Caching and Synchronizing Component Data with Lightning Data Service

Lightning Data Service is a declarative service that allows you to access, modify or create data in Salesforce with no need to write Apex code. The service provides a cache shared amongst all the components that use the service, and it is smart enough to notify those components about changes of the records performed through the service so that the components synchronize. It works both for Aura Components and Lightning Web Components. Want to know more? Continue reading.

Base form components in Aura and Lightning Web Components

In Lightning Web Components, several base components such as `lightning-record-form`, `lightning-record-edit-form` and `lightning-record-view-form` use the Lightning Data Service. These let you create custom forms in a declarative, quick and simple way without having to write Apex. The Aura version of these components are also available.

In the LWC Recipes sample application, there are examples of how to use these base components, as the `recordFormStaticContact`, `recordViewFormStaticContact` and `recordEditFormStaticContact` recipes. You can check their dynamic versions too (just a different way of specifying objects and field names). Check the documentation if you want to know more.

> You should use these components whenever possible. They are easier to build, they create a metadata-driven UI using just a few lines of code, and they fit in 99 percent of scenarios.

Lightning Data Service in Aura Components

Additionally, Lightning Data Service is surfaced in Aura through the `force:recordData` base component. Using this component it is possible to declaratively read, create and update records using the service. Its usage is recommended only if the base form components do not fit your needs.

In addition to providing data access, Lightning Data Service caches and synchronizes any data shared by the `lightning:record-XXX` base components, custom components that use `force:recordData`, and any standard forms used by Lightning.

Lightning Data Service in Lightning Web Components

In Lightning Web Components, there are several ways in which you can take advantage of the Lightning Data Service:

Reading data and metadata with @wire and wire adapters

When the Lightning Web Components programming model was released, we thought about simpler and more performant ways to access metadata, data and Apex. The result was the Wire Service.

You may find references to the Wire Service and the Lightning Data Service as if they were the same concept, but they are not. The Wire Service is a protocol with adapters as extension points, while Lightning Data Service defines several adapters for it.

The Wire Service is used in components with `@wire`. This is the syntax for the new JavaScript feature called decorators, which make it possible to annotate and modify classes and properties. The Wire Service provisions an immutable stream of data to a component.

Lightning Data Service's wire adapters support reading Salesforce data and metadata, from individual records and lists of records to object and layout schema.

Properties decorated with `@wire` become reactive properties, meaning the component DOM will be refreshed if the property changes value. This is the same behavior as with `@track` or `@api`.

The Lightning Data Service wire adapters use a client-side cache to improve performance. They invoke the server API only when a valid value is not in the cache.

