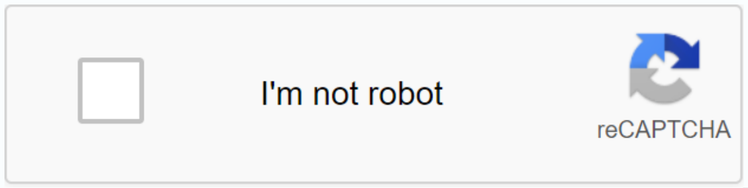

Continue

Apex picklist value

This common method uses Dynamic Apex to retrieve picklist values by passing on the name of the object API and the required field API name. We can reuse this method in aura components where it is needed. With the increasing popularity and adoption of the Force.com platform, I'm seeing a huge growth in custom Visualforce pages, especially pages that require interaction with multiple objects. A requirement that comes up regularly is to display an object's picklist values when the page uses a custom controller. To achieve this, we need to look at the very powerful and useful Dynamic Apex Describe Information capabilities. These capabilities allow us to interrogate a field or sObject and describe their characteristics. In our requirement above, we want to describe the fields on an object, especially a picklist field to determine the list of values. Let's just say we have a custom object called OfficeLocation__c. OfficeLocation__c contains a number of fields, including a picklist of land values that are called creatively Country__c. Our customer requirements are to include the picklist of countries on a Visualforce page that uses a custom controller. The first thing we need to do within our controller is to use the getDescribe() method to obtain information about the Country__c field: Schema.DescribeFieldResult fieldResult = OfficeLocation__c.Country__c.getDescribe(); We know Country__c is a picklist, so we want to get the picklist values: List<Schema.PicklistEntry> ple = fieldResult.getPicklistValues(); All we can do is map the picklist values in a <apex:selectOptions>tag that can be used for display. Here is our controller's full method of doing this: Public List<SelectOption> getCountries() { List<SelectOption> options = new list<SelectOption>(); Schema.DescribeFieldResult fieldResult = OfficeLocation__c.Country__c.getDescribe(); List<Schema.PicklistEntry> ple = fieldResult.getPicklistValues(); for: Schema.PicklistEntry f : ple) { options.add(new SelectOption(f.getLabel(), f.getValue()); } return options; } With our controller logic in full, we can call the getCountries() method on our Visualforce page and complete the <apex:selectList>tag: <apex:selectList id=countries={! Office_Location__c.Country__c} = = = size=1 required=true> <apex:selectOptions value={!countries}></apex:selectOptions> </apex:selectList> Search for an answer or ask a question about the zone or customer service. Need help? Firing Dismiss I have a Student object and have a Student Status list box in this item. Status can be full-time or part-time. Now my demand is that I want this status list, Part-Time or Full-Time in apex controller. how can I do this? If I have a student record it will only give me that particular student status</apex:selectList> </Schema.PicklistEntry> </SelectOption> </SelectOption> </SelectOptry> </SelectOptry> </Schema> part-time. But I have all the options available in the student status field. Help me see a list of values in Visualforce, must create the following method that Apex public List<Selectoption> getItemsList(){ List<SelectOption> options = new List<SelectOption>(); List<Schema.Picklistentry> fieldResult = CustomObjectName__c.CustomFieldName__c.getDescribe().getPicklistValues(); options.add('',-- select -- ');' for(Schema.PicklistEntry f : fieldResult) { options.add(new SelectOption(f.getValue(), f.getLabel()); } return options; } And then display it on visualforce page <apex:pageBlockSectionItem> <apex:outputLabel value={$ObjectType.CustomObjectName__c.fields.CustomFieldName__c.label}></apex:outputLabel> <apex:selectList value={!items} multiselect=false id=items size=1> <apex:selectOptions value={! ItemsList}&></apex:selectOptions> </apex:selectList> </apex:pageBlockSectionItem> Did you know that lightning web components help you get picklist values for each object field without writing a single line of Apex? In this blog post, learn how to combine a custom App Builder property, the context of the feature runtime, and the USER Interface API to retrieve picklist values. I present a completely generic approach that does not include hardcoding data such as the record type ID, the name of the object or the field name. I'll share the code of a sample picklist buttons component that uses that approach. This component is an alternative to the standard built-in Pad feature. It's interesting to use it if you have picklists that contain values that don't represent sequential order (as opposed to, say, opportunity phases). Let's dive into the various tools and techniques that make this possible. First, we start with custom properties of Lightning App Builder, then explore the information provided in the page context. We end with the User Interface API. Features of the Lightning App Builder component The first feature you're going to use to prevent hard-coding data is a few features of the Lightning App Builder component. Allows administrators to reuse the component as will by specifying the properties declaratively when the component is deployed on a Lightning page. To achieve this, configure the component metadata file as follows: <?xml version=1.0 encoding=UTF-8?><LightningComponentBundle xmlns= amp;gt; <apiVersion>46.0</apiVersion> <isExposed>true</isExposed> <targets> <target>lightning__RecordPage</target> </targets> <targetConfigs> <targetConfig <property name=qualifiedFieldName type=String label=Qualified Field Name required=true></property> <property name=label type=String label=Label required=true></property> </targetConfig> </targetConfigs> </LightningComponentBundle> U stelt uw component bloot aan de Lightning App</Schema.Picklistentry></SelectOption></SelectOption></Selectoption> App</Schema.Picklistentry></SelectOption></SelectOption></Selectoption> by setting the isExposed flag to true. You limit the use of your component to record pages with a lightning__RecordPage purpose. This is required for the next step in which you retrieve object and context information. Finally, you expose the properties of qualifiedFieldName and label components to the Lightning App Builder. You must also declare the related properties with @api qualifiedFieldName and @api label in your component's JavaScript. qualifiedFieldName has the name of the qualified picklist field (e.g. Account.Rating). Keep in mind that there is no built-in input validation in the App Builder that checks that the specified value is a name of a picklist field or that the value is in the correct format. If the administrator enters an incorrect value in the Qualified Field Name field (an unknown field name, an unqualified field name, a field name that is