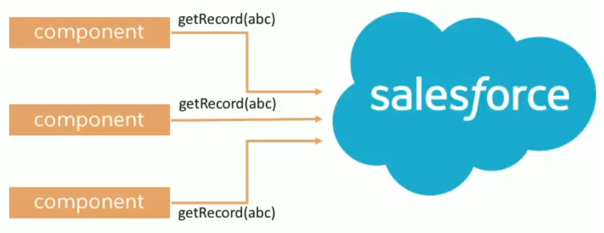
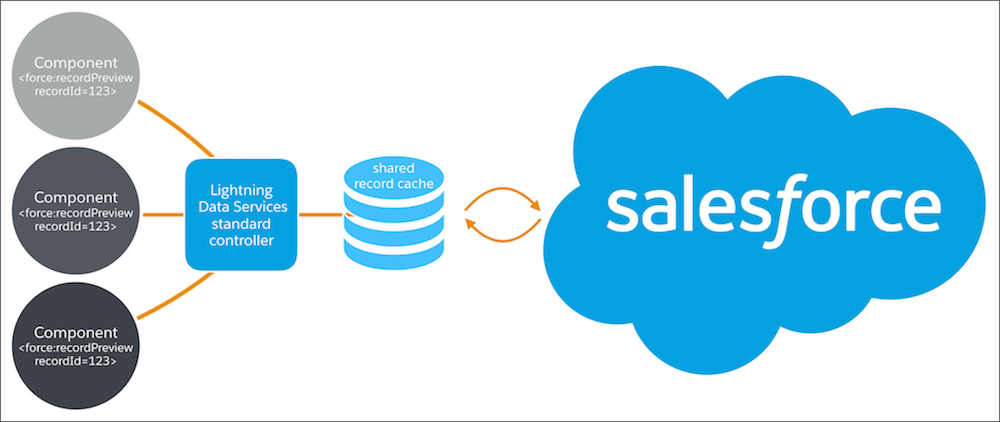
**LDS Examples**

Problems without LDS,

1. Multiple expensive server calls
2. No direct communication between components about changes in data. Need to depend on events.
3. Extra code to handle all this.
4. No cache and hence more time-consuming operations.
5. Field level security and record sharing to be managed through code.



Solution for all,



Lightning Data Service provides reusable Aura components that:

* Eliminates apex controller and still can perform CRUD operation. It’s like Standard Controllers in VF page.
* Minimize XMLHttpRequests (XHRs)
* Fetch records once, reducing network transfers, app server load, and database server load
* Cache record data on the client, separate from component metadata
* Share record data across components
* Enable progressive record loading, caching, and merging more fields and layouts into the cache
* Enable proactive cache population
* Promote consistency by using only one instance of the record data across multiple components
* Create notifications when record data changes

If you have a Lightning application that creates, reads, updates, or deletes records then LDS is the best and most efficient way to do CRUD operations.

| **Form Function** | **Tag/component to use** |
| --- | --- |
| Display, create, or edit records | lightning:recordForm |
| Display records only | lightning:recordViewForm (with lightning:outputField) |
| Create or edit records only | lightning:recordEditForm (with lightning:inputField) |
| Display, create, edit, or delete records with granular customization | force:recordData |

Example link - <https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/data_service_example.htm>

Reference link - <https://developer.salesforce.com/docs/atlas.en-us.218.0.lightning.meta/lightning/data_service.htm>

<https://trailhead.salesforce.com/content/learn/modules/lightning_data_service/lightning_data_service_manipulate_records>

Objects supported/unsupported - <https://developer.salesforce.com/docs/atlas.en-us.218.0.lightning.meta/lightning/data_service_considerations.htm>

**Record Form (lightning:recordForm)**

<lightning:recordForm

recordId="001xcxxxxxxxxxx"

objectApiName="Account"

layoutType="Compact"

mode="view"

columns="2"/>

<lightning:recordForm

recordId="{!v.recordId}"

objectApiName="Account"

layoutType="Compact"

mode="view"

columns="2"/>

<aura:attribute name="fields" type="String[]" default="['LastName','Phone','Email']" />