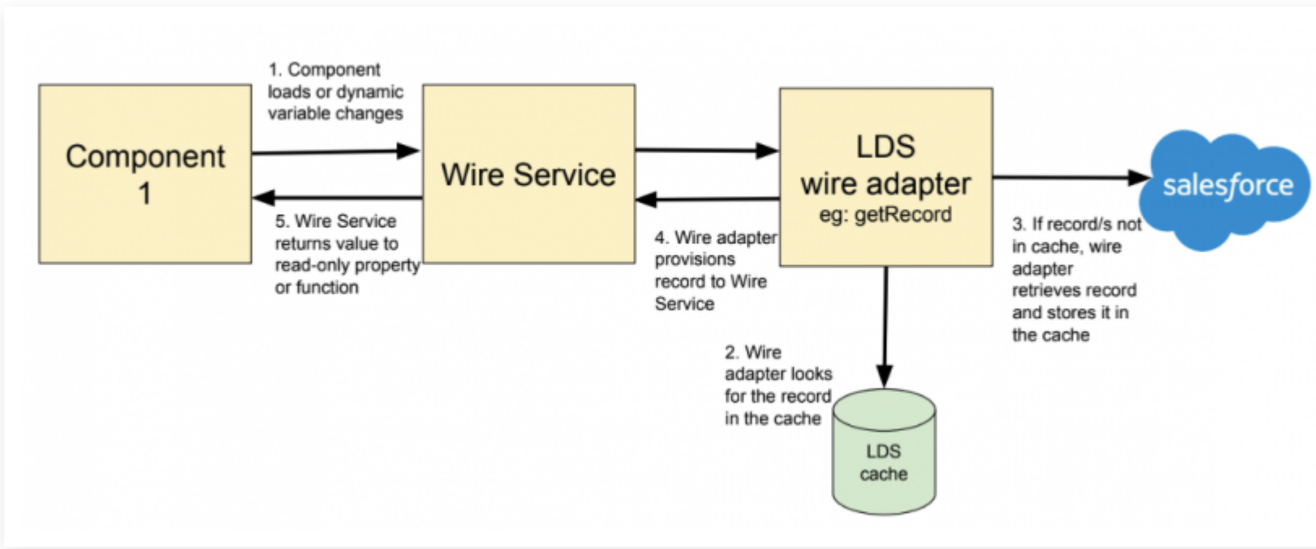
See a complete usage example here:

```
import { LightningElement, api, wire } from 'lwc';
import { getRecord } from 'lightning/uiRecordApi';
import ACCOUNT_NAME_FIELD from '@salesforce/schema/Account.Name';

export default class WireGetRecord extends LightningElement {
    @api recordId; // Get record Id from the record page in which the component is placed

    @wire(getRecord, { recordId: '$recordId', fields: [ACCOUNT_NAME_FIELD] })
    record;
}
```

Dynamic variables can be tied to the service prepending them with `$` (`$recordId` in the example). Each time a dynamic variable value changes, the Wire Service will provision new data to the component getting it through the Lightning Data Service wire adapter. The wire adapter will return that data either from the cache or the server.



As part of the Lightning Web Components library, there are several available out of the box wire adapters that expose Salesforce data and metadata. These wire adapters employ the User Interface API under the hood. This is the one that Salesforce uses to retrieve data and metadata from Lightning Experience, the Salesforce Mobile app and many other apps. It is important to know that this API enforces user-level access permissions like CRUD, Field Level Security, and Sharing. These are the available wire adapters as of Summer '19:

- lightning/uiRecordApi module:
    - `getRecord` – get a record's data.
    - `getRecordCreateDefaults` – get default field values for creating a record.
    - `getRecordUi` – get layout information, metadata, and data to build UI for one or more records. You really should not need this often, and instead look into using some of the Lightning Base Components like `lightning-record-XXX.`

- lightning/uiObjectInfoApi module:
    - `getObjectInfo` – get metadata about a specific object. The response includes metadata describing fields, record type, and theme.
    - `getPicklistValues` – get picklist values for a specific field.
    - `getPicklistValuesByRecordType` – get values for every picklist of a specified record type.
- lightning/uiListApi (Beta) module:
    - `getListUi (Beta)` – get records and metadata for a list view.

You can check more examples of how to use the `@wire` decorator to read data from some of these wire adapters in the `wireGetObjectInfo`, `wireGetPicklistValues` or `wireListView` recipes, from our LWC Recipes sample application.

