Custom Settings Methods

Custom settings are similar to custom objects and enable application developers to create custom sets of data, as well as create 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. This data is then available for formula fields, validation rules, flows, Apex, and the SOAP API.

Usage

Custom settings methods are all instance methods, that is, they are called by and operate on a specific instance of a custom setting. There are two types of custom settings: hierarchy and list. There are two types of methods: methods that work with list custom settings, and methods that work with hierarchy custom settings.

> **Note**
>
> All custom settings data is exposed in the application cache, which enables efficient access without the cost of repeated queries to the database. However, querying custom settings data using Standard Object Query Language (SOQL) doesn't use the application cache and is similar to querying a custom object. To benefit from caching, use other methods for accessing custom settings data such as the Apex Custom Settings methods.

For more information on creating custom settings in the Salesforce user interface, see “Create Custom Settings” in the Salesforce online help.

Custom Setting Examples

The following example uses a list custom setting called `Games`. The `Games` setting has a field called `GameType`. This example determines if the value of the first data set is equal to the string `PC`.

```
List<Games__C> mcs = Games__c.getall().values();
boolean textField = null;
if (mcs[0].GameType__c == 'PC') {
  textField = true;
}
system.assertEquals(textField, true);
```

The following example uses a custom setting called `Foundation_Countries`. This example demonstrates that the `getValues` and `getInstance` methods return identical values.

```
Foundation_Countries__c myCS1 = Foundation_Countries__c.getValues('United States');
String myCCVal = myCS1.Country_code__c;
Foundation_Countries__c myCS2 = Foundation_Countries__c.getInstance('United States');
String myCCInst = myCS2.Country_code__c;
system.assertEquals(myCCinst, myCCVal);
```

Hierarchy Custom Setting Examples

```
GamesSupport__c mhc = GamesSupport__c.getInstance(pid);
string mPhone = mhc.Corporate_number__c;
```

The example is identical if you choose to use the `getValues` method.

The following example shows how to use hierarchy custom settings methods. For `getInstance`, the example shows how field values that aren't set for a specific user or profile are returned from fields defined at the next lowest level in the hierarchy. The example also shows how to use `getOrgDefaults`.

Finally, the example demonstrates how `getValues` returns fields in the custom setting record only for the specific user or profile, and doesn't merge values from other levels of the hierarchy. Instead, `getValues` returns `null` for any fields that aren't set. This example uses a hierarchy custom setting called Hierarchy. Hierarchy has two fields: `OverrideMe` and `DontOverrideMe`. In addition, a user named Robert has a System Administrator profile. The organization, profile, and user settings for this example are as follows:

Organization settings

`OverrideMe`: Hello

`DontOverrideMe`: World

Profile settings

`OverrideMe`: Goodbye

`DontOverrideMe` is not set.

User settings

`OverrideMe`: Fluffy

`DontOverrideMe` is not set.

The following example demonstrates the result of the `getInstance` method when Robert calls it in his organization:

```
Hierarchy__c CS = Hierarchy__c.getInstance();
System.Assert(CS.OverrideMe__c == 'Fluffy');
System.assert(CS.DontOverrideMe__c == 'World');
```

If Robert passes his user ID specified by `RobertId` to `getInstance`, the results are the same. The identical results are because the lowest level of data in the custom setting is specified at the user level.

```
Hierarchy__c CS = Hierarchy__c.getInstance(RobertId);
System.Assert(CS.OverrideMe__c == 'Fluffy');
System.assert(CS.DontOverrideMe__c == 'World');
```

If Robert passes the System Administrator profile ID specified by `SysAdminID` to `getInstance`, the result is different. The data specified for the profile is returned:

```
Hierarchy__c CS = Hierarchy__c.getInstance(SysAdminID);
System.Assert(CS.OverrideMe__c == 'Goodbye');
System.assert(CS.DontOverrideMe__c == 'World');
```