<lightning:recordForm

recordId="{!v.recordId}"

aura:id="myRecordForm"

objectApiName="Contact"

fields="{!v.fields}"

columns="2"

mode="edit"

onsubmit="{!c.handleSubmit}" />

({

handleSubmit : function(cmp, event, helper) {

event.preventDefault(); *// stop the form from submitting*

const fields = event.getParam('fields');

fields.LastName = 'My Custom Last Name'; *// modify a field*

cmp.find('myRecordForm').submit(fields);

}

})

<aura:component implements="flexipage:availableForRecordHome,force:hasRecordId" access="global" >

<aura:attribute name="fields" type="String[]" default="['Name','AnnualRevenue','Industry']" />

<aura:attribute name="recordId" type="String"/>

<lightning:notificationsLibrary aura:id="notifLib"/>

<lightning:recordForm

objectApiName="Account"

fields="{!v.fields}"

onsuccess="{!c.handleSuccess}" />

</aura:component>

({

handleSuccess : function(component, event, helper) {

component.find('notifLib').showToast({

"variant": "success",

"title": "Account Created",

"message": "Record ID: " + event.getParam("id")

});

}

})

Pre-population of field value is not supported.

compact or comfy density can be set to show labels next to value or on top of the value respectively.

**Record Edit Form (lightning:recordEditForm)**

<aura:component>

<lightning:recordEditForm recordId="" objectApiName="Contact">

<lightning:messages />

<lightning:outputField fieldName="AccountId" />

<lightning:inputField fieldName="FirstName" value="Ramesh"/>

<lightning:inputField fieldName="LastName" />

<lightning:inputField fieldName="Email" />

<lightning:button class="slds-m-top\_small" variant="brand" type="submit" name="update" label="Update" />

</lightning:recordEditForm>

</aura:component>

<aura:component>

<lightning:recordEditForm aura:id="recordEditForm"

objectApiName="Contact">

<lightning:messages />

<lightning:inputField fieldName="Name" />

<lightning:button class="slds-m-top\_small" type="submit" label="Create new" />

</lightning:recordEditForm>

</aura:component>

<aura:component>

<lightning:recordEditForm

objectApiName="Account"

onload="{!c.handleCreateLoad}">

<lightning:messages />

<lightning:inputField aura:id="nameField" fieldName="Name"/>

<lightning:button class="slds-m-top\_small" type="submit" label="Create new" />

</lightning:recordEditForm>

({

handleCreateLoad: function (cmp, event, helper) {

var nameFieldValue = cmp.find("nameField").set("v.value", "My New Account");

}

})

<aura:component>

<aura:attribute name="disabled" type="Boolean" default="false" />

<aura:attribute name="saved" type="Boolean" default="false" />

<aura:attribute name="showSpinner" type="Boolean" default="false" />

<aura:if isTrue="{!v.showSpinner}">

<lightning:spinner />

</aura:if>

<aura:if isTrue="{!!v.saved}">

<lightning:recordEditForm

onload="{!c.handleLoad}"

onsubmit="{!c.handleSubmit}"

onsuccess="{!c.handleSuccess}"

recordId="{!v.recordId}"

objectApiName="Bad\_Guy\_\_c">

*<!-- the messages component is for error messages -->*

<lightning:messages />

<lightning:inputField fieldName="Name" />

<lightning:inputField fieldName="AccountNumber" />

<lightning:inputField fieldName="Active\_\_c" />

<div class="slds-m-top\_medium">

<lightning:button disabled="{!v.disabled}" variant="brand" type="submit" name="save" label="Save" />

