SObject Class

Contains methods for the sObject data type.

Namespace

System

Usage

SObject methods are all instance methods: they are called by and operate on an sObject instance such as an account or contact. The following are the instance methods for sObjects.

For more information on sObjects, see Working with sObjects.

SObject Methods

The following are methods for `SObject`. All are instance methods.

- **addError(errorMsg)**
  Marks a trigger record with a custom error message and prevents any DML operation from occurring.
- **addError(errorMsg, escape)**
  Marks a trigger record with a custom error message, specifies if the error message should be escaped, and prevents any DML operation from occurring.
- **addError(exceptionError)**
  Marks a trigger record with a custom error message and prevents any DML operation from occurring.
- **addError(exceptionError, escape)**
  Marks a trigger record with a custom exception error message, specifies whether or not the exception error message should be escaped, and prevents any DML operation from occurring.
- **addError(errorMsg)**
  Places the specified error message on a trigger record field in the Salesforce user interface and prevents any DML operation from occurring.
- **addError(errorMsg, escape)**
  Places the specified error message, which can be escaped or unescaped, on a trigger record field in the Salesforce user interface, and prevents any DML operation from occurring.
- **addError(fieldName, errorMsg)**
  Dynamically add errors to fields of an SObject associated with the specified field name.
- **addError(fieldToken, errorMsg)**
  Dynamically add errors to an SObject instance associated with the specified field.
- **addError(fieldName, errorMsg, escape)**
  Dynamically add errors to fields of an SObject associated with the specified field name.
- **addError(fieldToken, errorMsg, escape)**
  Dynamically add errors to an SObject instance associated with the specified field.
- **clear()**
  Clears all field values

- **get(field)**
  Returns the value for the field specified by the field token `Schema.sObjectField`, such as, `Schema.Account.AccountNumber`.
- **getCloneSourceId()**
  Returns the ID of the entity from which an object was cloned. You can use it for objects cloned through the Salesforce user interface. You can also use it for objects created using the `System.SObject.clone(preserveId, isDeepClone, preserveReadonlyTimestamps, preserveAutonumber)` method, provided that the *preserveId* parameter wasn't used or was set to `false`. The `getCloneSourceId()` method can only be used within the transaction where the entity is cloned, as clone information doesn't persist in subsequent transactions.
- **getErrors()**
  Returns a list of `Database.Error` objects for an SObject instance. If the SObject has no errors, an empty list is returned.
- **getOptions()**
  Returns the database.DMLOptions object for the SObject.
- **getPopulatedFieldsAsMap()**
  Returns a map of populated field names and their corresponding values. The map contains only the fields that have been populated in memory for the SObject instance.
- **getSObject(fieldName)**
  Returns the value for the specified field. This method is primarily used with dynamic DML to access values for external IDs.
- **getSObject(field)**
  Returns the value for the field specified by the field token `Schema.sObjectField`, such as, `Schema.MyObj.MyExternalId`. This method is primarily used with dynamic DML to access values for external IDs.
- **getSObjects(fieldName)**
  Returns the values for the specified field. This method is primarily used with dynamic DML to access values for associated objects, such as child relationships.
- **getSObjects(fieldName)**
  Returns the value for the field specified by the field token `Schema.fieldName`, such as, `Schema.Account.Contact`. This method is primarily used with dynamic DML to access values for associated objects, such as child relationships.
- **getSObjectType()**
  Returns the token for this SObject. This method is primarily used with describe information.
- **getQuickActionName()**
  Retrieves the name of a quick action associated with this SObject. Typically used in triggers.
- **hasErrors()**
  Returns true if an SObject instance has associated errors. The error message can be associated to the SObject instance by using `SObject.addError()`, validation rules, or by other means.
- **isClone()**
  Returns `true` if an entity is cloned from something, even if the entity hasn't been saved. The method can only be used within the transaction where the entity is cloned, as clone information doesn't persist in subsequent transactions.
- **isSet(fieldName)**
  Returns information about the queried sObject field. Returns `true` if the sObject field is populated, either by direct assignment or by inclusion in a SOQL query. Returns `false` if the sObject field isn't set. If an invalid field is specified, an SObjectException is thrown.
- **isSet(field)**
  Returns information about the queried sObject field. Returns `true` if the sObject field is populated, either by direct assignment or by inclusion in a SOQL query. Returns `false` if the sObject field isn't set. If an invalid field is specified, an SObjectException is thrown.

