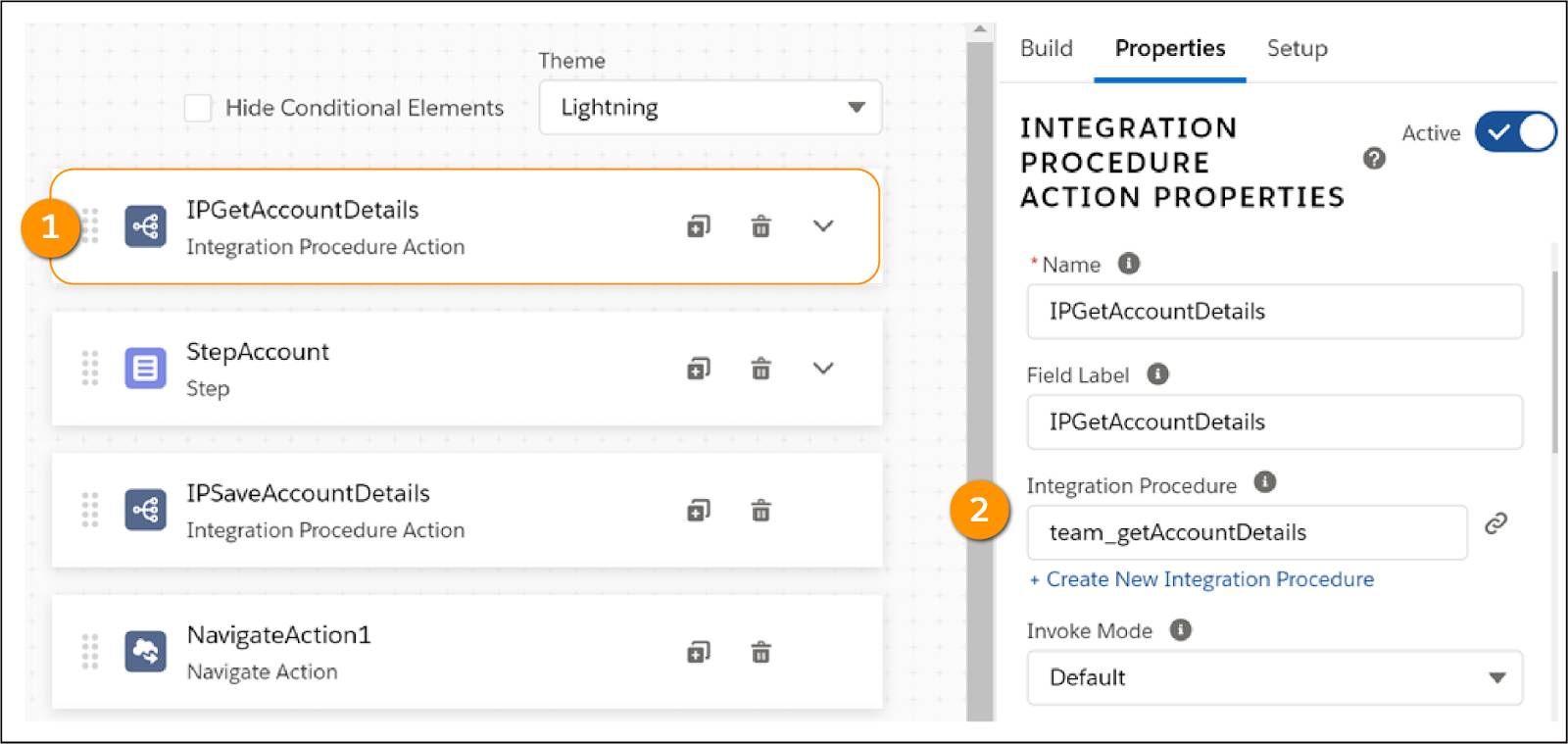
Everything seems correct and in order, so you’re ready to activate this version of the Integration Procedure.



In the Procedure Configuration (1) element properties, click **Activate Version** (2).