</div>

</lightning:recordEditForm>

<aura:set attribute="else">

<p>Saved!</p>

</aura:set>

</aura:if>

</aura:component>

**Record View Form (lightning:recordViewForm)**

<aura:component>

<lightning:recordViewForm recordId="001XXXXXXXXXXXXXXX" objectApiName="My\_Contact\_\_c">

<div class="slds-box">

<lightning:outputField fieldName="Name" />

<lightning:outputField fieldName="Email\_\_c" />

</div>

</lightning:recordViewForm>

</aura:component>

**Record Data(force:recordData)**

<aura:component implements="flexipage:availableForRecordHome, force:lightningQuickActionWithoutHeader, force:hasRecordId">

<aura:attribute name="record" type="Object"/>

<aura:attribute name="simpleRecord" type="Object"/>

<aura:attribute name="recordError" type="String"/>

<force:recordData aura:id="recordLoader"

recordId="{!v.recordId}"

targetFields="{!v.simpleRecord}"

targetError="{!v.recordError}"

recordUpdated="{!c.handleRecordUpdated}"

/>

<!-- Display a lightning card with details about the record -->

<div class="Record Details">

<lightning:card iconName="standard:account" title="{!v.simpleRecord.Name}" >

<div class="slds-p-horizontal--small">

<p class="slds-text-heading--small">

<lightning:formattedText title="Billing City" value="{!v.simpleRecord.BillingCity}" /></p>

<p class="slds-text-heading--small">

<lightning:formattedText title="Billing State" value="{!v.simpleRecord.BillingState}" /></p>

</div>

</lightning:card>

</div>

<!-- Display Lightning Data Service errors, if any -->

<aura:if isTrue="{!not(empty(v.recordError))}">

<div class="recordError">

{!v.recordError}</div>

</aura:if>

</aura:component>

({

handleRecordUpdated: function(component, event, helper) {

var eventParams = event.getParams();

if(eventParams.changeType === "LOADED") {

// record is loaded (render other component which needs record data value)

console.log("Record is loaded successfully.");

console.log("You loaded a record in " +

component.get("v.simpleRecord.Industry"));

} else if(eventParams.changeType === "CHANGED") {

// record is changed

} else if(eventParams.changeType === "REMOVED") {

// record is deleted

} else if(eventParams.changeType === "ERROR") {

// there’s an error while loading, saving, or deleting the record

}

}

})

<aura:component implements="flexipage:availableForRecordHome,force:hasRecordId">

<aura:attribute name="record" type="Object"/>

<aura:attribute name="simpleRecord" type="Object"/>

<aura:attribute name="recordError" type="String"/>

<force:recordData aura:id="recordHandler"

recordId="{!v.recordId}"

layoutType="FULL"

targetRecord="{!v.record}"

targetFields="{!v.simpleRecord}"

targetError="{!v.recordError}"

mode="EDIT"

recordUpdated="{!c.handleRecordUpdated}"

/>

<!-- Display a lightning card with details about the record -->

<div class="Record Details">

<lightning:card iconName="standard:account" title="{!v.simpleRecord.Name}" >

<div class="slds-p-horizontal--small">

<p class="slds-text-heading--small">

<lightning:formattedText title="Billing State" value="{!v.simpleRecord.BillingState}" /></p>

<p class="slds-text-heading--small">

<lightning:formattedText title="Billing City" value="{!v.simpleRecord.BillingCity}" /></p>

</div>

</lightning:card>

</div>

<!-- Display an editing form -->

<div class="Record Details">

<lightning:card iconName="action:edit" title="Edit Account">

<div class="slds-p-horizontal--small">

<lightning:input label="Account Name" value="{!v.simpleRecord.Name}"/>

<br/>

<lightning:button label="Save Account" variant="brand" onclick="{!c.handleSaveRecord}" />