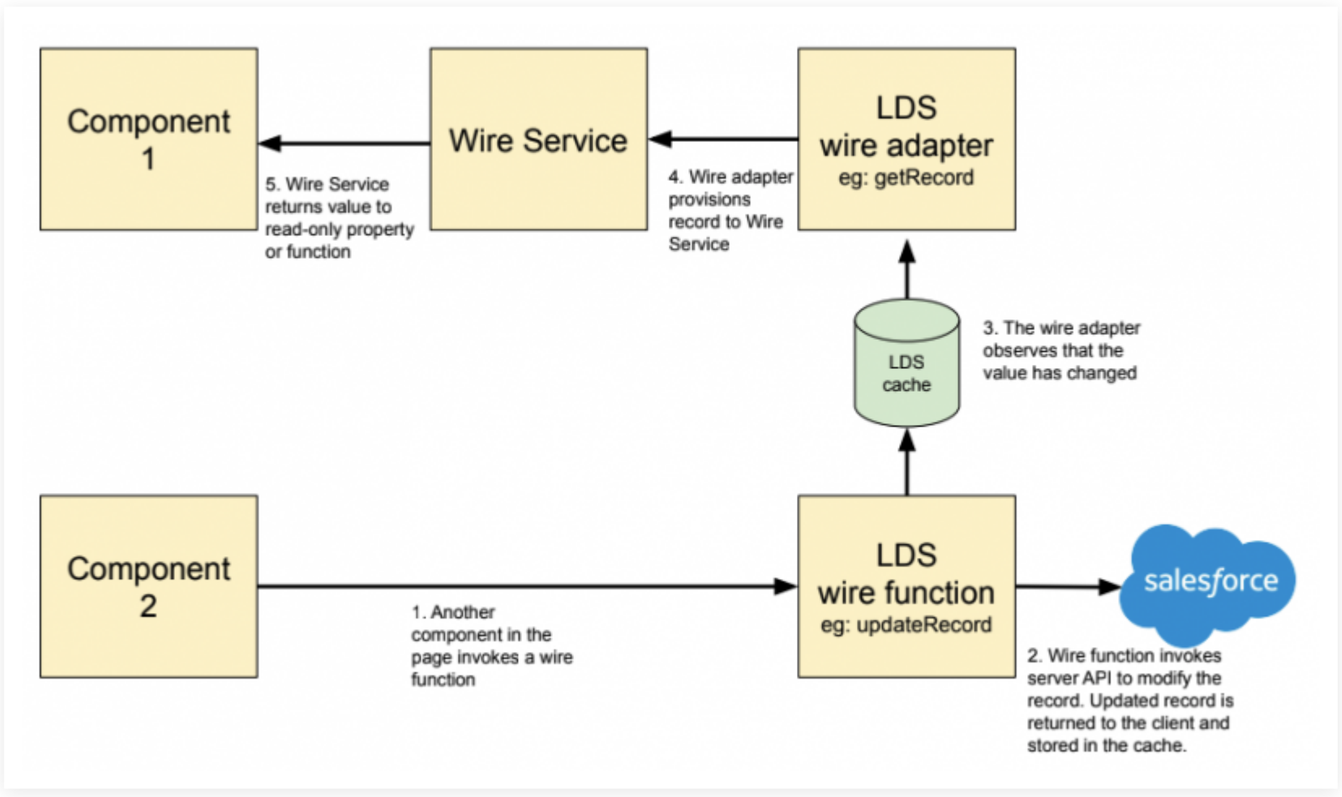
Creating, updating, and deleting data imperatively calling wire functions

In Lightning Web Components, records can be created, updated and deleted through Lightning Data Service too. In this case we don't use the `@wire` decorator, but we invoke wire functions imperatively. Note that anything decorated with `@wire` is read-only, because the receiver does not own the data, and the unidirectional data flow principles must be respected. Only read-only operations make sense. See an example of how to invoke a wire function to create a record here:

```
import { LightningElement, track } from 'lwc';
import { createRecord } from 'lightning/uiRecordApi';
import ACCOUNT_OBJECT from '@salesforce/schema/Account';
import NAME_FIELD from '@salesforce/schema/Account.Name';

export default class LdsCreateRecord extends LightningElement {
    createAccount() {
        const fields = {};
        fields[NAME_FIELD.fieldApiName] = 'ACME';
        const recordInput = { apiName: ACCOUNT_OBJECT.objectApiName, fields };
        // Invoke wire function imperatively
        createRecord(recordInput)
            .then(account => {
                // Do something on success
            })
            .catch(error => {
                // Do something on error
            });
    }
}
```

What the wire functions will do in this case, apart from the modification of records, is to notify the Lightning Data Service about the changes performed. This will force a refresh of other components that make use of the modified records, as they share the same underlying cache (the Lightning Data Service one).