`Schema.Account.AccountNumber` and returns the previous value for the field.

- **putSObject(fieldName, value)**
  Sets the value for the specified field. This method is primarily used with dynamic DML for setting external IDs. The method returns the previous value of the field.
- **putSObject(fieldName, value)**
  Sets the value for the field specified by the token `Schema.SObjectType`. This method is primarily used with dynamic DML for setting external IDs. The method returns the previous value of the field.
- **recalculateFormulas()**
  Deprecated as of API version 57.0. Use the `recalculateFormulas()` method in the `System.Formula` class instead.
- **setOptions(DMLOptions)**
  Sets the DMLOptions object for the SObject.

addError(errorMsg)

Marks a trigger record with a custom error message and prevents any DML operation from occurring.

Signature

```
public Void addError(String errorMsg)
```

Parameters

errorMsg

Type: String

The error message to mark the record with.

Return Value

Type: Void

Usage

When used on `Trigger.new` in `insert` and `update` triggers, and on `Trigger.old` in `delete` triggers, the error message is displayed in the application interface.

See Triggers and Trigger Exceptions.

> **Note**
>
> This method escapes any HTML markup in the specified error message. The escaped characters are: `\n`, `<`, `>`, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. As a result, HTML markup is not rendered; instead, it is displayed as text in the Salesforce user interface.

When used in Visualforce controllers, the generated message is added to the collection of errors for the page. For more information, see Validation Rules and Standard Controllers in the *Visualforce Developer's Guide*.

Example

```
Trigger.new[0].addError('bad');
```

addError(errorMsg, escape)

```
public Void addError(String errorMsg, Boolean escape)
```

Parameters

errorMsg

Type: String

The error message to mark the record with.

escape

Type: Boolean

Indicates whether any HTML markup in the custom error message should be escaped (`true`) or not (`false`). This parameter is ignored in both Lightning Experience and the Salesforce mobile app, and the HTML is always escaped. The escape parameter only applies in Salesforce Classic.

Return Value

Type: Void

Usage

The escaped characters are: `\n`, `<`, `>`, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. As a result, HTML markup is not rendered; instead, it is displayed as text in the Salesforce user interface.

Warning

Be cautious if you specify `false` for the *escape* argument. Unescaped strings displayed in the Salesforce user interface can represent a vulnerability in the system because these strings might contain harmful code. If you want to include HTML markup in the error message, call this method with a `false` *escape* argument. Make sure that you escape any dynamic content, such as input field values. Otherwise, specify `true` for the *escape* argument or call `addError(String errorMsg)` instead.

Example

```
Trigger.new[0].addError('Fix & resubmit', false);
```

addError(exceptionError)

Marks a trigger record with a custom error message and prevents any DML operation from occurring.

Signature

```
public Void addError(Exception exceptionError)
```

Parameters

exceptionError

Type: System.Exception

An Exception object or a custom exception object that contains the error message to mark the record with.

Return Value

Type: Void

See Triggers and Trigger Exceptions.

Note

This method escapes any HTML markup in the specified error message. The escaped characters are: `\n`, `<`, `>`, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. As a result, HTML markup is not rendered; instead, it is displayed as text in the Salesforce user interface.

When used in Visualforce controllers, the generated message is added to the collection of errors for the page. For more information, see Validation Rules and Standard Controllers in the *Visualforce Developer's Guide*.

Example

```
public class MyException extends Exception {}
Trigger.new[0].addError(new myException('Invalid Id'));
```

addError(exceptionError, escape)

Marks a trigger record with a custom exception error message, specifies whether or not the exception error message should be escaped, and prevents any DML operation from occurring.

Signature

`public Void addError(Exception exceptionError, Boolean escape)`

Parameters

exceptionError

Type: System.Exception

An Exception object or a custom exception object that contains the error message to mark the record with.

escape

Type: Boolean

Indicates whether any HTML markup in the custom error message should be escaped (`true`) or not (`false`). This parameter is ignored in both Lightning Experience and the Salesforce mobile app, and the HTML is always escaped. The escape parameter only applies in Salesforce Classic.

