Change Data Capture

With Change Data Capture, you can receive changes of Salesforce records in real time and synchronize corresponding records in an external data store.

Change Data Capture publishes events for changes in Salesforce records corresponding to create, update, delete, and undelete operations.

Object Support

Change Data Capture can generate change events for all custom objects defined in your Salesforce org and a subset of standard objects. It supports change events for the most popular standard objects including Account, Contact, Lead, User, Order, OrderItem, Product2, and others.

A Change Event Example

Structure of JSON of change event

i. **ChangeEventHandler**
   1. **entityName** : object api name
   2. **changeType :** UPDATE/INSERT/DELETE/UNDELETE
   3. **changedFields :** array of field Names
   4. **changeOrigin**
      a. /client= SfdcInternalAPI = internal salesforce action
      b. /client=clientName = some external action
   5. **transactionKey** : Use the transaction fields to maintain an accurate replica of your org's data in another system. The transactionKey field uniquely identifies the transaction that the change is part of.
   6. **sequenceNumber**: . The sequence number is useful for operations that include multiple steps. such as lead conversion or creating related records in an after insert Apex trigger. If not all objects involved in a transaction are enabled for Change Data Capture, there will be a gap in the sequence numbers.

ii. Custom Fields Changed – NA in case of deletion

iii. Standard Fields changed– NA in case of deletion

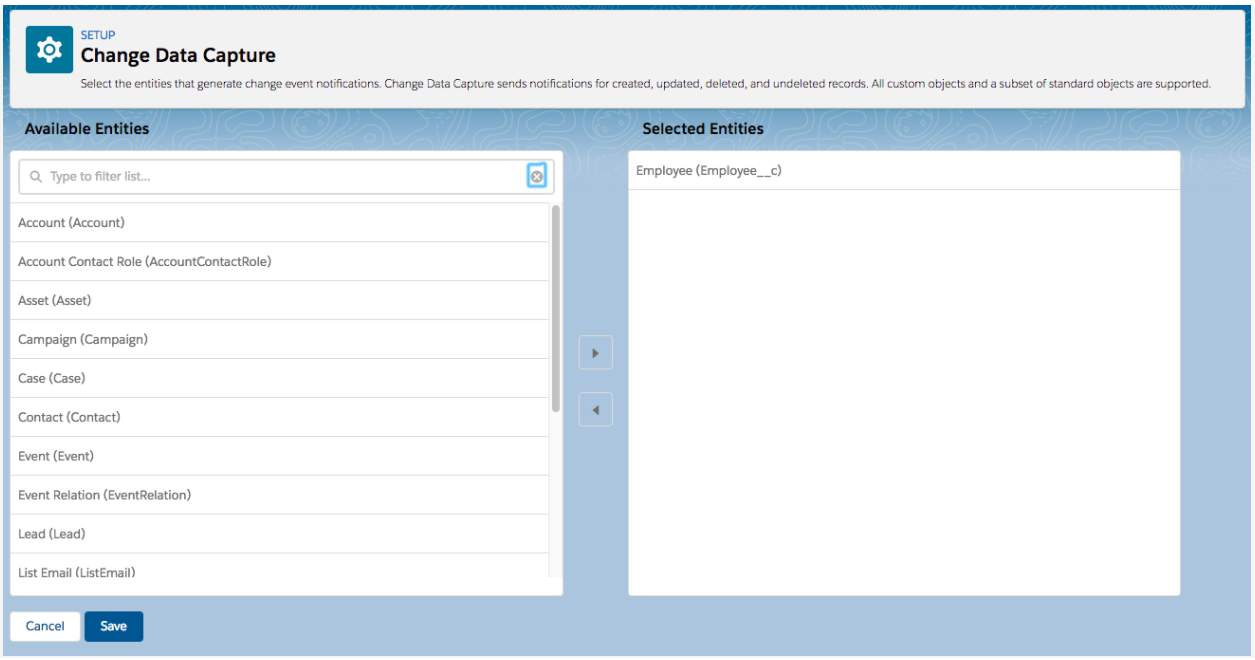
iv. **Event**:
   a. **replayId** : The last part of the event message contains a field called replayId. This field contains an ID for the event message that you can use to replay past events for up to 3 days.

