

APEX REFERENCE GUIDE

Apex is a strongly typed, object-oriented programming language that allows developers to execute flow and transaction control statements on the Salesforce Platform server, in conjunction with calls to the API. This reference guide includes built-in Apex classes, interfaces, enums, and exceptions, grouped by namespace. It also includes Apex DML statements to insert, update, merge, delete, and restore data in Salesforce.

For information on the Apex development process, see [Apex Developer Guide](#).



Note: In API version 51.0 and earlier, Apex Reference information was included in the Apex Developer Guide in the **Apex Language Reference** section.

IN THIS SECTION:

[Apex Release Notes](#)

Use the Salesforce Release Notes to learn about the most recent updates and changes to Apex.

[Apex DML Operations](#)

You can perform DML operations using the Apex DML statements or the methods of the `Database` class. For lead conversion, use the `convertLead` method of the `Database` class. There is no DML counterpart for it.

[ApexPages Namespace](#)

The `ApexPages` namespace provides classes used in Visualforce controllers.

[AppLauncher Namespace](#)

The `AppLauncher` namespace provides methods for managing the appearance of apps in the App Launcher, including their visibility and sort order.

[Approval Namespace](#)

The `Approval` namespace provides classes and methods for approval processes.

[Auth Namespace](#)

The `Auth` namespace provides an interface and classes for single sign-on into Salesforce and session security management.

[Cache Namespace](#)

The `Cache` namespace contains methods for managing the platform cache.

[Canvas Namespace](#)

The `Canvas` namespace provides an interface and classes for canvas apps in Salesforce.

[ChatterAnswers Namespace](#)

The `ChatterAnswers` namespace provides an interface for creating Account records.

[CommerceExtension Namespace](#)

Use the `CommerceExtension` namespace to define resolution strategies for registered Commerce extensions.

[CommerceOrders Namespace](#)

The `CommerceOrders` namespace provides classes and methods to place orders with integrated pricing, configuration, and validation.

[CommercePayments Namespace](#)

Use the `CommercePayments` namespace to provide a safe and customizable platform for managing customer payments and refunds.

[CommerceTax Namespace](#)

Manage the communication between Salesforce and an external tax engine.

[Compression Namespace \(Developer Preview\)](#)

The `Compression` namespace provides classes and methods to create and extract zip files.

[ConnectApi Namespace](#)

The `ConnectApi` namespace (also called Connect in Apex) provides classes for accessing the same data available in Connect REST API. Use Connect in Apex to create custom experiences in Salesforce.

[Context Namespace](#)

The `Context` namespace provides classes and methods to manage the sharing and consumption of business application data by using Context Service.

[Database Namespace](#)

The `Database` namespace provides classes used with DML operations.

[Datacloud Namespace](#)

The `Datacloud` namespace provides classes and methods for retrieving information about duplicate rules. Duplicate rules let you control whether and when users can save duplicate records within Salesforce.

[DataRetrieval Namespace](#)

The `DataRetrieval` namespace provides classes and methods to record details of customer-agent engagements, as well as transcripts of their conversations.

[DataSource Namespace](#)

The `DataSource` namespace provides the classes for the Apex Connector Framework. Use the Apex Connector Framework to develop a custom adapter for Salesforce Connect. Then connect your Salesforce organization to any data anywhere via the Salesforce Connect custom adapter.

[DataWeave Namespace](#)

The `DataWeave` namespace provides classes and methods to support the invocation of DataWeave scripts from Apex.

[Dom Namespace](#)

The `Dom` namespace provides classes and methods for parsing and creating XML content.

[EventBus Namespace](#)

The `EventBus` namespace provides classes and methods for platform events and Change Data Capture events.

[ExternalService Namespace](#)

The `ExternalService` namespace provides dynamically generated Apex service interfaces and Apex classes for complex object data types.

[Flow Namespace](#)

The `Flow` namespace provides a class for advanced access to flows from Apex such as from Visualforce controllers and asynchronous Apex.

[FormulaEval Namespace \(Beta\)](#)

The `FormulaEval` namespace provides classes and methods to evaluate user-defined dynamic formulas for Apex objects and SObject types. Use the methods to avoid unnecessary DML statements to recalculate formula field values or evaluate dynamic formula expressions.

[fscashflow Namespace](#)

The `fscashflow` namespace provides classes used in the FSCashFlow Flexcards and its child Flexcards.

[Functions Namespace](#)

The `Functions` namespace provides classes and methods used to invoke and manage Salesforce Functions.

[industriesNlpSvc](#)

Stores the objects used in Industries Einstein Natural Language Processing (NLP) services.

[IndustriesDigitalLending Namespace](#)

The `industriesDigitalLending` namespace provides classes used in the Digital Lending OmniScripts and Integration Procedures.

[Invocable Namespace](#)

The `Invocable` namespace provides classes for calling invocable actions from Apex.

[IsvPartners Namespace](#)

The `IsvPartners` namespace provides a class associated with Salesforce ISV partner use cases, such as optimizing code, providing great customer trial experiences, and driving feature adoption.

[KbManagement Namespace](#)

The `KbManagement` namespace provides a class for managing knowledge articles.

[LxScheduler Namespace](#)

The `LxScheduler` namespace provides an interface and classes for integrating Salesforce Scheduler with external calendars.

