Test Class

Contains methods related to Apex tests.

Namespace

System

Test Methods

The following are methods for `Test`. All methods are static.

- **calculatePermissionSetGroup(psgIds)**
  Calculates aggregate permissions in specified permission set groups for testing.
- **calculatePermissionSetGroup(psgId)**
  Calculates aggregate permissions in a specified permission set group for testing.
- **clearApexPageMessages()**
  Clear the messages on a Visualforce page while executing Apex test methods.
- **createSoqlStub(targetType, soqlStub)**
  Creates a stub that will respond to SOQL queries against the specified SObject type you can use during testing.
- **createStub(parentType, stubProvider)**
  Creates a stubbed version of an Apex class that you can use for testing. This method is part of the Apex stub API. You can use it with the `System.StubProvider` interface to create a mocking framework.
- **createStubQueryRow(targetType, fieldMapWithRelationshipKeys)**
  Creates an instance of a stubbed SObject type that you can use to provide testing results in the extended `System.SoqlStubProvider` class.
- **createStubQueryRows(targetType, fieldMapWithRelationshipKeysForMultipleRows)**
  Creates instances of stubbed SObject types that you can use to provide testing results in the extended `System.SoqlStubProvider` class.
- **enableChangeDataCapture()**
  Use this method in an Apex test so that change event notifications are generated for all supported Change Data Capture entities. Call this method at the beginning of your test before performing DML operations and calling `Test.getEventBus().deliver();`.
- **enqueueBatchJobs(numberOfJobs)**
  Adds the specified number of jobs with no-operation contents to the test-context queue. It first fills the test batch queue, up to the maximum 5 jobs, and then places jobs in the test flex queue. It throws a limit exception when the number of jobs in the test flex queue exceeds the allowed limit of 100 jobs.
- **getEventBus()**
  Returns an instance of the test event bus broker, which lets you operate on platform event or change event messages in an Apex test. For example, you can call `Test.getEventBus().deliver()` to deliver event messages.
- **getFlexQueueOrder()**
  Returns an ordered list of job IDs for jobs in the test-context flex queue. The job at index `0` is the next job slated to run. This method returns only test-context results, even if it's annotated with `@IsTest(SeeAllData=true)`.
- **getStandardPricebookId()**
  Returns the ID of the standard price book in the organization.

Returns `true` if the currently executing code was called by code contained in a test method, `false` otherwise. Use this method if you need to run different code depending on whether it was being called from a test.

- **isSoqlStubDefined(targetType)**
  Returns `true` if a SOQL stub is defined for an SObject type; otherwise returns `false`.
- **loadData(sObjectToken, resourceName)**
  Inserts test records from the specified static resource .csv file and for the specified sObject type, and returns a list of the inserted sObjects.
- **newSendEmailQuickActionDefaults(contextId, replyToId)**
  Creates a new QuickAction.SendEmailQuickActionDefaults instance for testing a class implementing the QuickAction.QuickActionDefaultsHandler interface.
- **setContinuationResponse(requestLabel, mockResponse)**
  Sets a mock response for a continuation HTTP request in a test method.
- **setCreatedDate(recordId, createdDatetime)**
  Sets `CreatedDate` for a test-context sObject.
- **setCurrentPage(page)**
  A Visualforce test method that sets the current PageReference for the controller.
- **setCurrentPageReference(page)**
  A Visualforce test method that sets the current PageReference for the controller.
- **setFixedSearchResults(fixedSearchResults)**
  Defines a list of fixed search results to be returned by all subsequent SOSL statements in a test method.
- **setMock(interfaceType, instance)**
  Sets the response mock mode and instructs the Apex runtime to send a mock response whenever a callout is made through the HTTP classes or the auto-generated code from WSDLs.
- **setReadOnlyApplicationMode(applicationMode)**
  Sets the application mode for an organization to read-only in an Apex test to simulate read-only mode during Salesforce upgrades and downtimes. The application mode is reset to the default mode at the end of each Apex test run.
- **startTest()**
  Marks the point in your test code when your test actually begins. Use this method when you are testing governor limits.
- **stopTest()**
  Marks the point in your test code when your test ends. Use this method in conjunction with the `startTest` method.
- **testInstall(installImplementation, version, isPush)**
  Tests the implementation of the InstallHandler interface, which is used for specifying a post install script in packages. Tests run as the test initiator in the development environment.
- **testSandboxPostCopyScript(script, organizationId, sandboxId, sandboxName)**
  Tests the implementation of the SandboxPostCopy Interface, which is used for specifying a script to run at the completion of a Sandbox copy. Tests run as the test initiator in the development environment.
- **testSandboxPostCopyScript(script, organizationId, sandboxId, sandboxName, RunAsAutoProcUser)**
  Tests the implementation of the SandboxPostCopy Interface, which is used for specifying a script to run at the completion of a Sandbox copy. When `RunAsAutoProcUser` is `true`, tests run as Automated Process user in the development environment.
- **testUninstall(uninstallImplementation)**
  Tests the implementation of the UninstallHandler interface, which is used for specifying an uninstall script in packages. Tests run as the test initiator in the development environment.