Same as with wire adapters, several **out of the box wire functions** are available as of Summer '19:

- Module lightning/uiRecordApi:
    - `generateRecordInputForCreate(record)` – creates a record object filtered to contain fields editable by the current user (read-only fields are removed). Normally used to create a form so that the user can edit values before calling `createRecord(recordInput)`.
    - `createRecord(recordInput)` – creates a record.
    - `generateRecordInputForUpdate(record)` – creates a record object filtered to contain updateable fields. Normally used to determine the fields that are part of an edit form, so that the user can edit the record.
    - `createRecordInputFilteredByEditedFields(recordInput, originalRecord)` – creates a RecordInput object with a list of fields that have been edited from their original values to pass in a call to `updateRecord(recordInput)`. Use it after user has edited some values in an edit form.
    - `updateRecord(recordInput)` – updates a record.
    - `deleteRecord(recordId)` – deletes a record.
    - `getFieldValue(record, field)` – gets a field's value from a record. Spanning fields are supported. Using this function eliminates the possibility of using a wrong field name, as it is validated from the server in the compilation process.
    - `getFieldDisplayValue(record, field)` – gets the display value of a field. Spanning fields are supported. Using this function eliminates the possibility of using a wrong field name too.

In the LWC Recipes sample application, you can see some examples of how to call some of these wire functions, like the `ldsCreateRecord` or the `ldsDeleteRecord` recipes.

Any Lightning web components that use the Lightning Data Service will share the Lightning Data Service caching and synchronization features, no matter if it is a custom component built using a wire adapter or a wire function, or if it is a custom component that makes use of the base Lightning web components base forms. This includes the standard forms used by the current version of Lightning Experience, which have been rewritten to use Lightning Web Components behind the scenes.

All together

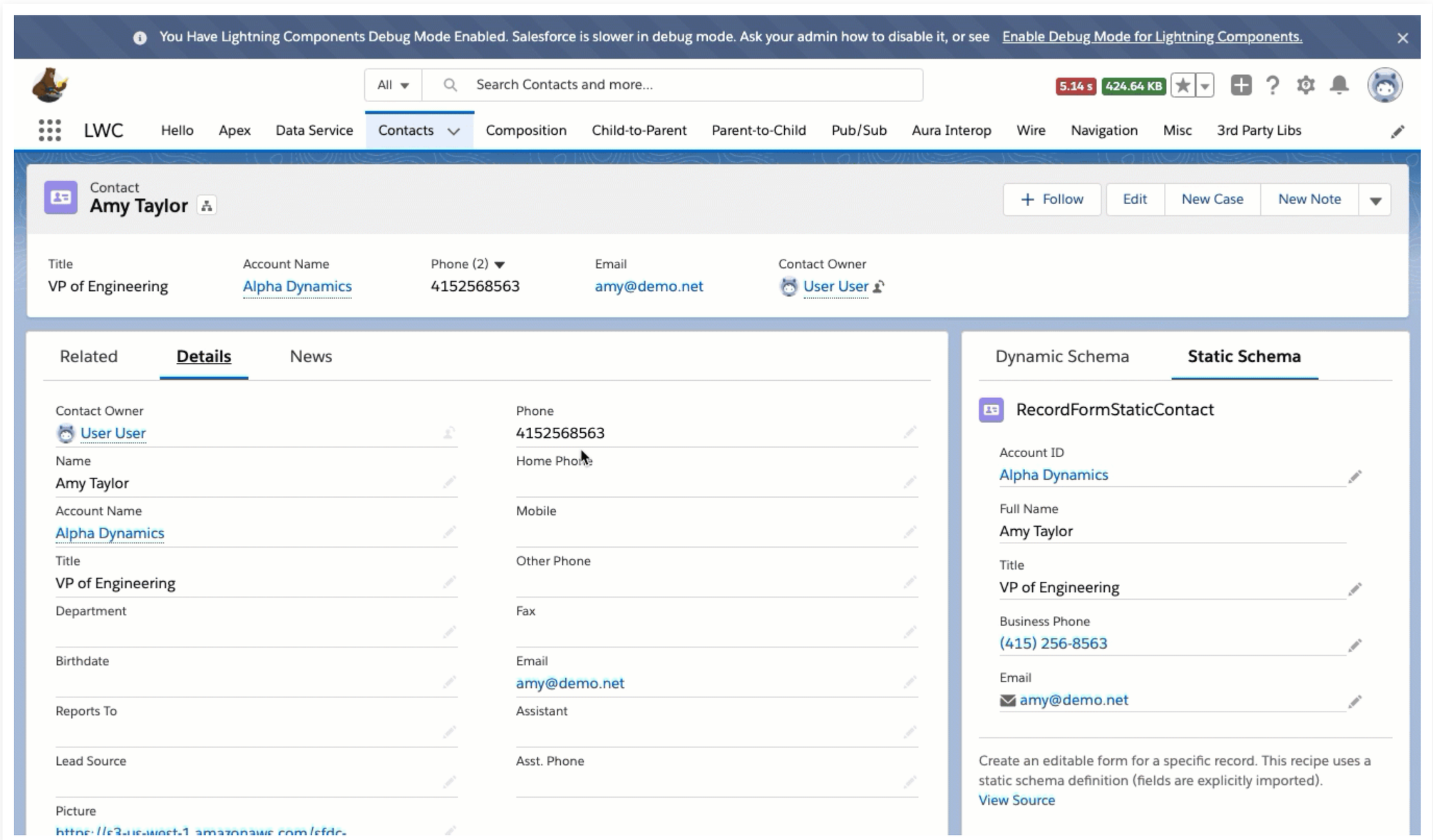
We have seen in which ways Lightning Data Service is used in Aura and in Lightning Web Components. But that is not all, because the Lightning Data Service layer is also **shared among Aura and Lightning Web Components**. This means the cache and synchronization features will be shared in all the use cases that we have mentioned along the blog, as long as the components belong to the same page and same user session.

