**Custom Settings**

Custom settings are similar to custom objects. Application developers can create custom sets of data and associate custom data for an organization, profile, or specific user. All custom settings data is exposed in the application cache, which enables efficient access without the cost of repeated queries to the database. Formula fields, validation rules, flows, Apex, and SOAP API can then use this data.

# **Types of Custom settings**

## **List Custom Settings**

A type of custom setting that provides a reusable set of static data that can be accessed across your organization. If you use a particular set of data frequently within your application, putting that data in a list custom setting streamlines access to it. Data in list settings does not vary with profile or user, but is available organization-wide. Examples of list data include two-letter state abbreviations, international dialing prefixes, and catalog numbers for products. Because the data is cached, access is low-cost and efficient: you don't have to use SOQL queries that count against your governor limits.

## **Hierarchy Custom Settings**

A type of custom setting that uses a built-in hierarchical logic that lets you “personalize” settings for specific profiles or users. The hierarchy logic checks the organization, profile, and user settings for the current user and returns the most specific, or “lowest,” value. In the hierarchy, settings for an organization are overridden by profile settings, which, in turn, are overridden by user settings.

Reference: <https://youtu.be/zxQu0Ozo4G0>

<https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_customsettings.htm>

**Suppose Country Phone Codes is a custom setting in my org. How to fetch all the records of custom setting?**

A: It is very easy as we do not need a SOQL query.

List<Country\_Phone\_Code\_\_c> phoneCodes = Country\_Phone\_code\_\_c.**getAll().values()**;

getAll() -> will fetch the map of custom setting records.

Values() -> will fetch the values from that map.

**To fetch a Single record:**

Country\_Phone\_Code\_\_c phoneCode = Country\_Phone\_code\_\_c.**getInstance**(‘India’);

getInstance(‘Key’) -> will fetch one record for the specified key.

For Hierarchical Custom Settings we have one more method which is getValues(userid/profileId)

Country\_Phone\_code\_\_c phoneCode = Country\_phone\_code\_\_c.**getValues**(userinfo.getUserId());

getValues(userid/profileId) -> this will fetch the records for which the provided user/profile has access to.

Custom Settings vs Custom Objects

Custom Settings are stored in the Cache Memory whereas Custom Objects are stored in Force.com Database.

Custom Settings will have only limited Data types, but custom Objects have large variety of data types.

Custom Settings cannot have Relationships like MD or Lookup but custom objects can have.

Custom Settings cannot have its own custom tab in app but a custom object can have its custom tab in app.

No formula fields or RollUp summary in Custom Settings. But in custom Objects we have.

No triggers, Validation Rules, Workflow rules, Processes, Approval Processes, Flows for Custom Settings, but for custom objects we have all those.

Custom Settings removes chances of hitting governor limits as we do not need SOQL queries for fetching. But Custom Objects can easily hit governor limits as we need to use SOQL queries to fetch records.

**Important Points:**

1. We can use SOQL queries in Custom Settings but it is not at all recommended bcoz we have dedicated methods like getAll() or getInstance() to fetch the values of custom settings.
2. Fast access of data is there in custom settings as data is stored in cache memory.
3. Hierarchical Custom settings allows different users/profiles have different data.
4. Custom Settings data **cannot** be deployed.
5. Only 300 fields are allowed in Custom Settings.
6. Tabs cannot be created for custom settings.

**Sharing in Apex**

By default Apex code runs on system mode or system context. This means that code can access all objects and fields, object permissions, Field-Level Security (FLS). Sharing rules are not applied for the current user. (without Sharing).

## **With Sharing**

It means that sharing rules will be enforced for the current/running user.

## **Without Sharing**

It means that sharing rules will not be enforced for the current/running user.

**Sharing setting of a class where method is defined is applied, not the class where the method is called.**

For Ex: Suppose a Class(A) with Sharing has method A and there is another class(B) without sharing has a method B. Now if Class B calls the method A, then the method A will execute in with Sharing context because method A was defined in the with Sharing class.

1. If a class is declared as “with sharing” then, the sharing settings apply to all code contained in the class, including initialization code, constructors and methods.
2. Inner class do not inherit the sharing settings from container class.
3. Classes inherit sharing setting from Parent class when a Class extends another class. Meaning that child class will inherit the sharing setting of the parent class.
4. If you run a class in execute Anonymous window then it will always enforce the with Sharing settings.
5. If a class is not specified with ‘with Sharing’ or ‘without sharing ’ then by default it will be in System Mode(without Sharing). Now, If a class with Sharing calls this class with no specified sharing setting then the called class(class with no specific sharing setting) will be executed in ‘With Sharing’ context.

**PFB Important Scenarios**

**Scenarios:**

### **Public with sharing class A { }**

### **Public without Sharing class B { }**

### **Public class C { } // No specified sharing >>>>>>> By default without Sharing**

## **Scenario 1:**

**Class B extends Class A**

Now Class B will inherit the sharing mode of Parent. So ,class B will now be following with Sharing.

## **Scenario 2:**

**Class A extends Class B**

Now Class A will inherit the sharing mode of Parent. In this case it will be without sharing.

## **Scenario 3:**

**Class C extends Class A**

Now Class C will inherit the sharing mode of the Parent.in this case it will be With Sharing.

## **Scenario 4:**

**Class B calls Class A:**

Since Sharing context will be depending upon where the method was defined. So, in this case the sharing context will be With Sharing because Class A has a with Sharing context.

## **Scenario 5:**

**Class C Calls Class A:**

Called method of class A, so method will follow the Class A’s sharing context. So, it will be with Sharing.

## **Scenario 6:**

**Class A calls Class C:**

Since C does not have any specified Sharing context here, So the Sharing context of A will be followed. So, in this case the sharing context will be With Sharing.

## **Scenario 7:**

**Class A extends Class C:**

Two cases kicks in here.

1. **Public class A with Sharing extends Class C:**

In this case, Since class C is in System Mode(as specified at top of this page), so A will follow its sharing context, which is with Sharing. So, sharing context will be With Sharing.

1. **Public class A extends Class C:**

In this case, since class A and C both are in System Mode. Class A will follow Parent Class’s sharing context. So, it will be without Sharing.

## **Scenario 8:**

**Class A calls Class B:**

Since the method follows the Sharing context of the class where it is defined. So, in this case it will be without Sharing.