When Robert tries to return the data set for the organization using `getOrgDefaults`, the result is:

user and profile settings. For example, if Robert passes his user ID `RobertId` to `getValues`, the result is:

```
Hierarchy__c CS = Hierarchy__c.getValues(RobertId);
System.Assert(CS.OverrideMe__c == 'Fluffy');
// Note how this value is null, because you are returning
// data specific for the user
System.assert(CS.DontOverrideMe__c == null);
```

If Robert passes his System Administrator profile ID `SysAdminID` to `getValues`, the result is:

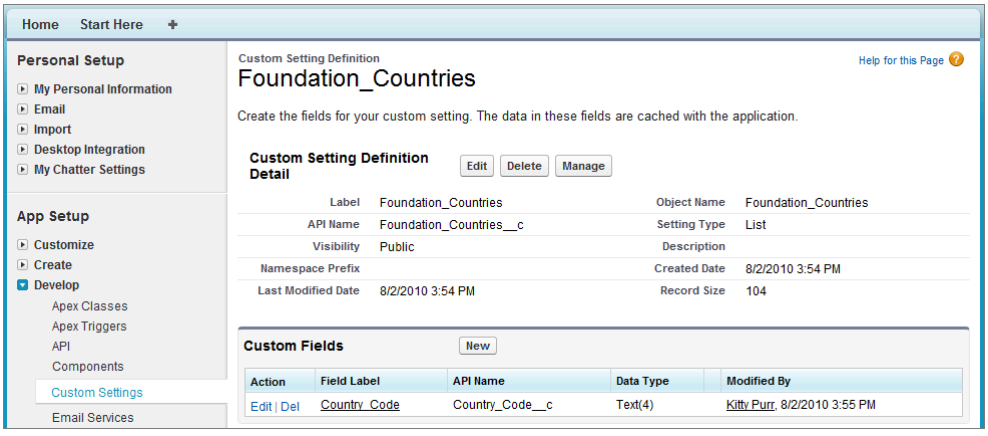
```
Hierarchy__c CS = Hierarchy__c.getValues(SysAdminID);
System.Assert(CS.OverrideMe__c == 'Goodbye');
// Note how this value is null, because you are returning
// data specific for the profile
System.assert(CS.DontOverrideMe__c == null);
```

Country and State Code Custom Settings Example

This example illustrates using two custom setting objects for storing related information, and a Visualforce page to display the data in a set of related picklists.

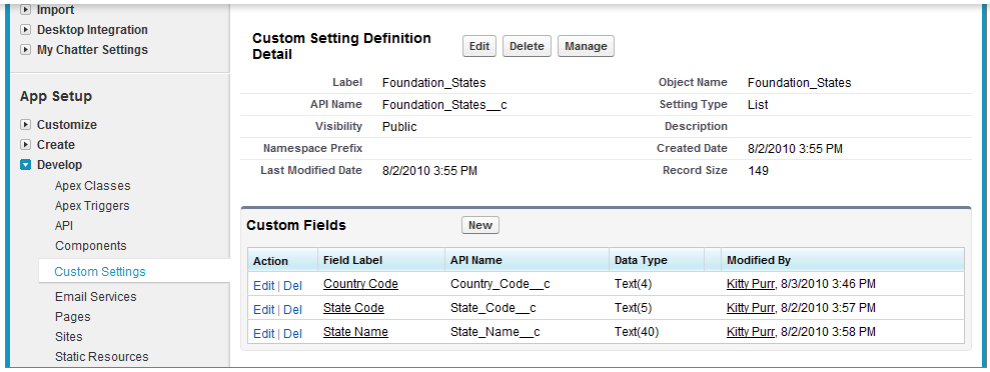
In the following example, country and state codes are stored in two different custom settings: Foundation_Countries and Foundation_States.

The Foundation_Countries custom setting is a list type custom setting and has a single field, `Country_Code`.