Return Value

Type: Void

Usage

The escaped characters are: `\n`, `<`, `>`, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. As a result, HTML markup is not rendered; instead, it is displayed as text in the Salesforce user interface.

strings might contain harmful code. If you want to include HTML markup in the error message, call this method with a `false` *escape* argument. Make sure that you escape any dynamic content, such as input field values. Otherwise, specify `true` for the *escape* argument or call `addError(Exception e)` instead.

Example

```
public class MyException extends Exception {}
Trigger.new[0].addError(new myException('Invalid Id & other issues', false));
```

addError(errorMsg)

Places the specified error message on a trigger record field in the Salesforce user interface and prevents any DML operation from occurring.

Signature

`public Void addError(String errorMsg)`

Parameters

errorMsg

Type: String

Return Value

Type: Void

Usage

Note:

- When used on `Trigger.new` in `before insert` and `before update` triggers, and on `Trigger.old` in `before delete` triggers, the error appears in the application interface.
- When used in Visualforce controllers, if there is an `inputField` component bound to field, the message is attached to the component. For more information, see Validation Rules and Standard Controllers in the *Visualforce Developer's Guide*.
- This method is highly specialized because the field identifier is not actually the invoking object–the sObject record is the invoker. The field is simply used to identify the field that should be used to display the error.

See Triggers and Trigger Exceptions.

Note

This method escapes any HTML markup in the specified error message. The escaped characters are: `\n`, `<`, `>`, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. As a result, HTML markup is not rendered; instead, it is displayed as text in the Salesforce user interface.

Example

```
Trigger.new[0].myField__c.addError('bad');
```

addError(errorMsg, escape)

```
public Void addError(String errorMsg, Boolean escape)
```

Parameters

errorMsg

Type: String

The error message to mark the record with.

escape

Type: Boolean

Indicates whether any HTML markup in the custom error message should be escaped (`true`) or not (`false`). This parameter is ignored in both Lightning Experience and the Salesforce mobile app, and the HTML is always escaped. The escape parameter only applies in Salesforce Classic.

Return Value

Type:

Usage

The escaped characters are: `\n`, `<`, `>`, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. As a result, HTML markup is not rendered; instead, it is displayed as text in the Salesforce user interface.

Warning

Be cautious if you specify `false` for the *escape* argument. Unescaped strings displayed in the Salesforce user interface can represent a vulnerability in the system because these strings might contain harmful code. If you want to include HTML markup in the error message, call this method with a `false` *escape* argument. Make sure that you escape any dynamic content, such as input field values. Otherwise, specify `true` for the *escape* argument or call `field.addError(String errorMsg)` instead.

Example

```
Trigger.new[0].myField__c.addError('Fix & resubmit', false);
```

addError(fieldName, errorMsg)

Dynamically add errors to fields of an SObject associated with the specified field name.

Signature

```
public void addError(String fieldName, String errorMsg)
```

Parameters

fieldName

Type: String

The field name of the SObject .

errorMsg

Type: String

The error message to be added. HTML special characters in the error message string are always escaped.

If the field name is an empty string or null, the error is associated with the SObject and not with a specific field.

Example

```
// Add an error to an SObject field using the addError() method.
Account acct = new Account(name = 'TestAccount');
acct.addError('name', 'error in name field');
// Use the hasErrors() method to verify that the error is added, and then the getErrors()
System.Assert(acct.hasErrors());
List<Database.Error> errors = acct.getErrors();
System.AssertEquals(1, errors.size());
```

addError(fieldToken, errorMsg)

Dynamically add errors to an SObject instance associated with the specified field.

Signature

```
public void addError(Schema.SObjectField fieldToken, String errorMsg
```

Parameters

fieldToken

Type: Schema.SObjectField

The field of the SObject instance.

errorMsg

Type: String

The error message to be added. HTML special characters in the error message string are always escaped.

Return Value

Type: void

Usage

Use this method to add errors to the specified field token of a standard or custom object. If `fieldToken` is null, the error is associated with the SObject and not with a specific field.

Example

```
// Add an error to a field of an SObject instance using the addError() method.
Account acct = new Account(name = 'TestAccount');
Schema.DescribeFieldResult nameDesc = Account.name.getDescribe();
Schema.sObjectField nameField = nameDesc.getSObjectField();
acct.addError(nameField, 'error is name field');
// Use the hasErrors() method to verify that the error is added, and then the getErrors()
System.Assert(acct.hasErrors());
List<Database.Error> errors = acct.getErrors();
System.AssertEquals(1, errors.size());
```

addError(fieldName, errorMsg, escape)

Dynamically add errors to fields of an SObject associated with the specified field name.

fieldName

Type: String

The field name of the SObject .

errorMsg

Type: String

The error message to be added.

escape

Type: Boolean

Indicates whether any HTML markup in the custom error message should be escaped (`true`) or not (`false`). This parameter is ignored in both Lightning Experience and the Salesforce mobile app, and the HTML is always escaped. The escape parameter only applies in Salesforce Classic.