calculatePermissionSetGroup(psgIds)

Parameters

psgIds

Type: List<String>

A list of IDs for permission set groups.

Return Value

Type: void

calculatePermissionSetGroup(psgId)

Calculates aggregate permissions in a specified permission set group for testing.

Signature

```
public static void calculatePermissionSetGroup(String psgId)
```

Parameters

psgId

Type: String

A single ID for a specified permission set group.

Return Value

Type: void

clearApexPageMessages()

Clear the messages on a Visualforce page while executing Apex test methods.

Signature

```
public static void clearApexPageMessages()
```

Return Value

Type: void

Usage

This method may only be used in tests.

Example

```
@isTest
    static void clearMessagesTest() {
        Test.setCurrentPage(new PageReference('/'));
        ApexPages.addMessage(
            new ApexPages.Message(ApexPages.Severity.WARNING, 'Sample Warning')
        );
        System.assertEquals(1, ApexPages.getMessages().size());
        Test.clearApexPageMessages();
        System.assertEquals(0, ApexPages.getMessages().size());
    }
```

createSoqlStub(targetType, soqlStub)

```
public static void createSoqlStub(Schema.SObjectType targetType, System.SoqlStubProvider soqlStub)
```

Parameters

targetType

Type: Schema.SObjectType

The SObject type to be stubbed. This parameter can't be null.

soqlStub

Type: System.SoqlStubProvider

An implementation of the `SoqlStubProvider` abstract class.

Return Value

Type: void

See Also

- *Apex Developer Guide*: Mock SOQL Tests for Data Cloud Data Model Objects

createStub(parentType, stubProvider)

Creates a stubbed version of an Apex class that you can use for testing. This method is part of the Apex stub API. You can use it with the `System.StubProvider` interface to create a mocking framework.

Signature

```
public static Object createStub(System.Type parentType, System.StubProvider stubProvider)
```

Parameters

parentType

Type: System.Type

The type of the Apex class to be stubbed.

stubProvider

System.StubProvider

An implementation of the `StubProvider` interface.

Return Value

Type: Object

Returns the stubbed object to use in testing.

Usage

The `createStub()` method works together with the `System.StubProvider` interface. You define the behavior of the stubbed object by implementing the `StubProvider` interface. Then you create a stubbed object using the `createStub()` method. When you invoke methods on the stubbed object, the `handleMethodCall()` method of the `StubProvider` interface is called to perform the behavior of the stubbed method.

See Also

Creates an instance of a stubbed SObject type that you can use to provide testing results in the extended `System.SoqlStubProvider` class.

Signature

```
public static SObject createStubQueryRow(Schema.SObjectType targetType, Map<String,Object>
fieldMapWithRelationshipKeys)
```

Parameters

targetType

Type: Schema.SObjectType

The SObject type to be stubbed. This parameter can't be null.

fieldMapWithRelationshipKeys

Type: Map<String,Object>

The map contains the fields for a parent entity, keyed by the field name with a value for each field. Key and value pairs can also be used for an aggregate relationship. The key holds the name of the aggregate relationship and the value is a list of SObjects.