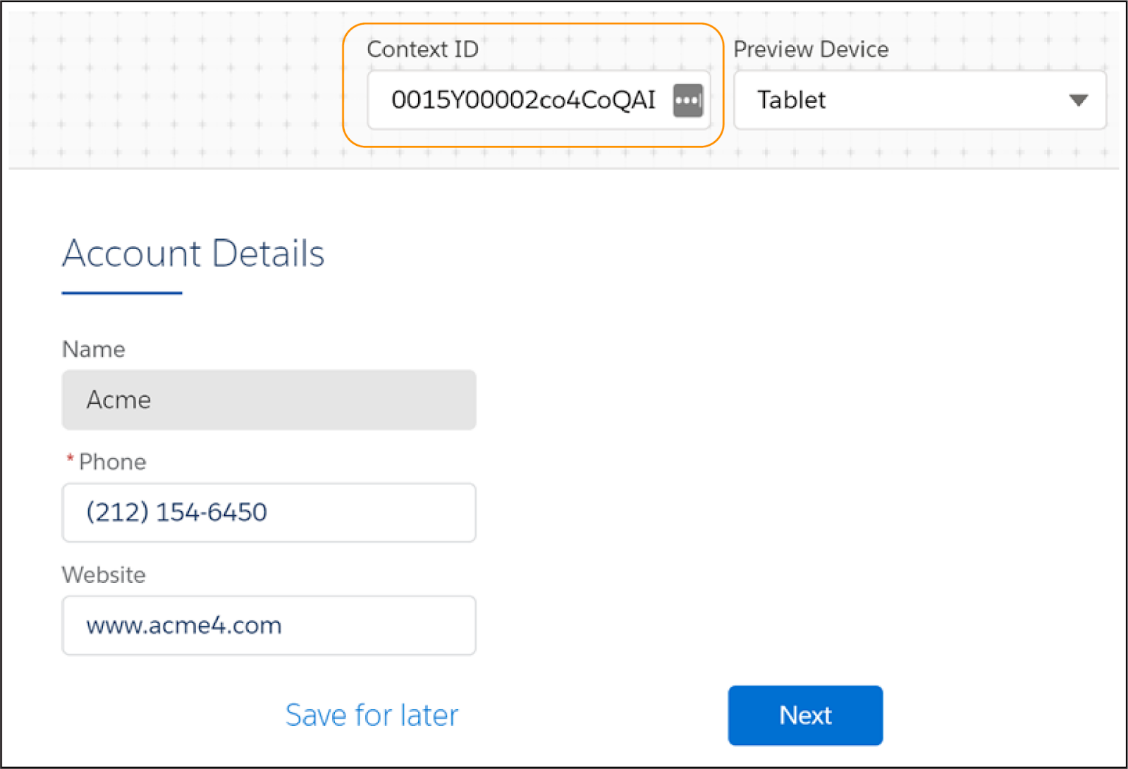
Do a final test to confirm the Integration Procedure is working in the Edit Account OmniScript, and that the Integration Procedure is available for use in the OmniScript only if it’s active.

The Edit Account OmniScript includes Integration Procedure Actions that you can configure to link Integration Procedures to the OmniScript.



The IPGetAccountDetails Integration Procedure Action (1) shows team\_getAccountDetails selected in the Integration Procedure field (2).

From the Edit Account OmniScript’s Preview tab, enter your RecordId into the Context ID field. The Preview displays data retrieved by the Integration Procedure.

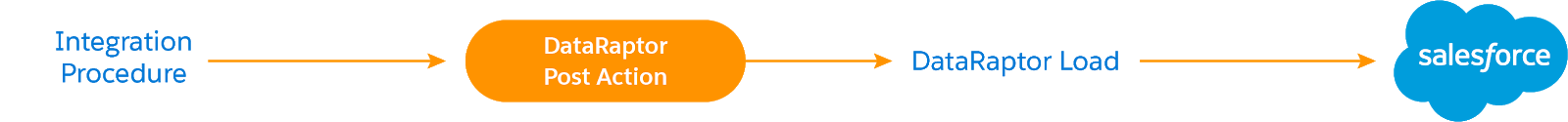


Congratulations! You’ve learned how to create an Integration Procedure that gets data for an OmniScript.