Specifically, the Lightning Data Service cache will be shared by:

- The Lightning Experience app (for example, the standard Lightning record pages)
- Custom Aura components that use Base Aura forms
- Custom LWC components that use Base LWC forms
- Custom LWC components that use `@wire` to read data and metadata using wire adapters
- Custom LWC components that call wire functions imperatively

Note that `@wire` can be used to invoke Apex cacheable methods and perform read operations too, but the cache of these kind of invocations has nothing to do with the standard wire adapters, wire functions or base form components.

Do you want to see this in action? In the LWC Recipes sample application, you can navigate to the Contact record page, which includes recipes for most of the use cases mentioned in this blog. If you then try to change some contact information, you will see that the other components in the page are refreshed immediately, because of the Lightning Data Service being shared among all the components. Here's an animation that demonstrates this:



Conclusion

In this blog post, we have seen how the Lightning Data Service was implemented in Aura and has been reimagined and empowered thanks to the Wire Service in Lightning Web Components. We have learned how to make use of the service in both programming models, and how both Aura and Lightning Web Components share the caching and synchronization capabilities of the service.

Do you need to build a custom form? Follow these steps:

1. Try first with base form components. They are simple to use, maintained by Salesforce and they will spare you from writing Apex code.
    1. First try with `lightning-record-form.` It is the simplest one. Additionally, if a layout is specified, the form fields can be configured at any time by an admin in a declarative way (with no code modification).
    2. If it is not flexible enough, try with `lightning-record-view-form` or `lightning-record-edit-form`. They are more configurable than lightning-record-form but also have more complexity.
2. If you are not able to use any of the base form components, build a form from scratch:
    1. Use an available wire adapter for read operations.
    2. Use an available wire function for create, update and delete operations.
3. If you are not able to use one of the available wire adapters or wire functions, write:
    1. A cacheable Apex method and invoke it using `@wire` for read operations.
    2. A non-cacheable Apex method and invoke it imperatively for create, update and delete operations.

Bonus: At the moment we are working on adding more features to Lightning Data Service. In the future, we want to optimize how data is fetched and also add offline capabilities to it, provided some constraints are respected.

Now it is time to get your hands on the Lightning Data Service.