Return Value

Type: SObject

Returns the stubbed SObject to use in testing.

Example

```
ssot__EmailEngagement__dlm engagement = (ssot__EmailEngagement__dlm)Test.createStubQueryRo
    new Map<string, object> {
        'ssot__Name__c' => 'My Email Engagement',
        'ssot__CityName__c' => 'San Francisco'
    }
);
```

See Also

- *Apex Developer Guide*: Mock SOQL Tests for Data Cloud Data Model Objects

createStubQueryRows(targetType, fieldMapWithRelationshipKeysForMultipleRows)

Creates instances of stubbed SObject types that you can use to provide testing results in the extended `System.SoqlStubProvider` class.

Signature

```
public static List<SObject> createStubQueryRows(Schema.SObjectType targetType,
List<Map<String,Object>> fieldMapWithRelationshipKeysForMultipleRows)
```

Parameters

targetType

Type: Schema.SObjectType

The SObject type to be stubbed. This parameter can't be null.

each field. Key and value pairs can also be used for an aggregate relationship used in the query. The key holds the name of the aggregate relationship and the value is a list of SObjects.

Return Value

Type: List<SObject>

Returns a list of stubbed SObject types to use in testing.

Example

```
List<Map<String, Object>> engagementMaps = new List<Map<String, Object>>();
Map<String, Object> engagement = new Map<String, Object> {
        'ssot__Name__c' => 'My Email Engagement',
        'ssot__CityName__c' => 'San Francisco'
};
Map<String, Object> engagement2 = new Map<String, Object> {
        'ssot__Name__c' => 'My Other Email Engagement',
        'ssot__CityName__c' => 'New York'
};

engagementMaps.add(engagement);
engagementMaps.add(engagement2);

List<ssot__EmailEngagement__dlm> engagements = (List<ssot__EmailEngagement__dlm>)Test.crea
    engagementMaps);
```

See Also

- *Apex Developer Guide*: Mock SOQL Tests for Data Cloud Data Model Objects

enableChangeDataCapture()

Use this method in an Apex test so that change event notifications are generated for all supported Change Data Capture entities. Call this method at the beginning of your test before performing DML operations and calling `Test.getEventBus().deliver();`.

Signature

```
public static void enableChangeDataCapture()
```

Return Value

Type: void

Usage

The `enableChangeDataCapture()` method ensures that Apex tests can fire change event triggers regardless of the entities selected in Setup in the Change Data Capture page. The `enableChangeDataCapture()` method doesn't affect the entities selected in Setup.

See Also

- *Change Data Capture Developer Guide*

enqueueBatchJobs(numberOfJobs)

Adds the specified number of jobs with no-operation contents to the test-context queue. It first fills the test batch queue, up to the maximum 5 jobs, and then places jobs in the test flex queue. It

```
public static List<Id> enqueueBatchJobs(Integer numberOfJobs)
```

Parameters

numberOfJobs

Type: Integer

Number of test jobs to enqueue.

Return Value

Type: List<Id>

A list of IDs of enqueued test jobs.

Usage

Use this method to reduce testing time. Instead of using your org's real batch jobs for testing, you can use this method to simulate batch-job enqueueing. Using `enqueueBatchJobs(numberOfJobs)` is faster than enqueuing real batch jobs.

getEventBus()

Returns an instance of the test event bus broker, which lets you operate on platform event or change event messages in an Apex test. For example, you can call `Test.getEventBus().deliver()` to deliver event messages.

Signature

```
public static EventBus.TestBroker getEventBus()
```

Return Value

Type: EventBus.TestBroker

A broker for the test event bus.