Return Value

Type: void

Usage

If the field name is an empty string or null, the error is associated with the SObject and not with a specific field.

The escaped characters are: `\n`, `<`, `>`, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. As a result, HTML markup is not rendered; instead, it is displayed as text in the Salesforce user interface.

Warning

- The *escape* parameter cannot be disabled in Lightning Experience and in the Salesforce mobile app, and will be ignored.
- Be cautious if you specify `false` for the *escape* argument. Unescaped strings displayed in the Salesforce user interface can represent a vulnerability in the system because these strings might contain harmful code. If you want to include HTML markup in the error message, call this method with a `false` *escape* argument. Make sure that you escape any dynamic content, such as input field values. Otherwise, specify `true` for the *escape* argument or call `addError(String fieldName, String errorMsg)` instead.

Example

```
// Add an error to an SObject field using the addError() method.
Account acct = new Account(name = 'TestAccount');
acct.addError('name', 'error in name field', false);
// Use the hasErrors() method to verify that the error is added, and then the getErrors()
System.Assert(acct.hasErrors());
List<Database.Error> errors = acct.getErrors();
System.AssertEquals(1, errors.size());
```

addError(fieldToken, errorMsg, escape)

Dynamically add errors to an SObject instance associated with the specified field.

Signature

```
public void addError(Schema.SObjectField fieldToken, String errorMsg, Boolean escape)
```

The field of the SObject instance.

errorMsg

Type: String

The error message to be added.

escape

Type: Boolean

Indicates whether any HTML markup in the custom error message should be escaped (`true`) or not (`false`). This parameter is ignored in both Lightning Experience and the Salesforce mobile app, and the HTML is always escaped. The escape parameter only applies in Salesforce Classic.

Return Value

Type: void

Usage

Use this method to add errors to the specified field token of a standard or custom object. If `fieldToken` is null, the error is associated with the SObject and not with a specific field.

The escaped characters are: `\n`, `<`, `>`, `&`, `"`, `\`, `\u2028`, `\u2029`, and `\u00a9`. As a result, HTML markup is not rendered; instead, it is displayed as text in the Salesforce user interface.

Warning

- The *escape* parameter cannot be disabled in Lightning Experience and in the Salesforce mobile app, and will be ignored.
- Be cautious if you specify `false` for the *escape* argument. Unescaped strings displayed in the Salesforce user interface can represent a vulnerability in the system because these strings might contain harmful code. If you want to include HTML markup in the error message, call this method with a `false` *escape* argument. Make sure that you escape any dynamic content, such as input field values. Otherwise, specify `true` for the *escape* argument or call `addError(Schema.SObjectField fieldToken, String errorMsg)` instead.

Example

```
// Add an error to a field of an SObject instance using the addError() method.
Account acct = new Account(name = 'TestAccount');
Schema.DescribeFieldResult nameDesc = Account.name.getDescribe();
Schema.sObjectField nameField = nameDesc.getSObjectField();
acct.addError(nameField, 'error is name field', false);
// Use the hasErrors() method to verify that the error is added, and then the getErrors()
System.Assert(acct.hasErrors());
List<Database.Error> errors = acct.getErrors();
System.AssertEquals(1, errors.size());
```

clear()

Clears all field values

Signature

```
public Void clear()
```

Return Value

```
Account acc = new account(Name = 'Acme');
acc.clear();
Account expected = new Account();
system.assertEquals(expected, acc);
```

clone(preserveId, isDeepClone, preserveReadonlyTimestamps, preserveAutonumber)

Creates a copy of the SObject record.

