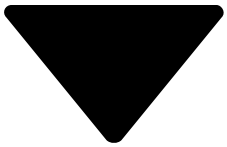
1. Lightning Data Service Basics for Aura Components



2. Handle Record Changes and Errors

Handle Record Changes and Errors

Learning Objectives

After completing this unit, you'll be able to:

- Explain how Lightning Data Service for Aura Components uses notifications.
- Use recordUpdated to handle errors and record changes.

Record Changes

In the last unit, we covered how Lightning Data Service handles CRUD. Now let's find out how to take actions when your record changes, so that your components can respond to a record load, change, update, or deletion.

To take an action when a record changes, handle the `recordUpdated` event.

```
<force:recordData aura:id="forceRecordDataCmp"
    recordId="{!v.recordId}"
    layoutType="{!v.layout}"
    targetRecord="{!v.record}"
    targetFields="{!v.simpleRecord}"
    targetError="{!v.error}"
    recordUpdated="{!c.recordUpdated}" />
```

Copy

Implement an action handler that handles the change. Field changes are passed in through the `changedFields` object, which contains the old field values as they relate to the new values.

```
({
    recordUpdated: function(component, event, helper) {
        var eventParams = event.getParams();
        if(eventParams.changeType === "CHANGED") {
```

```
            // get the fields that are changed for this record
            var changedFields = eventParams.changedFields;
            console.log('Fields that are changed: ' + JSON.stringify(changedFields));
            // record is changed so refresh the component (or other component logic)
            var resultsToast = $A.get("e.force:showToast");
            resultsToast.setParams({
                "title": "Saved",
                "message": "The record was updated."
            });
            resultsToast.fire();
        } else if(eventParams.changeType === "LOADED") {
            console.log("Record is loaded successfully.");
        } else if(eventParams.changeType === "REMOVED") {
            var resultsToast = $A.get("e.force:showToast");
            resultsToast.setParams({
                "title": "Deleted",
                "message": "The record was deleted."
            });
            resultsToast.fire();
        } else if(eventParams.changeType === "ERROR") {
            console.log('Error: ' + component.get("v.error"));
        }
    }
})
```

Copy

When LDS detects a record change, it notifies the components that use the record of the changes. If you don't handle the change, the record is still updated, so any reference to the `targetRecord` or `targetFields` properties automatically shows up in your components. For every `force:recordData` component referencing the updated record, LDS does two things.

- LDS notifies all other instances of `force:recordData` of the change by firing the `recordUpdated` event with the appropriate `changeType` and `changedFields` value.
- It sets the `targetRecord` and `targetFields` attribute on each `force:recordData` to the new record value. If `targetRecord` or `targetFields` is referenced by any UI, this automatically triggers a rerender so that the UI displays the latest data.

Note

If `force:recordData` is in EDIT mode, `targetRecord` and `targetFields` are **not** automatically updated when the record changes. This is to avoid clobberin' edits that are still in progress and to prevent unsaved changes from appearing in other components. Never fear, records can be refreshed manually by handling the `recordUpdated` event and calling the `reloadRecord` method.

When LDS detects a change originating from the server, it uses the same change notification mechanism. LDS detects a change on the server when it receives a new request for a record. LDS only notifies components that are registered and flagged as `isValid`. When handling the `recordUpdated` event, check the `changeType` for the kind of changes you want to handle.

Error Handling

The `targetError` attribute is set to a localized error message if an error occurs on load. An error occurs if the attributes in `force:recordData` are invalid, or if the server is unreachable and the record isn't in the local

cache. From there, it's up to you to decide how to display the error.

If the record becomes inaccessible on the server, the `recordUpdated` event fires with `changeType` = `REMOVED`, and no error is set to `targetError`, because a record becoming inaccessible is sometimes an expected outcome. Records can also become inaccessible due to record or entity sharing and visibility settings, or because the record was deleted.

Putting It All Together

Congrats! You now know everything you need to know to get started with Lightning Data Service. It's relatively simple, but it does a lot! Before you go off and earn that fancy new badge, lets put all of that theory into practice. We'll put together a page that contains two components that use the same record data and properly respond to record changes.

This component displays a contact's details. Note that it uses `fields` instead of `layoutType`, and `targetFields` but not `targetRecord`. Remember that either `fields` or `layoutType` (or both!) can be included, and you can use either `targetFields` or `targetRecord` (or both!) to retrieve record data. With `fields` you have to specify which specific fields you want to query, while with `layoutType`, you just need to specify the record layout you want to use, either FULL or COMPACT. If you use `targetFields` to retrieve data, which is the preferred method, use the format `v.targetFields.Name` in your UI. If you're using `targetRecord`, use `v.targetRecord.fields.Name.value`.