Usage

Enclose `Test.getEventBus().deliver()` within the `Test.startTest()` and `Test.stopTest()` statement block.

```
Test.startTest();
// Create test events
// ...
// Publish test events with EventBus.publish()
// ...
// Deliver test events
Test.getEventBus().deliver();
// Perform validation
// ...
Test.stopTest();
```

See Also

- *Platform Events Developer Guide*

getFlexQueueOrder()

Returns an ordered list of job IDs for jobs in the test-context flex queue. The job at index `0` is the next job slated to run. This method returns only test-context results, even if it's annotated with

Return Value

Type: List<Id>

An ordered list of IDs of the jobs in the test's flex queue.

getStandardPricebookId()

Returns the ID of the standard price book in the organization.

Signature

```
public static Id getStandardPricebookId()
```

Return Value

Type: Id

The ID of the standard price book.

Usage

This method returns the ID of the standard price book in your organization regardless of whether the test can query organization data. By default, tests can't query organization data unless they're annotated with `@isTest(SeeAllData=true)`.

Creating price book entries with a standard price requires the ID of the standard price book. Use this method to get the standard price book ID so that you can create price book entries in your tests.

Example

This example creates some test data for price book entries. The test method in this example gets the standard price book ID and uses this ID to create a price book entry for a product with a standard price. Next, the test creates a custom price book and uses the ID of this custom price book to add a price book entry with a custom price.

```
@isTest
public class PriceBookTest {
    // Utility method that can be called by Apex tests to create price book entries.
    static testmethod void addPricebookEntries() {
        // First, set up test price book entries.
        // Insert a test product.
        Product2 prod = new Product2(Name = 'Laptop X200',
            Family = 'Hardware');
        insert prod;

        // Get standard price book ID.
        // This is available irrespective of the state of SeeAllData.
        Id pricebookId = Test.getStandardPricebookId();

        // 1. Insert a price book entry for the standard price book.
        // Standard price book entries require the standard price book ID we got earlier.
        PricebookEntry standardPrice = new PricebookEntry(
            Pricebook2Id = pricebookId, Product2Id = prod.Id,
            UnitPrice = 10000, IsActive = true);
        insert standardPrice;

        // Create a custom price book
        Pricebook2 customPB = new Pricebook2(Name='Custom Pricebook', isActive=true);
        insert customPB;

        // 2. Insert a price book entry with a custom price.
        PricebookEntry customPrice = new PricebookEntry(
            Pricebook2Id = customPB.Id, Product2Id = prod.Id,
```

invokeContinuationMethod(controller, request)

Invokes the callback method for the specified controller and continuation in a test method.

Signature

```
public static Object invokeContinuationMethod(Object controller, Continuation request)
```

Parameters

controller

Type: Object

An instance of the controller class that invokes the continuation request.

request

Type: Continuation

The continuation that is returned by an action method in the controller class.

Return Value

Type: Object

The response of the continuation callback method.

Usage

Use the `Test.setContinuationResponse` and `Test.invokeContinuationMethod` methods to test continuations. In test context, callouts of continuations aren't sent to the external service. By using these methods, you can set a mock response and cause the runtime to call the continuation callback method to process the mock response.

Call `Test.setContinuationResponse` before you call `Test.invokeContinuationMethod`. When you call `Test.invokeContinuationMethod`, the runtime executes the callback method that is associated with the continuation. The callback method processes the mock response that is set by `Test.setContinuationResponse`.

isRunningTest()

Returns `true` if the currently executing code was called by code contained in a test method, `false` otherwise. Use this method if you need to run different code depending on whether it was being called from a test.

Signature

```
public static Boolean isRunningTest()
```

Return Value

Type: Boolean

isSoqlStubDefined(targetType)

Returns `true` if a SOQL stub is defined for an SObject type; otherwise returns `false`.

Signature

```
public static Boolean isSoqlStubDefined(Schema.SObjectType targetType)
```

Parameters

Return Value

Type: Boolean

loadData(sObjectToken, resourceName)

Inserts test records from the specified static resource .csv file and for the specified sObject type, and returns a list of the inserted sObjects.