Signature

`public SObject clone(Boolean preserveId, Boolean isDeepClone, Boolean preserveReadonlyTimestamps, Boolean preserveAutonumber)`

Parameters

preserveId

Type: Boolean

(Optional) Determines whether the ID of the original object is preserved or cleared in the duplicate. If set to `true`, the ID is copied to the duplicate. The default is `false`, that is, the ID is cleared.

isDeepClone

Type: Boolean

(Optional) Determines whether the method creates a full copy of the SObject field or just a reference:

- If set to `true`, the method creates a full copy of the SObject. All fields on the SObject are duplicated in memory, including relationship fields. Consequently, if you change a field on the cloned SObject, the original SObject isn't affected.
- If set to `false`, the method performs a shallow copy of the SObject fields. All copied relationship fields reference the original SObjects. Consequently, if you change a relationship field on the cloned SObject, the corresponding field on the original SObject is also affected, and vice versa. The default is `false`.

preserveReadonlyTimestamps

Type: Boolean

(Optional) Determines whether the read-only timestamp fields are preserved or cleared in the duplicate. If set to `true`, the read-only fields `CreatedById`, `CreatedDate`, `LastModifiedById`, and `LastModifiedDate` are copied to the duplicate. The default is `false`, that is, the values are cleared.

Note

Audit field values won't be persisted to the database via DML on the cloned SObject instance.

preserveAutonumber

Type: Boolean

(Optional) Determines whether auto number fields of the original object are preserved or cleared in the duplicate. If set to `true`, auto number fields are copied to the cloned object. The default is `false`, that is, auto number fields are cleared.

Return Value

Note

For Apex saved using Salesforce API version 22.0 or earlier, the default value for the *preserveId* argument is `true`, that is, the ID is preserved.

Example

```
Account acc = new account(Name = 'Acme', Description = 'Acme Account');
Account clonedAcc = acc.clone(false, false, false, false);
System.assertEquals(acc, clonedAcc);
```

get(fieldName)

Returns the value for the field specified by *fieldName*, such as `AccountNumber`.

Signature

`public Object get(String fieldName)`

Parameters

fieldName

Type: String

Return Value

Type: Object

Usage

For more information, see Dynamic SOQL.

Example

```
Account acc = new account(Name = 'Acme', Description = 'Acme Account');
String description = (String)acc.get('Description');
System.assertEquals('Acme Account', description);
```

Versioned Behavior Changes

In API version 34.0 and later, you must include the namespace name to retrieve a field from a field Map using this method. For example, to get the *account__c* field in the *MyNamespace* namespace from a *fields* field Map, use: `fields.get('MyNamespace__account__c')`.

get(field)

Returns the value for the field specified by the field token `Schema.sObjectField`, such as, `Schema.Account.AccountNumber`.

Signature

`public Object get(Schema.sObjectField field)`

Parameters

field

Type: Schema.SObjectField

For more information, see Dynamic SOQL.

Note

Field tokens aren't available for person accounts. If you access `Schema.Account.fieldname`, you get an exception error. Instead, specify the field name as a string.

Example

```
Account acc = new account(Name = 'Acme', Description = 'Acme Account');
String description = (String)acc.get(Schema.Account.Description);
System.assertEquals('Acme Account', description);
```

getCloneSourceId()

Returns the ID of the entity from which an object was cloned. You can use it for objects cloned through the Salesforce user interface. You can also use it for objects created using the `System.SObject.clone(preserveId, isDeepClone, preserveReadonlyTimestamps, preserveAutonumber)` method, provided that the *preserveId* parameter wasn't used or was set to `false`. The `getCloneSourceId()` method can only be used within the transaction where the entity is cloned, as clone information doesn't persist in subsequent transactions.

Signature

`public Id getCloneSourceId()`

Return Value

Type: Id

Usage

If A is cloned to B, B is cloned to C, and C is cloned to D, then B, C, and D all point back to A as their clone source.

Example

```
Account acc0 = new Account(Name = 'Acme');
insert acc0;
Account acc1 = acc0.clone();
Account acc2 = acc1.clone();
Account acc3 = acc2.clone();
Account acc4 = acc3.clone();
System.assert(acc0.Id != null);
System.assertEquals(acc0.Id, acc1.getCloneSourceId());
System.assertEquals(acc0.Id, acc2.getCloneSourceId());
System.assertEquals(acc0.Id, acc3.getCloneSourceId());
System.assertEquals(acc0.Id, acc4.getCloneSourceId());
System.assertEquals(null, acc0.getCloneSourceId());
```

getErrors()

Returns a list of `Database.Error` objects for an SObject instance. If the SObject has no errors, an empty list is returned.