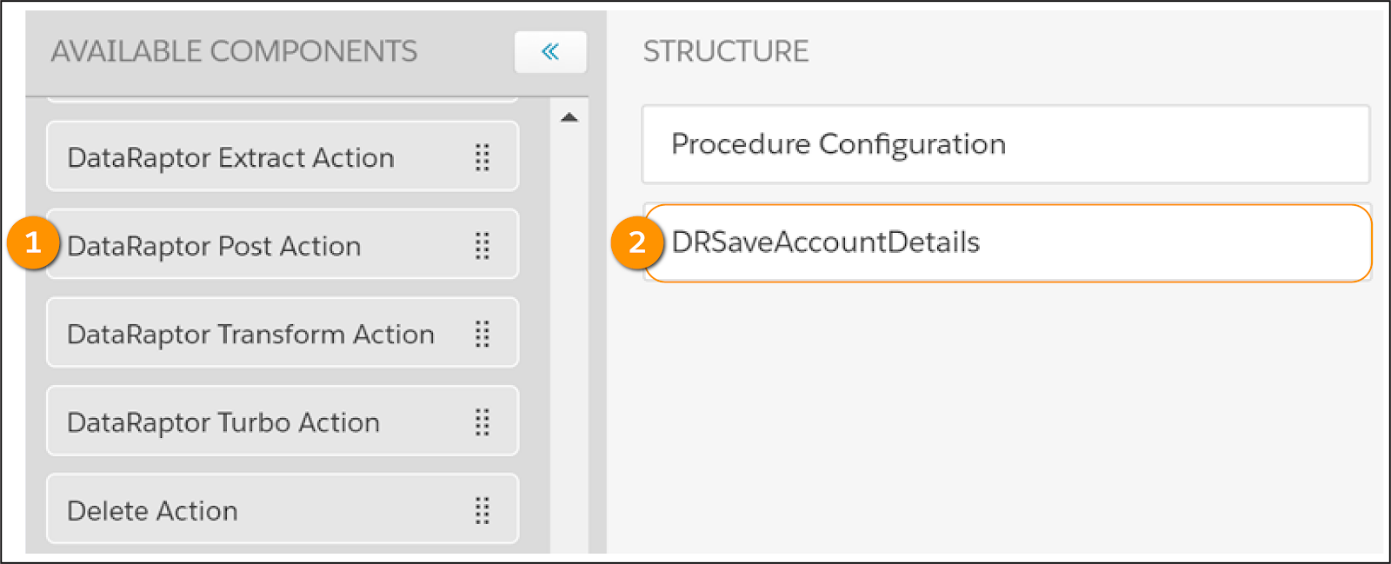
The data displayed for the Website is www.acme4.com. But what if you want to change that to www.acme.com? Let’s turn our attention to an Integration Procedure that saves data from an OmniScript.

**Build an Integration Procedure to Save Data**

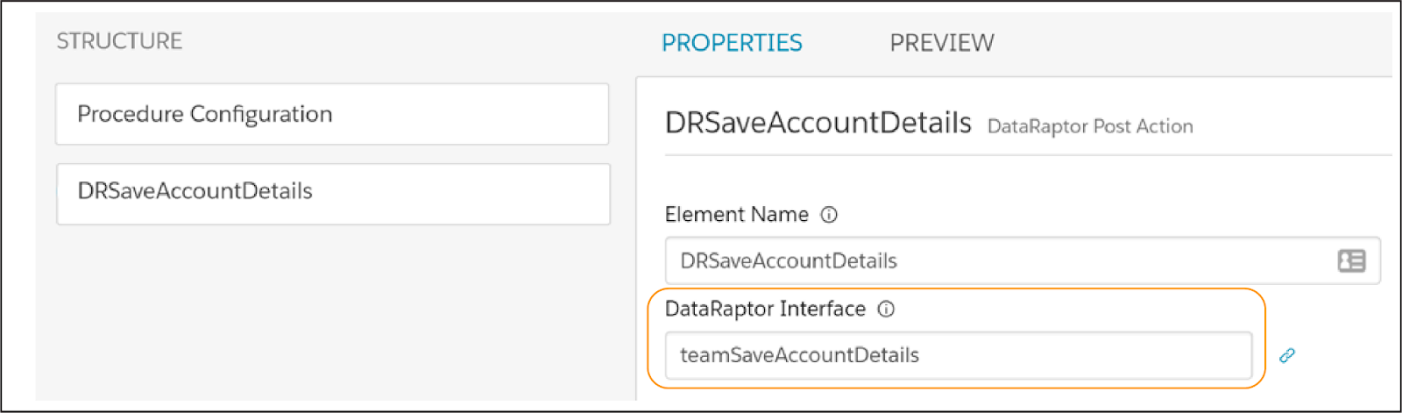
When you designed the Integration Procedure for this scenario, you chose the DataRaptor Post Action element to save updated data from the Edit Account OmniScript to Salesforce. Here’s the flow.



Let’s configure it.