Signature

```
public static List<sObject> loadData(Schema.SObjectType sObjectToken, String resourceName)
```

Parameters

sObjectToken

Type: Schema.SObjectType

The sObject type for which to insert test records.

resourceName

Type: String

The static resource that corresponds to the .csv file containing the test records to load. The name is case insensitive.

Return Value

Type: List<sObject>

Usage

You must create the static resource prior to calling this method. The static resource is a comma-delimited file ending with a .csv extension. The file contains field names and values for the test records. The first line of the file must contain the field names and subsequent lines are the field values. To learn more about static resources, see “Defining Static Resources” in the Salesforce online help.

Once you create a static resource for your .csv file, the static resource will be assigned a MIME type. Supported MIME types are:

- text/csv
- application/vnd.ms-excel
- application/octet-stream
- text/plain

newSendEmailQuickActionDefaults(contextId, replyToId)

Creates a new QuickAction.SendEmailQuickActionDefaults instance for testing a class implementing the QuickAction.QuickActionDefaultsHandler interface.

Signature

```
public static QuickAction.SendEmailQuickActionDefaults newSendEmailQuickActionDefaults(ID contextId, ID replyToId)
```

Parameters

contextId

Type: Id

Parent record of the email message.

Return Value

Type: SendEmailQuickActionDefaults Class

The default values used for an email message quick action.

setContinuationResponse(requestLabel, mockResponse)

Sets a mock response for a continuation HTTP request in a test method.

Signature

```
public static void setContinuationResponse(String requestLabel, System.HttpResponse
mockResponse)
```

Parameters

requestLabel

Type: String

The unique label that corresponds to the continuation HTTP request. This label is returned by `Continuation.addHttpRequest`.

mockResponse

Type: HttpResponse

The fake response to be returned by `Test.invokeContinuationMethod`.

Return Value

Type: void

Usage

Use the `Test.setContinuationResponse` and `Test.invokeContinuationMethod` methods to test continuations. In test context, callouts of continuations aren't sent to the external service. By using these methods, you can set a mock response and cause the runtime to call the continuation callback method to process the mock response.

Call `Test.setContinuationResponse` before you call `Test.invokeContinuationMethod`. When you call `Test.invokeContinuationMethod`, the runtime executes the callback method that is associated with the continuation. The callback method processes the mock response that is set by `Test.setContinuationResponse`.

setCreatedDate(recordId, createdDatetime)

Sets `CreatedDate` for a test-context sObject.

Signature

```
public static void setCreatedDate(Id recordId, Datetime createdDatetime)
```

Parameters

recordId

Type: Id

The ID of an sObject.

createdDatetime

Type: Datetime

The value to assign to the sObject's `CreatedDate` field.

All database changes are rolled back at the end of a test. You can't use this method on records that existed before your test executed. You also can't use `setCreatedDate` in methods annotated with `@isTest(SeeAllData=true)`, because those methods have access to all data in your org. If you set `CreatedDate` to a future value, it can cause unexpected results. This method takes two parameters–an sObject ID and a Datetime value–neither of which can be null.

Insert your test record before you set its `CreatedDate`, as shown in this example.

```
@isTest
private class SetCreatedDateTest {
    static testMethod void testSetCreatedDate() {
        Account a = new Account(name='myAccount');
        insert a;
        Test.setCreatedDate(a.Id, DateTime.newInstance(2012,12,12));
        Test.startTest();
        Account myAccount = [SELECT Id, Name, CreatedDate FROM Account
                             WHERE Name ='myAccount' limit 1];
        System.assertEquals(myAccount.CreatedDate, DateTime.newInstance(2012,12,12));
        Test.stopTest();
    }
}
```

setCurrentPage(page)

A Visualforce test method that sets the current PageReference for the controller.

Signature

```
public static Void setCurrentPage(PageReference page)
```

Parameters

page

Type: System.PageReference

Return Value

Type: Void

setCurrentPageReference(page)

A Visualforce test method that sets the current PageReference for the controller.