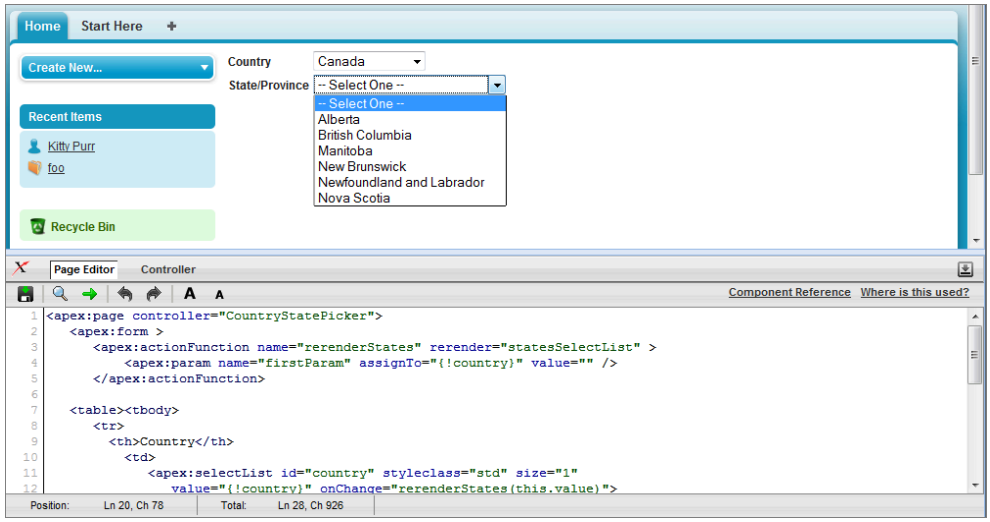


The Foundation_States custom setting is also a List type of custom setting and has the following fields:

- `Country Code`
- `State Code`
- `State Name`



The Visualforce page shows two picklists: one for country and one for state.




```
<apex:page controller="CountryStatePicker">
   <apex:form >
      <apex:actionFunction name="rerenderStates" rerender="statesSelectList" >
          <apex:param name="firstParam" assignTo="{!country}" value="" />
      </apex:actionFunction>

   <table><tbody>
      <tr>
        <th>Country</th>
          <td>
             <apex:selectList id="country" styleclass="std" size="1"
                value="{!country}" onChange="rerenderStates(this.value)">
                    <apex:selectOptions value="{!countriesSelectList}"/>
             </apex:selectList>
          </td>
      </tr>
      <tr id="state_input">
        <th>State/Province</th>
          <td>
            <apex:selectList id="statesSelectList" styleclass="std" size="1"
                 value="{!state}">
                   <apex:selectOptions value="{!statesSelectList}"/>
            </apex:selectList>
          </td>
      </tr>
   </tbody></table>
   </apex:form>
</apex:page>
```

```
public with sharing class CountryStatePicker {

// Variables to store country and state selected by user
    public String state { get; set; }
    public String country {get; set;}

    // Generates country dropdown from country settings
    public List<SelectOption> getCountriesSelectList() {
        List<SelectOption> options = new List<SelectOption>();
        options.add(new SelectOption('', '-- Select One --'));

        // Find all the countries in the custom setting
        Map<String, Foundation_Countries__c> countries = Foundation_Countries__c.getAll();

        // Sort them by name
        List<String> countryNames = new List<String>();
        countryNames.addAll(countries.keySet());
        countryNames.sort();

        // Create the Select Options.
        for (String countryName : countryNames) {
            Foundation_Countries__c country = countries.get(countryName);
            options.add(new SelectOption(country.country_code__c, country.Name));
        }
        return options;
    }

    // To generate the states picklist based on the country selected by user.
    public List<SelectOption> getStatesSelectList() {
        List<SelectOption> options = new List<SelectOption>();
        // Find all the states we have in custom settings.
        Map<String, Foundation_States__c> allstates = Foundation_States__c.getAll();

        // Filter states that belong to the selected country
        Map<String, Foundation_States__c> states = new Map<String, Foundation_States__c>()
        for(Foundation_States__c state : allstates.values()) {
            if (state.country_code__c == this.country) {
                states.put(state.name, state);
            }
        }

        // Sort the states based on their names
        List<String> stateNames = new List<String>();
        stateNames.addAll(states.keySet());
        stateNames.sort();

        // Generate the Select Options based on the final sorted list
        for (String stateName : stateNames) {
            Foundation_States__c state = states.get(stateName);
            options.add(new SelectOption(state.state_code__c, state.state_name__c));
        }

        // If no states are found, just say not required in the dropdown.
        if (options.size() > 0) {
            options.add(0, new SelectOption('', '-- Select One --'));
        } else {
            options.add(new SelectOption('', 'Not Required'));
        }
        return options;
    }
}
```

- List Custom Setting Methods
- Hierarchy Custom Setting Methods

See Also

- *Apex Developer Guide*: Custom Settings

Returns a map of the data sets defined for the custom setting.

