Id Class

Contains methods for the ID primitive data type.

Namespace

System

Example: Getting an sObject Token From an ID

This sample shows how to use the `getSObjectType` method to obtain an sObject token from an ID. The `updateOwner` method in this sample accepts a list of IDs of the sObjects to update the ownerId field of. This list contains IDs of sObjects of the same type. The second parameter is the new owner ID. Note that since it is a future method, it doesn't accept sObject types as parameters; this is why it accepts IDs of sObjects. This method gets the sObject token from the first ID in the list, then does a describe to obtain the object name and constructs a query dynamicallly. It then queries for all sObjects and updates their owner ID fields to the new owner ID.

```
public class MyDynamicSolution {
    @future
    public static void updateOwner(List<ID> objIds, ID newOwnerId) {
        // Validate input
        System.assert(objIds != null);
        System.assert(objIds.size() > 0);
        System.assert(newOwnerId != null);

        // Get the sObject token from the first ID
        // (the List contains IDs of sObjects of the same type).
        Schema.SObjectType token = objIds[0].getSObjectType();

        // Using the token, do a describe
        // and construct a query dynamically.
        Schema.DescribeSObjectResult dr = token.getDescribe();
        String queryString = 'SELECT ownerId FROM ' + dr.getName() +
            ' WHERE ';
        for(ID objId : objIds) {
            queryString += 'Id=\'' + objId + '\' OR ';
        }
        // Remove the last ' OR'
        queryString = queryString.subString(0, queryString.length() - 4);

        sObject[] objDBList = Database.query(queryString);
        System.assert(objDBList.size() > 0);

        // Update the owner ID on the sObjects
        for(Integer i=0;i<objDBList.size();i++) {
            objDBList[i].put('ownerId', newOwnerId);
        }
        Database.SaveResult[] srList = Database.update(objDBList, false);
        for(Database.SaveResult sr : srList) {
            if (sr.isSuccess()) {
                System.debug('Updated owner ID successfully for ' +
                    dr.getName() + ' ID ' + sr.getId());
            }
            else {
                System.debug('Updating ' + dr.getName() + ' returned the following errors.
                for(Database.Error e : sr.getErrors()) {
                    System.debug(e.getMessage());
```

Id Methods

The following are methods for `Id`.

- 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 error message and prevents any DML operation from occurring.
- getSObjectType()
  Returns the token for the sObject corresponding to this ID. This method is primarily used with describe information.
- to15()
  Converts an 18-character Id value to a 15-character case-sensitive string.
- valueOf(toID)
  Converts the specified String into an ID and returns the ID.
- valueOf(str, restoreCasing)
  Converts the specified string into an ID and returns the ID. If `restoreCasing` is `true`, and the string represents an 18-character ID that has incorrect casing, the method returns an 18-character ID that is correctly aligned with its encoded casing.

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

This method is similar to the `addError(errorMsg)` sObject method.

> **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.

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

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.

Signature

```
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].Id.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)
```

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

Return Value

Type: Void

Usage

This method is similar to the `addError(exceptionError)` sObject method.

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

```
public class MyException extends Exception{}

Trigger.new[0].Id.addError(new myException('Invalid Id'));
```

addError(exceptionError, escape)

Marks a trigger record with a custom error message 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.

> **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

```
public class MyException extends Exception{}

account a = new account();
a.addError(new MyException('Invalid Id & other issues'), false);
```

getSObjectType()

Returns the token for the sObject corresponding to this ID. This method is primarily used with describe information.

Signature

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

Return Value

Type: Schema.SObjectType

Usage

For more information about describes, see Understanding Apex Describe Information.

Example

```
account a = new account(name = 'account');
insert a;
Id myId = a.id;
system.assertEquals(Schema.Account.SObjectType, myId.getSobjectType());
```

to15()

Converts an 18-character Id value to a 15-character case-sensitive string.

Signature

```
public static string to15()
```

Return Value

Type: String

Example

```
String Id_15_char = '0D5B000001DVM9t';
String Id_18_char = '0D5B000001DVM9tkAh';
ID testId = Id_18_char;
System.assertEquals(testId.to15(),Id_15_char);
```

valueOf(toID)

Converts the specified String into an ID and returns the ID.

Signature

```
public static ID valueOf(String toID)
```

Return Value

Type: ID

Example

```
Id myId = Id.valueOf('001xa000003DIlo');
```

Versioned Behavior Changes

In API version 54.0 and later, assignment of an invalid 15 or 18 character ID to a variable results in a `System.StringException` exception.

valueOf(str, restoreCasing)

Converts the specified string into an ID and returns the ID. If `restoreCasing` is `true`, and the string represents an 18-character ID that has incorrect casing, the method returns an 18-character ID that is correctly aligned with its encoded casing.

Signature

```
public static Id valueOf(String str, Boolean restoreCasing)
```

Parameters

str

Type: String

String to be converted to an ID

restoreCasing

Type: Boolean

If set to `true`, and *str* represents an 18-character ID, the method returns an 18-character ID that is correctly aligned with its casing.

Return Value

Type: Id

The return value depends on both the *str* and the *restoreCasing* parameter values.

> **Note**
>
> If the *str* is invalid, the method throws a `System.StringException` exception.

Parameters	restoreCasing=true	restoreCasing=false
Valid 15-character *str* value	Because the 15-character input value can't be encoded for casing, the method throws a `System.StringException`.	The method returns a 15-character ID.
Valid 18-character *str* value	The method returns an 18-character ID that is correctly aligned with its encoded casing.	The method doesn't attempt to correctly align the casing of the 18-character ID and returns an 18-character ID.

DEVELOPER CENTERS

- Heroku
- MuleSoft
- Tableau
- Commerce Cloud
- Lightning Design System
- Einstein
- Quip

POPULAR RESOURCES

- Documentation
- Component Library
- APIs
- Trailhead
- Sample Apps
- Podcasts
- AppExchange

COMMUNITY

- Trailblazer Community
- Events and Calendar
- Partner Community
- Blog
- Salesforce Admins
- Salesforce Architects