Signature

```
public static Void setCurrentPageReference(PageReference page)
```

Parameters

page

Type: System.PageReference

Return Value

Type: Void

setFixedSearchResults(fixedSearchResults)

Defines a list of fixed search results to be returned by all subsequent SOSL statements in a test method.

fixedSearchResults

Type: ID[]

The list of record IDs specified by *opt_set_search_results* replaces the results that would normally be returned by the SOSL queries if they were not subject to any `WHERE` or `LIMIT` clauses. If these clauses exist in the SOSL queries, they are applied to the list of fixed search results.

Return Value

Type: Void

Usage

If *opt_set_search_results* is not specified, all subsequent SOSL queries return no results.

For more information, see Dynamic SOSL.

setMock(interfaceType, instance)

Sets the response mock mode and instructs the Apex runtime to send a mock response whenever a callout is made through the HTTP classes or the auto-generated code from WSDLs.

Signature

```
public static Void setMock(Type interfaceType, Object instance)
```

Parameters

interfaceType

Type: System.Type

instance

Type: Object

Return Value

Type: Void

Usage

> **Note**
>
> To mock a callout if the code that performs the callout is in a managed package, call `Test.setMock` from a test method in the same package with the same namespace.

setReadOnlyApplicationMode(applicationMode)

Sets the application mode for an organization to read-only in an Apex test to simulate read-only mode during Salesforce upgrades and downtimes. The application mode is reset to the default mode at the end of each Apex test run.

Signature

```
public static Void setReadOnlyApplicationMode(Boolean applicationMode)
```

Parameters

applicationMode

Type: Boolean

Also see the `getApplicationReadWriteMode()` System method.

Do not use `setReadOnlyApplicationMode` for purposes unrelated to Read-Only Mode testing, such as simulating DML exceptions.

Example

The following example sets the application mode to read-only and attempts to insert a new account record, which results in the exception. It then resets the application mode and performs a successful insert.

```
@isTest
private class ApplicationReadOnlyModeTestClass {
  public static testmethod void test() {
    // Create a test account that is used for querying later.
    Account testAccount = new Account(Name = 'TestAccount');
    insert testAccount;

    // Set the application read only mode.
    Test.setReadOnlyApplicationMode(true);

    // Verify that the application is in read-only mode.
    System.assertEquals(
              ApplicationReadWriteMode.READ_ONLY,
              System.getApplicationReadWriteMode());

    // Create a new account object.
    Account testAccount2 = new Account(Name = 'TestAccount2');

    try {
      // Get the test account created earlier. Should be successful.
      Account testAccountFromDb =
        [SELECT Id, Name FROM Account WHERE Name = 'TestAccount'];
      System.assertEquals(testAccount.Id, testAccountFromDb.Id);

      // Inserts should result in the InvalidReadOnlyUserDmlException
      // being thrown.
      insert testAccount2;
      System.assertEquals(false, true);
    } catch (System.InvalidReadOnlyUserDmlException e) {
      // Expected
    }
    // Insertion should work after read only application mode gets disabled.
    Test.setReadOnlyApplicationMode(false);

    insert testAccount2;
    Account testAccount2FromDb =
       [SELECT Id, Name FROM Account WHERE Name = 'TestAccount2'];
    System.assertEquals(testAccount2.Id, testAccount2FromDb.Id);
  }
}
```

startTest()

Marks the point in your test code when your test actually begins. Use this method when you are testing governor limits.

Signature

```
public static Void startTest()
```

Return Value

Type: Void

variables, populate data structures, and so on, allowing you to set up everything you need to run your test. Any code that executes after the call to `startTest` and before `stopTest` is assigned a new set of governor limits.

stopTest()

Marks the point in your test code when your test ends. Use this method in conjunction with the `startTest` method.

Signature

```
public static Void stopTest()
```

Return Value

Type: Void

Usage

Each test method is allowed to call this method only once. Any code that executes after the `stopTest` method is assigned the original limits that were in effect before `startTest` was called. All asynchronous calls made after the `startTest` method are collected by the system. When `stopTest` is executed, all asynchronous processes are run synchronously.

