You can *listen* to a document with the `onSnapshot()` method. An initial call using the callback you provide creates a document snapshot immediately with the current contents of the single document. Then, each time the contents change, another call updates the document snapshot.

Realtime listeners are not supported in the PHP client library.

Local writes in your app will invoke snapshot listeners immediately. This is because of an important feature called "latency compensation." When you perform a write, your listeners will be notified with the new data *before* the data is sent to the backend.

Retrieved documents have a `metadata.hasPendingWrites` property that indicates whether the document has local changes that haven't been written to the backend yet. You can use this property to determine the source of events received by your snapshot listener:

When listening for changes to a document, collection, or query, you can pass options to control the granularity of events that your listener will receive.

By default, listeners are not notified of changes that only affect metadata. Consider what happens when your app writes a new document:

1. A change event is immediately fired with the new data. The document has not yet been written to the backend so the "pending writes" flag is `true`.
2. The document is written to the backend.
3. The backend notifies the client of the successful write. There is no change to the document data, but there is a metadata change because the "pending writes" flag is now `false`.

If you want to receive snapshot events when the document or query metadata changes, pass a listen options object when attaching your listener:

If you just want to know when your write has completed, you can listen to the completion callback rather than using `ndingWrites`. In JavaScript, use the `Promise` returned from your write operation by attaching a `.then()` callback. pass a completion callback to your write function.

As with documents, you can use `onSnapshot()` instead of `get()` to listen to the results of a query. This creates a query snapshot. For example, to listen to the documents with state `CA`:

The snapshot handler will receive a new query snapshot every time the query results change (that is, when a document is added, removed, or modified).

**ant:** As explained above under Events for local changes (#events-local-changes), you will receive events *immediately* ocal writes. Your listener can use the **`metadata.hasPendingWrites`** field on each document to determine whether t ment has local changes that have not yet been written to the backend.

It is often useful to see the actual changes to query results between query snapshots, instead of simply using the entire query snapshot. For example, you may want to maintain a cache as individual documents are added, removed, and modified.

**ant:** The first query snapshot contains **added** events for all existing documents that match the query. This is becaus getting a set of changes that bring your query snapshot current with the initial state of the query. This allows you, fo ce, to directly populate your UI from the changes you receive in the first query snapshot, without needing to add spec or handling the initial state.

The initial state can come from the server directly, or from a local cache. If there is state available in a local cache, the query snapshot will be initially populated with the cached data, then updated with the

server's data when the client has caught up with the server's state.

When you are no longer interested in listening to your data, you must detach your listener so that your event callbacks stop getting called. This allows the client to stop using bandwidth to receive updates. For example:

A listen may occasionally fail — for example, due to security permissions, or if you tried to listen on an invalid query. (Learn more about valid and invalid queries

(/firestore/docs/query-data/queries#compound_queries).) To handle these failures, you can provide an error callback when you attach your snapshot listener. After an error, the listener will not receive any more events, and there is no need to detach your listener.

- Combine listeners with simple and compound queries (/firestore/docs/query-data/queries).

- Order and limit the documents retrieved (/firestore/docs/query-data/order-limit-data).
- Understand billing for listeners (https://cloud.google.com/firestore/docs/pricing#operations).