</div>

</lightning:card>

</div>

<!-- Display Lightning Data Service errors, if any -->

<aura:if isTrue="{!not(empty(v.recordError))}">

<div class="recordError">

{!v.recordError}</div>

</aura:if>

</aura:component>

({

handleSaveRecord: function(component, event, helper) {

component.find("recordHandler").saveRecord($A.getCallback(function(saveResult) {

// use the recordUpdated event handler to handle generic logic when record is changed

if (saveResult.state === "SUCCESS" || saveResult.state === "DRAFT") {

// handle component related logic in event handler

} else if (saveResult.state === "INCOMPLETE") {

console.log("User is offline, device doesn't support drafts.");

} else if (saveResult.state === "ERROR") {

console.log('Problem saving record, error: ' + JSON.stringify(saveResult.error));

} else {

console.log('Unknown problem, state: ' + saveResult.state + ', error: ' + JSON.stringify(saveResult.error));

}

}));

},

/\*\*

\* 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 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") {

// there’s an error while loading, saving or deleting the record

}

}

})

<aura:component implements="flexipage:availableForRecordHome,force:hasRecordId">

<aura:attribute name="recordError" type="String" access="private"/>

<force:recordData aura:id="recordHandler"

recordId="{!v.recordId}"

fields="Id"

targetError="{!v.recordError}"

recordUpdated="{!c.handleRecordUpdated}" />

<!-- Display the delete record form -->

<div class="Delete Record">

<lightning:card iconName="action:delete" title="Delete Record">

<div class="slds-p-horizontal--small">

<lightning:button label="Delete Record" variant="destructive" onclick="{!c.handleDeleteRecord}"/>

</div>

</lightning:card>

</div>

<!-- Display Lightning Data Service errors, if any -->

<aura:if isTrue="{!not(empty(v.recordError))}">

<div class="recordError">

{!v.recordError}</div>

</aura:if>

</aura:component>

({

handleDeleteRecord: function(component, event, helper) {

component.find("recordHandler").deleteRecord($A.getCallback(function(deleteResult) {

// NOTE: If you want a specific behavior(an action or UI behavior) when this action is successful

// then handle that in a callback (generic logic when record is changed should be handled in recordUpdated event handler)

if (deleteResult.state === "SUCCESS" || deleteResult.state === "DRAFT") {

// record is deleted

console.log("Record is deleted.");

} else if (deleteResult.state === "INCOMPLETE") {

console.log("User is offline, device doesn't support drafts.");

} else if (deleteResult.state === "ERROR") {

console.log('Problem deleting record, error: ' + JSON.stringify(deleteResult.error));

} else {

console.log('Unknown problem, state: ' + deleteResult.state + ', error: ' + JSON.stringify(deleteResult.error));

}

}));

},

/\*\*

\* 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") {

// record is changed

} else if(eventParams.changeType === "LOADED") {

// record is loaded in the cache

} else if(eventParams.changeType === "REMOVED") {

// record is deleted, show a toast UI message

var resultsToast = $A.get("e.force:showToast");

resultsToast.setParams({

"title": "Deleted",

"message": "The record was deleted."

});

resultsToast.fire();