> **Note**
>
> Asynchronous calls, such as `@future` or `executeBatch`, called in a `startTest, stopTest` block, do not count against your limits for the number of queued jobs.

testInstall(installImplementation, version, isPush)

Tests the implementation of the InstallHandler interface, which is used for specifying a post install script in packages. Tests run as the test initiator in the development environment.

Signature

```
public static Void testInstall(InstallHandler installImplementation, Version version, Boolean isPush)
```

Parameters

installImplementation

Type: System.InstallHandler

A class that implements the `InstallHandler` interface.

version

Type: System.Version

The version number of the existing package installed in the subscriber organization.

isPush

Type: Boolean

(Optional) Specifies whether the upgrade is a push. The default value is `false`.

Return Value

Type: Void

Usage

```
@isTest static void test() {
  PostInstallClass postinstall =
    new PostInstallClass();
    Test.testInstall(postinstall,
      new Version(1,0));
  }
```

testSandboxPostCopyScript(script, organizationId, sandboxId, sandboxName)

Tests the implementation of the SandboxPostCopy Interface, which is used for specifying a script to run at the completion of a Sandbox copy. Tests run as the test initiator in the development environment.

Signature

```
public static void testSandboxPostCopyScript(System.SandboxPostCopy script, Id organizationId,
Id sandboxId, String sandboxName)
```

Parameters

script

Type: System.SandboxPostCopy

A class that implements the `SandboxPostCopy` interface.

organizationId

Type: Id

The sandbox organization ID

sandboxId

Type: Id

The sandbox ID to be provided to the SandboxPostCopy script.

sandboxName

Type: String

The sandbox name to be provided to the SandboxPostCopy script.

Return Value

Type: void

Usage

This method throws a run-time exception if the test install fails.

> **Note**
>
> Salesforce recommends that you use the `testSandboxPostCopyScript(script, organizationId, sandboxId, sandboxName, isRunAsAutoProcUser)` overload instead of this method. When `isRunAsAutoProcUser` is `true`, the `SandboxPostCopy` script is tested with the same user access permissions as used by post-copy tasks during sandbox creation. Using the same permissions enables the test to better simulate the actual usage of the class, and to uncover potential issues.

Example

Tests the implementation of the SandboxPostCopy Interface, which is used for specifying a script to run at the completion of a Sandbox copy. When `RunAsAutoProcUser` is `true`, tests run as Automated Process user in the development environment.

Signature

```
public static void testSandboxPostCopyScript(System.SandboxPostCopy script, Id organizationId,
Id sandboxId, String sandboxName, Boolean RunAsAutoProcUser)
```

Parameters

script

Type: System.SandboxPostCopy

A class that implements the `SandboxPostCopy` interface.

organizationId

Type: Id

The sandbox organization ID.

sandboxId

Type: Id

The sandbox ID to be provided to the SandboxPostCopy script.

sandboxName

Type: String

The sandbox name to be provided to the SandboxPostCopy script.

RunAsAutoProcUser

Type: Boolean

When `true`, the `SandboxPostCopy` script is tested with the same user access permissions as used by post-copy tasks during sandbox creation. Using the same permissions enables the test to better simulate the actual usage of the class, and to uncover potential issues.

When `false`, the test runs as the test initiator. This option can alter the permissions with which the script is tested, such as the ability to access objects and features.

Return Value

Type: void

Usage

This method throws a run-time exception if the test install fails.

Example

See SandboxPostCopy Example Implementation

testUninstall(uninstallImplementation)

Tests the implementation of the UninstallHandler interface, which is used for specifying an uninstall script in packages. Tests run as the test initiator in the development environment.

Signature

```
public static Void testUninstall(UninstallHandler uninstallImplementation)
```

A class that implements the `UninstallHandler` interface.

Return Value

Type: Void

Usage

This method throws a run-time exception if the test uninstall fails.

Example

```
@isTest static void test() {
  UninstallClass uninstall =
    new UninstallClass();
    Test.testUninstall(uninstall);
  }
```