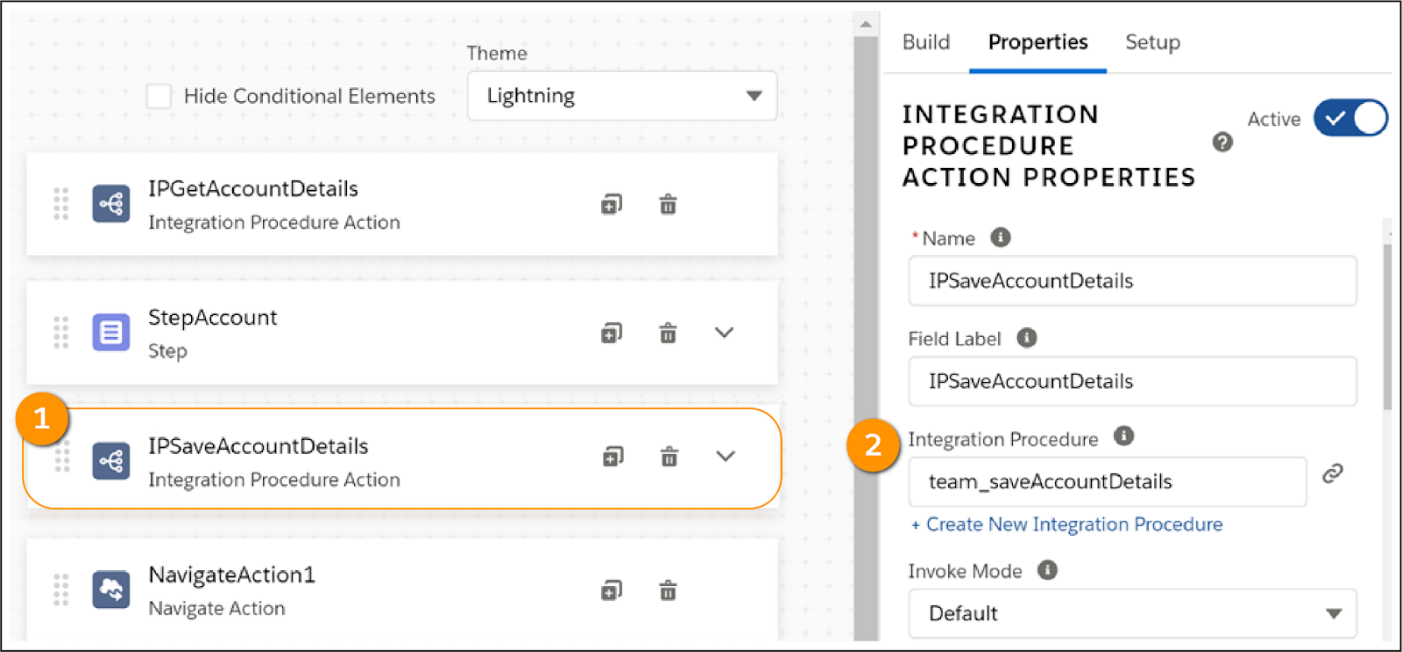


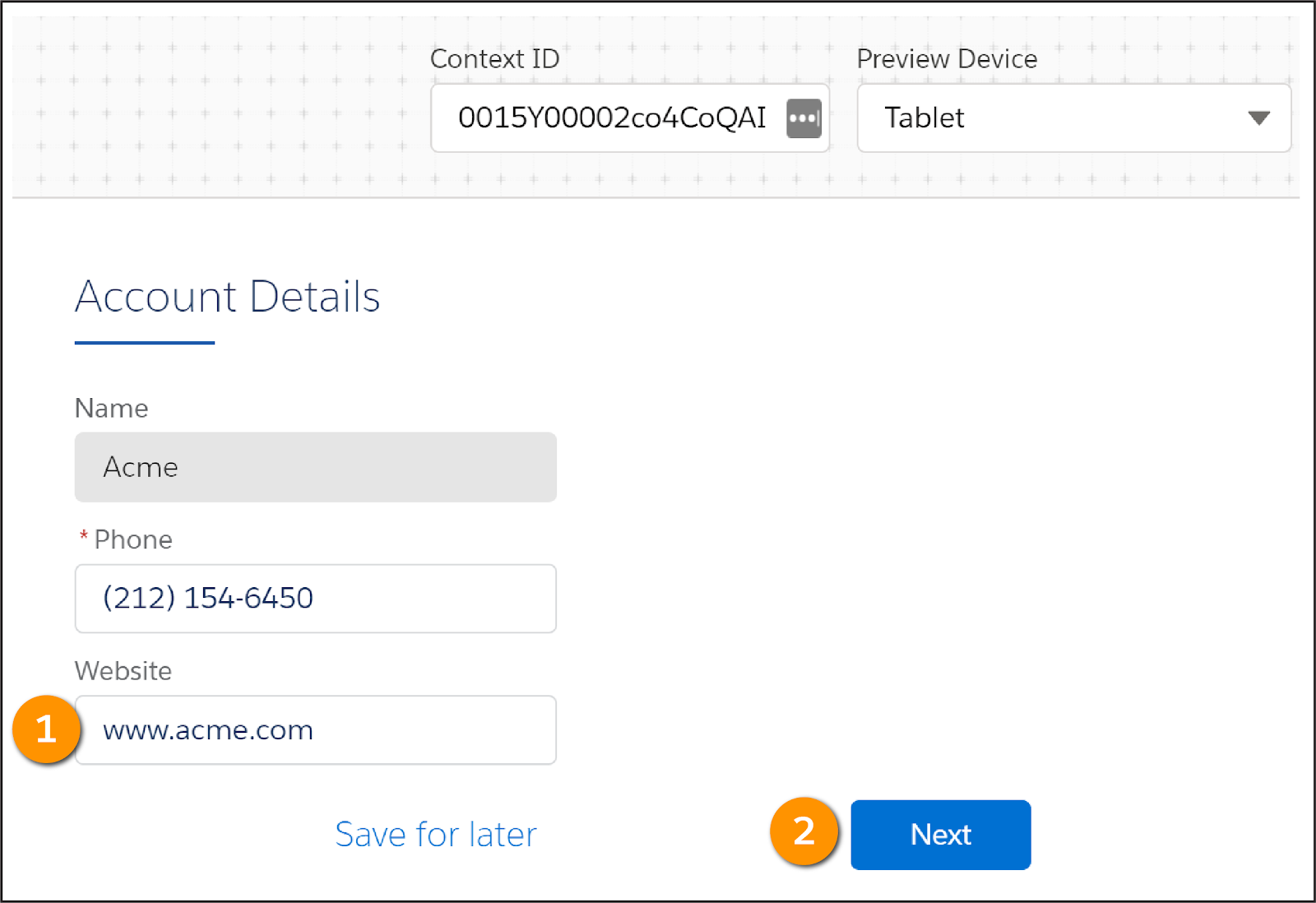
1. Drag a **DataRaptor Post Action** (1) from the Available Components panel to the Structure panel.
2. Give it a name that describes its purpose, such as DRSaveAccountDetails(2).