} else if(eventParams.changeType === "ERROR") {

// there’s an error while loading, saving, or deleting the record

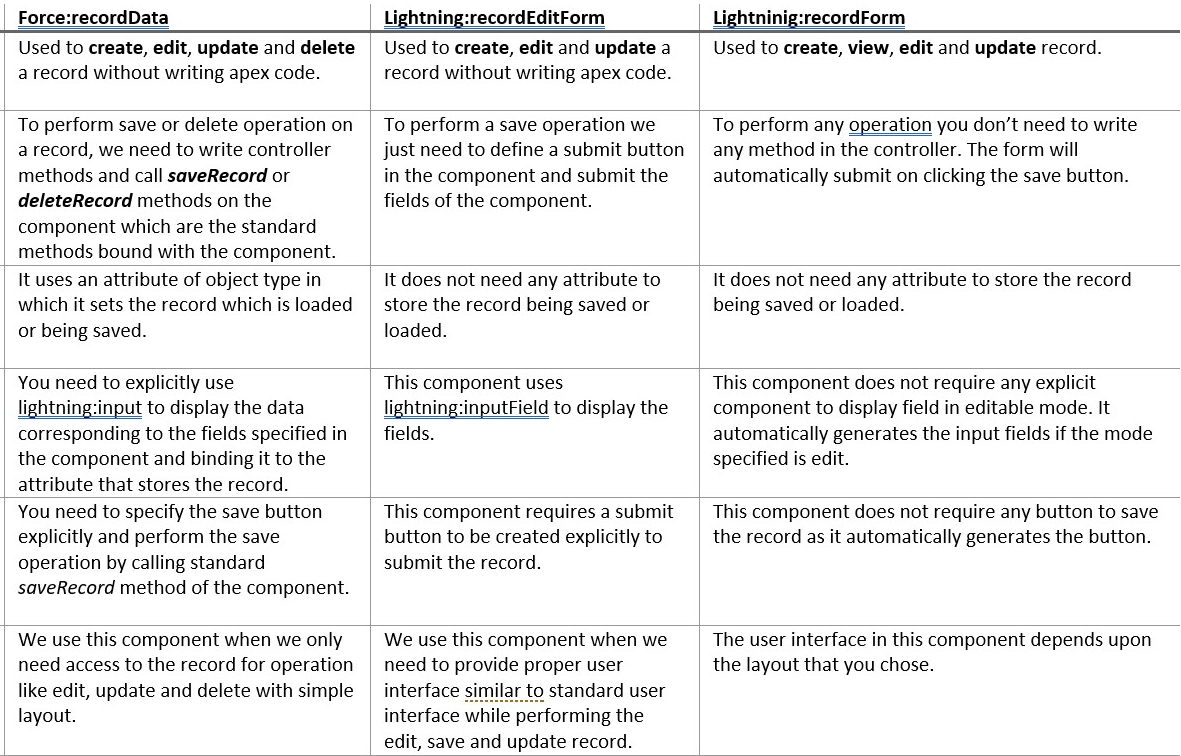
}

}

})

**Methods**

| **NAME** | **ARGUMENTS** | **ACCESS** | **DESCRIPTION** |
| --- | --- | --- | --- |
| getNewRecord | | **NAME** | **TYPE** | **DESCRIPTION** | | --- | --- | --- | | objectApiName | string | The object api name of the record. | | recordTypeId | string | If null, the default record type for the object will be used. | | skipCache | boolean | Force a trip to the server, skipping any local cache. | | callback | unknown | Callback function to be invoked after new record template is retrieved. | | global | Loads an empty record template into v.targetRecord which includes any predefined default values for the object and record type. |
| reloadRecord | | **NAME** | **TYPE** | **DESCRIPTION** | | --- | --- | --- | | skipCache | boolean | Force a trip to the server, skipping any local cache. | | callback | unknown | Callback function to be invoked after record is retrieved. | | global | Performs the same load function as on init, using current config values (recordId, layoutType, mode, etc) but does not force a server trip if not required. |
| saveRecord | | **NAME** | **TYPE** | **DESCRIPTION** | | --- | --- | --- | | callback | unknown | Callback function invoked after new Record is retrieved | | global | Saves the current record. |
| deleteRecord | | **NAME** | **TYPE** | **DESCRIPTION** | | --- | --- | --- | | callback | unknown | Callback function to be invoked after the record has been deleted. | | global | Deletes the current record. |



Also field labels will be derived from schema directly in form based components but in recordData labels are to be added manually.

**Limitations of Lightning Data Services**

1. Can be operated only on one record at a time. Can’t handle many records at a time.
2. One operation at a time. Multiple operations are not possible.
3. Doesn’t support all the objects. Ex – Task, Event etc. are not supported.
4. Lightning Data Service supports spanned fields with a maximum depth of five levels
5. Not supportive/available everywhere like classic and all.