Signature

getOptions()

Returns the database.DMLOptions object for the SObject.

Signature

```
public Database.DMLOptions getOptions()
```

Return Value

Type: Database.DMLOptions

Example

```
Database.DMLOptions dmo = new Database.dmlOptions();
dmo.assignmentRuleHeader.useDefaultRule = true;

Account acc = new Account(Name = 'Acme');
acc.setOptions(dmo);
Database.DMLOptions accDmo = acc.getOptions();
```

getPopulatedFieldsAsMap()

Returns a map of populated field names and their corresponding values. The map contains only the fields that have been populated in memory for the SObject instance.

Signature

```
public Map<String,Object> getPopulatedFieldsAsMap()
```

Return Value

Type: Map<String,Object>

A map of field names and their corresponding values.

Usage

The returned map contains only the fields that have been populated in memory for the SObject instance, which makes it easy to iterate over those fields. A field is populated in memory in the following cases.

- The field has been queried by a SOQL statement.
- The field has been explicitly set before the call to the `getPopulatedFieldsAsMap()` method.

Fields on related objects that are queried or set are also returned in the map.

The following example iterates over the map returned by the `getPopulatedFieldsAsMap()` method after a SOQL query.

```
Account a = new Account();
a.name = 'TestMapAccount1';
insert a;
a = [select Id,Name from Account where id=:a.Id];
Map<String, Object> fieldsToValue = a.getPopulatedFieldsAsMap();

for (String fieldName : fieldsToValue.keySet()){
    System.debug('field name is ' + fieldName + ', value is ' +
        fieldsToValue.get(fieldName));
}
```

on the SObject are explicitly set.

```
Account a = new Account();
a.name = 'TestMapAccount2';
a.phone = '123-4567';
insert a;
Map<String, Object> fieldsToValue = a.getPopulatedFieldsAsMap();

for (String fieldName : fieldsToValue.keySet()) {
    System.debug('field name is ' + fieldName + ', value is ' +
        fieldsToValue.get(fieldName));
}

// Example debug statement output:
// DEBUG|field name is Name, value is TestMapAccount2
// DEBUG|field name is Phone, value is 123-4567
// DEBUG|field name is Id, value is 001R0000003EPPpIAO
```

The following example shows how to use the `getPopulatedFieldsAsMap()` method with related objects.

```
Account a = new Account();
a.name='TestMapAccount3';
insert a;
Contact c = new Contact();
c.firstname='TestContactFirstName';
c.lastName ='TestContactLastName';
c.accountid = a.id;
insert c;

c = [SELECT id, Contact.Firstname, Contact.Account.Name FROM Contact
        where id=:c.id limit 1];
Map<String, Object> fieldsToValue = c.getPopulatedFieldsAsMap();

// To get the fields on Account, get the Account object
// and call getMapPopulatedFieldsAsMap() on that object.

a = (Account)fieldsToValue.get('Account');
fieldsToValue = a.getPopulatedFieldsAsMap();

for (String fieldName : fieldsToValue.keySet()) {
    System.debug('field name is ' + fieldName + ', value is ' +
        fieldsToValue.get(fieldName));
}

// Example debug statement output:
// DEBUG|field name is Id, value is 001R0000003EPPuIAO
// DEBUG|field name is Name, value is TestMapAccount3
```

Versioned Behavior Changes

In API version 39.0 and later, getPopulatedFieldsAsMap returns all values set on the SObject, even if values were set after the record was queried. This behavior is dependent on the version of the apex class calling this method and not on the version of the class that generated the SObject. If you query an SObject at API version 20.0, and then call this method in a class with API version 40.0, you will get the full set of fields.

getSObject(fieldName)

Returns the value for the specified field. This method is primarily used with dynamic DML to access values for external IDs.

Signature

Type: String

Return Value

Type: SObject

Example

```
Account acc = new account(Name = 'Acme', Description = 'Acme Account');
insert acc;
Contact con = new Contact(Lastname = 'AcmeCon', AccountId = acc.id);
insert con;

SObject contactDB =
    [SELECT Id, AccountId, Account.Name FROM Contact WHERE id = :con.id LIMIT 1];
Account a = (Account)contactDB.getSObject('Account');
System.assertEquals('Acme', a.name);
```

getSObject(field)

Returns the value for the field specified by the field token `Schema.sObjectField`, such as, `Schema.MyObj.MyExternalId`. This method is primarily used with dynamic DML to access values for external IDs.