- **getInstance(dataSetName)**
  Returns the custom setting data set record for the specified data set name. This method returns the exact same object as `getValues(dataSetName)`.
- **getValues(dataSetName)**
  Returns the custom setting data set record for the specified data set name. This method returns the exact same object as `getInstance(dataSetName)`.

getAll()

Returns a map of the data sets defined for the custom setting.

Signature

```
public Map<String, CustomSetting__c> getAll()
```

Return Value

Type: Map<String, CustomSetting__c>

Usage

If no data set is defined, this method returns an empty map.

> **Note**
>
> For Apex saved using Salesforce API version 20.0 or earlier, the data set names, which are the keys in the returned map, are converted to lower case. For Apex saved using Salesforce API version 21.0 and later, the case of the data set names in the returned map keys is not changed and the original case is preserved.

getInstance(dataSetName)

Returns the custom setting data set record for the specified data set name. This method returns the exact same object as `getValues(dataSetName)`.

Signature

```
public CustomSetting__c getInstance(String dataSetName)
```

Parameters

dataSetName

Type: String

Return Value

Type: CustomSetting__c

Usage

If no data is defined for the specified data set, this method returns `null`.

getValues(dataSetName)

Returns the custom setting data set record for the specified data set name. This method returns the exact same object as `getInstance(dataSetName)`.

Signature

Type: String

Return Value

Type: CustomSetting__c

Usage

If no data is defined for the specified data set, this method returns `null`.

Hierarchy Custom Setting Methods

The following are instance methods for hierarchy custom settings.

> **Note**
>
> - In API version 41.0 and below, each method in an Apex test class, including `testSetup` methods, are able to insert hierarchy custom setting values. This behavior is true even when the methods have the same `SetupOwnerId` value as a hierarchy custom setting record inserted in a different test method.
> - In API version 42.0 and later, if a hierarchy custom setting is inserted in a `testSetup` method, inserting a hierarchy custom setting record with the same `SetupOwnerId` in a test method throws a `DUPLICATE_VALUE` exception.

- **getInstance()**
  Returns a custom setting data set record for the current user. The fields returned in the custom setting record are merged based on the lowest level fields that are defined in the hierarchy.
- **getInstance(userId)**
  Returns the custom setting data set record for the specified user ID. The lowest level custom setting record and fields are returned. Use this when you want to explicitly retrieve data for the custom setting at the user level.
- **getInstance(profileId)**
  Returns the custom setting data set record for the specified profile ID. The lowest level custom setting record and fields are returned. Use this when you want to explicitly retrieve data for the custom setting at the profile level.
- **getOrgDefaults()**
  Returns the custom setting data set record for the organization.
- **getValues(userId)**
  Returns the custom setting data set record for the specified user ID.
- **getValues(profileId)**
  Returns the custom setting data set for the specified profile ID.

getInstance()

Returns a custom setting data set record for the current user. The fields returned in the custom setting record are merged based on the lowest level fields that are defined in the hierarchy.

Signature

```
public CustomSetting__c getInstance()
```

Return Value

Type: CustomSetting__c

Usage

except for the SetupOwnerId field which contains the user ID.

> **Note**
>
> For Apex saved using Salesforce API version 21.0 or earlier, this method returns the custom setting data set record with fields merged from field values defined at the lowest hierarchy level, starting with the user. Also, if no custom setting data is defined in the hierarchy, this method returns `null`.

This method is equivalent to a method call to `getInstance(User_Id)` for the current user.