Sample JSON


```
{
 "schema": "-pszPCNGMHqUPU1ftkjxEA",
 "payload": {
  "ChangeEventHeader": {
   "commitNumber": 65824495947,
   "commitUser": "005RM000001vI4mYAE",
   "sequenceNumber": 1,
   "entityName": "Employee__c",
   "changeType": "CREATE",
   "changedFields": [],
   "changeOrigin": "com/salesforce/api/soap/47.0;client=SfdcInternalAPI/",
   "transactionKey": "0005163c-8d04-d729-39bd-b917b035a66c",
   "commitTimestamp": 1569436136000,
   "recordIds": [
    "a00RM0000004ICOYA2"
   ]
  },
  "First_Name__c": "Jane",
  "Tenure__c": 2.0,
  "Name": "e-100",
  "Last_Name__c": "Smith",
  "CreatedDate": "2019-09-25T18:28:55.000Z",
  "LastModifiedDate": "2019-09-25T18:28:55.000Z",
  "OwnerId": "005RM000001vI4mYAE",
  "CreatedById": "005RM000001vI4mYAE",
  "LastModifiedById": "005RM000001vI4mYAE",
 },
 "event": {
  "replayId": 1
 }
}
```


Subscribe Using Event channel

1. From Setup, enter `Change Data Capture` in the Quick Find box, and click **Change Data Capture**.
2. In Available Entities, select **Account** and click the **>** arrow.
3. Click **Save**.



Subscribe Using Apex trigger

```
trigger TriggerName on ChangeEventName (after insert) {

}
```

- Only after insert triggers are supported.
- change event triggers run asynchronously after the database transaction is completed
- They run under the Automated Process entity.
- They are subject to Apex synchronous governor limits.
- They have a maximum batch size of 2,000 event messages (the number of items in Trigger.New).

Testing the Change Event Trigger

Test.enableChangeDataCapture() –

1. Should be the very first line of test method
2. Enables all entities change data capture for test context only.

Test.getEventBus().deliver() – The method delivers the event messages from the test event bus to the corresponding change event trigger and causes the trigger to fire.

Sample trigger

```
trigger EmployeeChangeTrigger on Employee__ChangeEvent (after insert) {
  List<Task> tasks = new List<Task>();
  // Iterate through each event message.
  for (Employee__ChangeEvent event : Trigger.New) {
    // Get some event header fields
    EventBus.ChangeEventHeader header = event.ChangeEventHeader;
    System.debug('Received change event for ' +
      header.entityName +
      ' for the ' + header.changeType + ' operation.');
    // For update operations, we can get a list of changed fields
    if (header.changetype == 'UPDATE') {
       System.debug('List of all changed fields:');
       for (String field : header.changedFields) {
          if (null == event.get(field)) {
             System.debug('Deleted field value (set to null): ' + field);
          } else {
             System.debug('Changed field value: ' + field + '. New Value: '
                + event.get(field));
          }
       }
    }
    // Get record fields and display only if not null.
    System.debug('Some Employee record field values from the change event:');
    if (event.First_Name__c != null) {
      System.debug('First Name: ' + event.First_Name__c);
    }
    if (event.Last_Name__c != null) {
      System.debug('Last Name: ' + event.Last_Name__c);
    }
    if (event.Name != null) {
      System.debug('Name: ' + event.Name);
    }
    if (event.Tenure__c != null) {
      System.debug('Tenure: ' + event.Tenure__c);
    }
    // Create a followup task
    Task tk = new Task();
    tk.Subject = 'Follow up on employee record(s): ' +
    header.recordIds;
    tk.OwnerId = header.CommitUser;
    tasks.add(tk);
  }
  // Insert all tasks in bulk.
```

```
  if (tasks.size() > 0) {
    insert tasks;
  }
}
```

Sample Test class

```
@isTest
public class TestEmployeeChangeTrigger {
    @isTest static void testCreateAndUpdateEmployee() {
        // Enable all Change Data Capture entities for notifications.
        Test.enableChangeDataCapture();
        // Insert an Employee test record
        insert new Employee__c(Name='e-101',
            First_Name__c='Astro',
            Last_Name__c='Test',
            Tenure__c=1);
        // Call deliver to fire the trigger and deliver the test change event.
        Test.getEventBus().deliver();
        // VERIFICATIONS
        // Check that the change event trigger created a task.
        Task[] taskList = [SELECT Id,Subject FROM Task];
        System.assertEquals(1, taskList.size(),
            'The change event trigger did not create the expected task.');
        // Update employee record
        Employee__c[] empRecords = [SELECT Id,OwnerId,First_Name__c,Tenure__c FROM Employee__c];
        // There is only one test record, so get the first one
        Employee__c emp = empRecords[0];
        // Debug
        System.debug('Retrieved employee record: ' + emp);
        // Update one field and empty another
        emp.First_Name__c = 'Codey';
        emp.Tenure__c = null;
        update emp;
        // Call deliver to fire the trigger for the update operation.
        Test.getEventBus().deliver();
        // VERIFICATIONS
        // Check that the change event trigger created a task.
        // We should have two tasks now, including one from the first trigger invocation.
        Task[] taskList2 = [SELECT Id,Subject FROM Task];
        System.assertEquals(2, taskList2.size(),
            'The change event trigger did not create the expected task.');
    }
}
```