Signature

```
public SObject getSObject(Schema.SObjectField field)
```

Parameters

field

Type: Schema.SObjectField

Return Value

Type: SObject

Usage

If the method references polymorphic fields, a Name object is returned. Use the `TYPEOF` clause in the SOQL SELECT statement to directly get results that depend on the runtime object type referenced by the polymorphic field. See Working with Polymorphic Relationships in SOQL Queries.

Example

```
Account acc = new account(name = 'Acme', description = 'Acme Account');
insert acc;
Contact con = new contact(lastname = 'AcmeCon', accountid = acc.id);
insert con;

Schema.DescribeFieldResult fieldResult = Contact.AccountId.getDescribe();
Schema.SObjectField field = fieldResult.getSObjectField();

SObject contactDB =
    [SELECT Id, AccountId, Account.Name FROM Contact WHERE id = :con.id LIMIT 1];
Account a = (Account)contactDB.getSObject(field);
System.assertEquals('Acme', a.name);
```

getSObjects(fieldName)

```
public SObject[] getSObjects(String fieldName)
```

Parameters

fieldName

Type: String

Return Value

Type: SObject[]

Usage

For more information, see Dynamic DML.

Example

```
Account acc = new account(name = 'Acme', description = 'Acme Account');
insert acc;
Contact con = new contact(lastname = 'AcmeCon', accountid = acc.id);
insert con;

SObject[] a = [SELECT id, (SELECT Name FROM Contacts LIMIT 1) FROM Account WHERE id = :acc
SObject[] contactsDB = a.get(0).getSObjects('Contacts');
String fieldValue = (String)contactsDB.get(0).get('Name');
System.assertEquals('AcmeCon', fieldValue);
```

getSObjects(fieldName)

Returns the value for the field specified by the field token `Schema.`*`fieldName`*, such as, `Schema.Account.Contact`. This method is primarily used with dynamic DML to access values for associated objects, such as child relationships.

Signature

```
public SObject[] getSObjects(Schema.SObjectType fieldName)
```

Parameters

fieldName

Type: Schema.SObjectType

Return Value

Type: SObject[]

getSObjectType()

Returns the token for this SObject. This method is primarily used with describe information.

Signature

```
public Schema.SObjectType getSObjectType()
```

Return Value

Type: Schema.SObjectType

Usage

For more information, see apex_dynamic_describe_objects_understanding.

```
    Schema.SObjectType expected = Schema.Account.getSObjectType();
    System.assertEquals(expected, acc.getSObjectType());
```

getQuickActionName()

Retrieves the name of a quick action associated with this SObject. Typically used in triggers.

Signature

`public String getQuickActionName()`

Return Value

Type: String

Example

```
    trigger accTrig2 on Contact (before insert) {
        for (Contact c : Trigger.new) {
            if (c.getQuickActionName() == QuickAction.CreateContact) {
                c.WhereFrom__c = 'GlobaActionl';
            } else if (c.getQuickActionName() == Schema.Account.QuickAction.CreateContact) {
                c.WhereFrom__c = 'AccountAction';
            } else if (c.getQuickActionName() == null) {
                c.WhereFrom__c = 'NoAction';
            } else {
                System.assert(false);
            }
        }
    }
```

hasErrors()

Returns true if an SObject instance has associated errors. The error message can be associated to the SObject instance by using `SObject.addError()`, validation rules, or by other means.

Signature

`public Boolean hasErrors()`

Return Value

Type: Boolean

isClone()

Returns `true` if an entity is cloned from something, even if the entity hasn't been saved. The method can only be used within the transaction where the entity is cloned, as clone information doesn't persist in subsequent transactions.

Signature

`public Boolean isClone()`

Return Value

Type: Boolean

Example

```
insert acc2;
// Test after saving
System.assertEquals(true, acc2.isClone());
```

isSet(fieldName)

Returns information about the queried sObject field. Returns `true` if the sObject field is populated, either by direct assignment or by inclusion in a SOQL query. Returns `false` if the sObject field isn't set. If an invalid field is specified, an SObjectException is thrown.

Signature

`public Boolean isSet(String fieldName)`

Parameters

fieldName

Type: String

Return Value

Type: Boolean

Usage

The `isSet` method doesn't check if a field is accessible to a specific user via org permissions or other specialized access permissions.