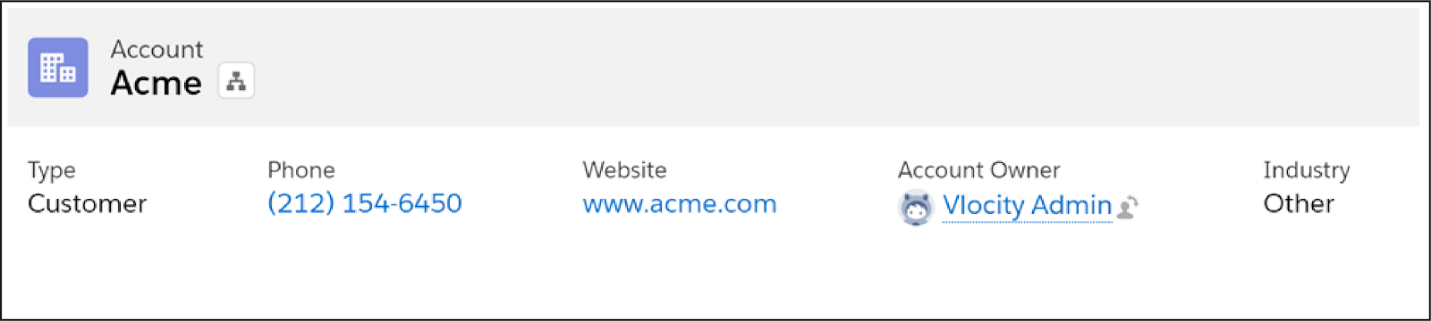
In the DataRaptor Post Action’s Properties panel, select the DataRaptor Load you want to use in the DataRaptor Interface field.

Now, you need to activate this Integration Procedure to make it available for use in the Edit Account OmniScript.

Test if the Integration Procedure is working in the Edit Account OmniScript.



1. In the Edit Account OmniScript, link the IPSaveAccountDetails Integration Procedure Action (1) to an Integration Procedure, in this case, the team\_saveAccountDetails(2).
2. From the Edit Account OmniScript’s Preview tab, enter your RecordId into the Context ID field. The Preview displays data retrieved by the first Integration Procedure you built.
3. To test if the second Integration Procedure works, make a change to the data. Let’s change the www.acme4.com website to www.acme.com (1) and click **Next**(2).
4. Open the Account record to confirm the data was updated successfully.



We hope you’re feeling pretty confident about your Integration Procedure skills now. Remember, if you need to get, save, and manipulate data behind the scenes, Integration Procedures are your go-to tools.