not a picklist, etc.), there is an error. In that case, your component should display a user-friendly error message, but a message that doesn't prevent the page from being saved. To validate user input, create a dynamic picklist parameter in the App Builder and use it as a data source for your parameter. Use that to limit user selection to field names on the list. Component runtime context Now that your component can be placed on record pages, you benefit from certain context data about runtime. Among other things, you can access the object name and record ID with the following public properties: @api recordId; @api objectApiName; These properties automatically receive a value when the component appears. You then use these values in your UI API calls to retrieve additional object and include information. User Interface API The UI API is the last (and most important) piece of the puzzle. A major advantage of using Lightning Web Components through Aura is that you have access to many more UI API resources with the wire service. You get access to valuable information with a minimum amount of code. You'll need to use three UI API wire adapters to retrieve your picklist values: getPicklistValues allows us to get all picklist values for a particular record type. The record type ID doesn't appear directly in the component context, so you'll need another API call for the UI to retrieve it. getRecord allows us to retrieve record data. You use it to retrieve the current value of the picklist field and the record type ID. You need this along with getRecord because in some cases the record type ID isn't specified in the record data. In this case, you must find the default record type ID. To chain this API, to to be retrieved from a tracked picklistValue property and the list of all picklist values in a tracked button property: Get the item information First wire adapter you need to call is getObjectInfo. By calling this thread, you get the default record type ID for that record. If the record is not specified over a record type, you fall back on that default value for that item. @wire(getObjectInfo, { objectApiName: '$objectApiName' }) getObjectInfo({ error, data }) { if (data) { this.defaultRecordTypeId = data.defaultRecordTypeId; // Check that we must override record type with default as (this.hasRecordTypeId === false) { this.recordTypeId = this.defaultRecordTypeId; } else if (error) { // Handle error } } You pass it an objectApi parameterName with the value of the property ApiName that you have previously registered. Note the specific syntax of the parameter value: A string with a $ prefix. This allows you to pass a reactive parameter value. What this means is that the thread is called every time the parameter value changes. You wonder: is the value of objectApiName ever going to change? The answer to that question is yes, at least once. When the page appears, the property value is not set for a few milliseconds. You must respond to the value change (the property's initialization) to call the thread with the correct object name. Get the record Once you've retrieved the item information, you get the record data using the getRecord adapter. You specify the record ID and the fields you want to pick up. In your case, that's only the name of the list box field that you defined in the Lightning App Builder and stored in qualifiedFieldName. You retrieve and save two things from the wire data: the record type ID in a recordTyped property and the picklist field value in a picklistValue property. @wire(getRecord, { recordId: '$recordId', fields: '$qualifiedFieldName' }) getRecord ({ error, data }) { if (data) { // Check if record data includes record type if (data.recordTypeInfo) { this.hasRecordTypeId = true; this.recordTypeType = data.recordTypeTypeId; } different { // Record type is missing from record data this.hasRecordTypeId = false; // Use default type if available (it can still be loaded) as (this.defaultRecordTypeId) { this.recordTypeId = this.defaultRecordTypeId; } // Current picklist value fetch const fieldName = this.getFieldName(); this.picklistValue = data.fields[fieldName].value; } else (if error) { // Handle error } } Please note that we typically recommend not using dynamic field names (such as qualifiedFieldName) when working with Salesforce data. Instead, you'll need to import static field reference to ensure dependency integrity (for example, we won't let you delete a record field referenced in a component). We make an exception for our use case because builds a generic component. Keep in mind that this flexibility implies that an administrator may be able to remove or rename the picklist field the component. You've been warned! As mentioned earlier, the record data returned by the thread may not contain the record type ID. This happens when a record is created at a time when no record types are defined. In that case, you'll fall back on the default record type ID that you retrieve from the object Info property. Get the picklist values Right now gives you access to the record type ID and you can call the last wire adapter: getPicklistValues. This gives us the list of all picklist values for your record type. @wire(getPicklistValues, { recordTypeD: '$recordTypeId', fieldApiName: '$qualifiedFieldName' }) getPicklistValues({ error, data }) { if (data) { // Card picker values to buttons this.buttons = data.values.map(plValue => { return { label: plValue.label, value: plValue.value }; }); } different as (error) { // Handle } error } With that last thread call, you now have access to all picklist values. In the case of our picklist buttons component, all you have to do is to map these values to the property buttons to render them as buttons. Summary That's a wrap! You've learned how to get picklist values in Lightning web components without writing a single line of Apex thanks to the combined use of App Builder parameters, page context, and UI APIs. Another great advantage of not using Apex in this context is that you benefit from the Lightning Data Service when using the UI API. For example, if you change the value of the picklist in the record details component, you'll see that your custom component is automatically updated with the new value. Feel free to explore the code presented in this post in this sample picklist button component. About the author Philippe Ozil is Lead Developer Evangelist at Salesforce where he focuses on the Salesforce Platform. He writes technical content and speaks regularly at conferences. He is a full stack developer and enjoys working on robotics and VR projects. Follow him on Twitter @PhilippeOzil or check out his GitHub projects @pozil. Source: Salesforce