Example

```
Contact  c = new Contact(LastName = 'Joyce');
System.assertEquals(true, c.isSet('LastName'));
System.assertEquals(false, c.isSet('FirstName')); // FirstName field is not written to
c.firstName = null;
System.assertEquals(true, c.isSet('FirstName')); //FirstName field is written to
```

isSet(field)

Returns information about the queried sObject field. Returns `true` if the sObject field is populated, either by direct assignment or by inclusion in a SOQL query. Returns `false` if the sObject field isn't set. If an invalid field is specified, an SObjectException is thrown.

Signature

`public Boolean isSet(Schema.SObjectField field)`

Parameters

field

Type:SObjectField Class

Return Value

Type: Boolean

Usage

The `isSet` method doesn't check if a field is accessible to a specific user via org permissions or other specialized access permissions.

Example

```
    System.assertEquals(true, c.isSet(Contact.FirstName)); //FirstName field in query
    System.assertEquals(false, c.isSet(Contact.LastName)); //LastName field not in query
```

put(fieldName, value)

Sets the value for the specified field and returns the previous value for the field.

Signature

`public Object put(String fieldName, Object value)`

Parameters

fieldName

Type: String

value

Type: Object

Return Value

Type: Object

Example

```
Account acc = new Account(name = 'test', description = 'old desc');
String oldDesc = (String)acc.put('description', 'new desc');
System.assertEquals('old desc', oldDesc);
System.assertEquals('new desc', acc.description);
```

put(field, value)

Sets the value for the field specified by the field token `Schema.sObjectField`, such as, `Schema.Account.AccountNumber` and returns the previous value for the field.

Signature

`public Object put(Schema.SObjectField field, Object value)`

Parameters

field

Type: Schema.SObjectField

value

Type: Object

Return Value

Type: Object

Example

```
Account acc = new Account(name = 'test', description = 'old desc');
String oldDesc = (String)acc.put(Schema.Account.Description, 'new desc');
System.assertEquals('old desc', oldDesc);
System.assertEquals('new desc', acc.description);
```

putSObject(fieldName, value)

Sets the value for the specified field. This method is primarily used with dynamic DML for setting external IDs. The method returns the previous value of the field.

Signature

```
public SObject putSObject(String fieldName, SObject value)
```

Parameters

fieldName

Type: String

value

Type: SObject

Return Value

Type: SObject

Example

```
Account acc = new Account(name = 'Acme', description = 'Acme Account');
insert acc;
Contact con = new contact(lastname = 'AcmeCon', accountid = acc.id);
insert con;
Account acc2 = new account(name = 'Not Acme');

Contact contactDB =
    (Contact)[SELECT Id, AccountId, Account.Name FROM Contact WHERE id = :con.id LIMIT 1]
Account a = (Account)contactDB.putSObject('Account', acc2);
System.assertEquals('Acme', a.name);
System.assertEquals('Not Acme', contactDB.Account.name);
```

putSObject(fieldName, value)

Sets the value for the field specified by the token `Schema.SObjectType`. This method is primarily used with dynamic DML for setting external IDs. The method returns the previous value of the field.

Signature

```
public SObject putSObject(Schema.SObjectType fieldName, SObject value)
```

Parameters

fieldName

Type: Schema.SObjectType

value

Type: SObject

Return Value

Type: SObject

recalculateFormulas()

```
public Void recalculateFormulas()
```

Return Value

Type: Void

Usage

This method doesn't recalculate cross-object formulas. If you call this method on objects that have both cross-object and non-cross-object formula fields, only the non-cross-object formula fields are recalculated.

Each `recalculateFormulas` call counts against the SOQL query limits. See Execution Governors and Limits.

See Also

- recalculateFormulas(sobjects)
- What Is a Cross-Object Formula?

setOptions(DMLOptions)

Sets the DMLOptions object for the SObject.

Signature

```
public Void setOptions(database.DMLOptions DMLOptions)
```

Parameters

DMLOptions

Type: Database.DMLOptions

Return Value

Type: Void

Example

```
Database.DMLOptions dmo = new Database.dmlOptions();
dmo.assignmentRuleHeader.useDefaultRule = true;

Account acc = new Account(Name = 'Acme');
acc.setOptions(dmo);
```

DID THIS ARTICLE SOLVE YOUR ISSUE?
Let us know so we can improve!

Share your feedback