ldsShowContact.cmp

```
<aura:component implements="force:hasRecordId,flexipage:availableForRecordHome">
    <aura:attribute name="contactRecord" type="Object"/>
    <aura:attribute name="recordLoadError" type="String"/>

    <force:recordData aura:id="recordLoader"
        recordId="{!v.recordId}"
        fields="Name,Description,Phone"
        targetFields="{!v.contactRecord}"
        targetError="{!v.recordLoadError}"
    />
    <!-- Display a lightning card with details about the contact -->
    <lightning:card iconName="standard:contact" title="{!v.contactRecord.Name}" >
        <div class="slds-p-horizontal--small">
            <p class="slds-text-heading--small">
                <lightning:formattedPhone title="Phone"  value="{!v.contactRecord.Phone}" /></p>
            <p class="slds-text-heading--small">
                <lightning:formattedText title="Description" value="{!v.contactRecord.Description}" /></p>
        </div>
    </lightning:card>
</aura:component>
```

Copy

Our next component loads the same contact, but in EDIT mode, along with a form that lets the user edit the contact itself. It also handles the `recordUpdated` event so that we can take certain actions depending on the outcome of the edit.

ldsEditContact.cmp

```
<aura:component implements="force:hasRecordId,flexipage:availableForRecordHome">
    <aura:attribute name="contactRecord" type="Object"/>
    <aura:attribute name="recordSaveError" type="String" default=""/>
    <!-- Load record in EDIT mode -->
    <force:recordData aura:id="recordLoader"
```

```
        recordId="{!v.recordId}"
        fields="Name,Description,Phone"
        targetFields="{!v.contactRecord}"
        targetError="{!v.recordSaveError}"
        mode="EDIT"
        recordUpdated="{!c.handleRecordUpdated}" />
    <!-- Contact edit form -->
    <lightning:card iconName="action:edit" title="Edit Contact">
        <div class="slds-p-horizontal--small">
            <lightning:input label="Contact Name" value="{!v.contactRecord.Name}"/>
            <lightning:input label="Contact Description" value="{!v.contactRecord.Description}"/>
            <lightning:input label="Contact Phone" value="{!v.contactRecord.Phone}"/>
            <br/>
            <lightning:button label="Save Contact" variant="brand" onclick="{!c.saveContact}" />
        </div>
    </lightning:card>

    <!-- Display error message -->
    <aura:if isTrue="{!not(empty(v.recordSaveError))}">
        <div class="recordSaveError">
            {!v.recordSaveError}</div>
    </aura:if>
</aura:component>
```

Copy

Finally, we have our controller, which reports a successful edit with a toast message or shows us an error message, depending on the outcome of the edit.

ldsEditContactController.js

```
({
    saveContact : function(cmp, event, helper) {
        var recordLoader = cmp.find("recordLoader");
        recordLoader.saveRecord($A.getCallback(function(saveResult) {
            if (saveResult.state === "ERROR") {
                var errMsg = "";
                // saveResult.error is an array of errors,
                // so collect all errors into one message
                for (var i = 0; i < saveResult.error.length; i++) {
                    errMsg += saveResult.error[i].message + "\n";
                }
                cmp.set("v.recordSaveError", errMsg);
            } else {
                cmp.set("v.recordSaveError", "");
            }
        }));
    },
    // Control the component behavior here when record is changed (via any component)
    handleRecordUpdated: function(component, event, helper) {
        var eventParams = event.getParams();
        if(eventParams.changeType === "CHANGED") {
            // get the fields that are changed for this record
            var changedFields = eventParams.changedFields;
            console.log('Fields that are changed: ' + JSON.stringify(changedFields));
            // record is changed so refresh the component (or other component logic)
            var resultsToast = $A.get("e.force:showToast");
            resultsToast.setParams({
                "title": "Saved",
                "message": "The record was updated."
            });
            resultsToast.fire();
        } else if(eventParams.changeType === "LOADED") {
            // record is loaded in the cache
        } else if(eventParams.changeType === "REMOVED") {
            // record is deleted and removed from the cache
        } else if(eventParams.changeType === "ERROR") {
```

```
            console.log('Error: ' + component.get("v.error"));
        }
    }
})
```

Copy

There you have it, you're a Lightning Data Service master. Now go forth and use your newfound skillz to improve the performance and consistency of your Lightning components for Trailblazers everywhere.

Resources

- Lightning Data Service: Record Changes
- Lightning Data Service: Errors

Get Ready

You'll be completing this challenge in your own personal Salesforce environment. Choose from the dropdown menu, then click **Launch** to get started. If you use Trailhead in a language other than English, set the language of your Trailhead Playground to English before you attempt this challenge. Want to find out more about using hands-on orgs for Trailhead learning? Check out the Trailhead Playground Management module.

Your Challenge

Use force:recordData to create a component that shows an error message if it is loaded with invalid data Build on the components you created in the previous unit challenge by adding notification handling to your components. If the edits in your editable component create an error, make sure the display component shows an error message.

- The **accEdit** component displays an error message if the edits made in the component are invalid
- Name the attribute that holds the error message **recordSaveError**

My Trailhead Playground 1

Launch

Check challenge to earn 500 points