Example

- Custom setting data set defined for the user: If you have a custom setting data set defined for the user "Uriel Jones," for the profile "System Administrator," and for the organization as a whole, and the user running the code is Uriel Jones, this method returns the custom setting record defined for Uriel Jones.
- Merged fields: If you have a custom setting data set with fields A and B for the user "Uriel Jones" and for the profile "System Administrator," and field A is defined for Uriel Jones, field B is `null` but is defined for the System Adminitrator profile, this method returns the custom setting record for Uriel Jones with field A for Uriel Jones and field B from the System Administrator profile.
- No custom setting data set record defined for the user: If the current user is "Barbara Mahonie," who also shares the "System Administrator" profile, but no data is defined for Barbara as a user, this method returns a new custom setting record with the ID set to `null` and with fields merged based on the fields defined in the lowest level in the hierarchy.

getInstance(userId)

Returns the custom setting data set record for the specified user ID. The lowest level custom setting record and fields are returned. Use this when you want to explicitly retrieve data for the custom setting at the user level.

Signature

```
public CustomSetting__c getInstance(ID userId)
```

Parameters

userId

Type: ID

Return Value

Type: CustomSetting__c

Usage

If no custom setting data is defined for the user, this method returns a new custom setting object. The new custom setting object contains an ID set to `null` and merged fields from higher in the hierarchy. You can add this new custom setting record for the user by using `insert` or `upsert`. If no custom setting data is defined in the hierarchy, the returned custom setting has empty fields, except for the SetupOwnerId field which contains the user ID.

> **Note**
>
> For Apex saved using Salesforce API version 21.0 or earlier, this method returns the custom setting data set record with fields merged from field values defined at the lowest hierarchy level, starting with the user. Also, if no custom setting data is defined in the hierarchy, this method returns `null`.

Returns the custom setting data set record for the specified profile ID. The lowest level custom setting record and fields are returned. Use this when you want to explicitly retrieve data for the custom setting at the profile level.

Signature

```
public CustomSetting__c getInstance(ID profileId)
```

Parameters

profileId

Type: ID

Return Value

Type: CustomSetting__c

Usage

If no custom setting data is defined for the profile, this method returns a new custom setting record. The new custom setting object contains an ID set to `null` and with merged fields from your organization's default values. You can add this new custom setting for the profile by using `insert` or `upsert`. If no custom setting data is defined in the hierarchy, the returned custom setting has empty fields, except for the `SetupOwnerId` field which contains the profile ID.

> **Note**
>
> For Apex saved using SalesforceAPI version 21.0 or earlier, this method returns the custom setting data set record with fields merged from field values defined at the lowest hierarchy level, starting with the profile. Also, if no custom setting data is defined in the hierarchy, this method returns `null`.

getOrgDefaults()

Returns the custom setting data set record for the organization.

Signature

```
public CustomSetting__c getOrgDefaults()
```

Return Value

Type: CustomSetting__c

Usage

If no custom setting data is defined for the organization, this method returns an empty custom setting object.

> **Note**
>
> For Apex saved using Salesforce API version 21.0 or earlier, this method returns `null` if no custom setting data is defined for the organization.

getValues(userId)

Returns the custom setting data set record for the specified user ID.

Signature

Type: ID

Return Value

Type: CustomSetting__c

Usage

Use this if you only want the subset of custom setting data that has been defined at the user level. For example, suppose you have a custom setting field that has been assigned a value of "alpha" at the organizational level, but has no value assigned at the user or profile level. Using `getValues(UserId)` returns `null` for this custom setting field.

getValues(profileId)

Returns the custom setting data set for the specified profile ID.

Signature

```
public CustomSetting__c getValues(ID profileId)
```

Parameters

profileId

Type: ID

Return Value

Type: CustomSetting__c

Usage

Use this if you only want the subset of custom setting data that has been defined at the profile level. For example, suppose you have a custom setting field that has been assigned a value of "alpha" at the organizational level, but has no value assigned at the user or profile level. Using `getValues(ProfileId)` returns `null` for this custom setting field.