[Messaging Namespace](#)

The `Messaging` namespace provides classes and methods for Salesforce outbound and inbound email functionality.

[Metadata Namespace](#)

The `Metadata` namespace provides classes and methods for working with custom metadata in Salesforce

[PlaceQuote Namespace](#)

The `PlaceQuote` namespace provides classes and methods to create or update quotes with pricing preferences and configuration options.

[Pref_center Namespace](#)

The `Pref_center` namespace provides an interface, classes, and methods to create and retrieve data in forms in Preference Manager. Preference Manager, previously called Preference Center, is a feature within the Privacy Center app.

[Process Namespace](#)

The `Process` namespace provides an interface and classes for passing data between your organization and a flow.

[QuickAction Namespace](#)

The `QuickAction` namespace provides classes and methods for quick actions.

[Reports Namespace](#)

The `Reports` namespace provides classes for accessing the same data as is available in the Salesforce Reports and Dashboards REST API.

[RichMessaging Namespace](#)

Provides objects and methods for handling content in enhanced Messaging channels.

[Schema Namespace](#)

The `Schema` namespace provides classes and methods for schema metadata information.

[Search Namespace](#)

The `Search` namespace provides classes for getting search results and suggestion results.

[Sfc Namespace](#)

The `Sfc` namespace contains classes used in Salesforce Files.

[Sfdc_Checkout Namespace](#)

The `Sfdc_Checkout` namespace provides an interface and classes for B2B Commerce apps in Salesforce.

[Sfdc_Enablement Namespace](#)

The `sfdc_enablement` namespace provides classes for creating custom learning items to implement custom exercise types in Enablement programs. Lightning web components are used to render the custom exercises on Program Builder.

[sfdc_surveys Namespace](#)

The `sfdc_surveys` namespace provides an interface for shortening survey invitations.

[Site Namespace](#)

The `site` namespace provides an interface for rewriting Sites URLs.

[Slack Namespace](#)

The `Slack` Namespace provides tools designed to accelerate and ease the process of developing Slack apps on the Salesforce platform.

[Support Namespace](#)

The `Support` namespace provides an interface used for Case Feed.

[System Namespace](#)

The `System` namespace provides classes and methods for core Apex functionality.

[TerritoryMgmt Namespace](#)

The `TerritoryMgmt` namespace provides an interface used for territory management.

[TxnSecurity Namespace](#)

The `TxnSecurity` namespace provides an interface used for transaction security.

[UserProvisioning Namespace](#)

The `UserProvisioning` namespace provides methods for monitoring outbound user provisioning requests.

[VisualEditor Namespace](#)

The `VisualEditor` namespace provides classes and methods for interacting with the Lightning App Builder. The classes and methods in this namespace operate on Lightning components, which include Lightning web components and Aura components.

[Wave Namespace](#)

The classes in the `wave` namespace are part of the CRM Analytics Analytics SDK, designed to facilitate querying CRM Analytics data from Apex code.

[Appendices](#)

Apex Release Notes

Use the Salesforce Release Notes to learn about the most recent updates and changes to Apex.

For Apex updates and changes that impact the Salesforce Platform, see the [Apex Release Notes](#).

For new and changed Apex classes, methods, exceptions and interfaces, see [Apex: New and Changed Items](#) in the Salesforce Release Notes.

Apex DML Operations

You can perform DML operations using the Apex DML statements or the methods of the `Database` class. For lead conversion, use the `convertLead` method of the `Database` class. There is no DML counterpart for it.

SEE ALSO:

[Apex Developer Guide: Working with Data in Apex](#)
[Database Class](#)

Apex DML Statements

Use Data Manipulation Language (DML) statements to insert, update, merge, delete, and restore data in Salesforce.

The following Apex DML statements are available:

Insert Statement

The `insert` DML operation adds one or more sObjects, such as individual accounts or contacts, to your organization's data. `insert` is analogous to the INSERT statement in SQL.

Syntax

```
insert sObject  
insert sObject[]
```

Example

The following example inserts an account named 'Acme':

```
Account newAcct = new Account(name = 'Acme');  
try {  
    insert newAcct;  
} catch (DmlException e) {  
    // Process exception here  
}
```

 **Note:** For more information on processing `DmlExceptions`, see [Bulk DML Exception Handling](#).

Update Statement

The `update` DML operation modifies one or more existing sObject records, such as individual accounts or contacts, in your organization's data. `update` is analogous to the UPDATE statement in SQL.

Syntax

```
update sObject  
update sObject[]
```

Example

The following example updates the `BillingCity` field on a single account named 'Acme':

```
Account a = new Account (Name='Acme2');
insert(a);

Account myAcct = [SELECT Id, Name, BillingCity FROM Account WHERE Id = :a.Id];
myAcct.BillingCity = 'San Francisco';

try {
    update myAcct;
} catch (DmlException e) {
    // Process exception here
}
```

 **Note:** For more information on processing `DmlExceptions`, see [Bulk DML Exception Handling](#).

Upsert Statement

The `upsert` DML operation creates new records and updates sObject records within a single statement, using a specified field to determine the presence of existing objects, or the ID field if no field is specified.

Syntax

```
upsert sObject [opt_field]
```

```
upsert sObject[] [opt_field]
```


The `upsert` statement matches the sObjects with existing records by comparing values of one field. If you don't specify a field when calling this statement, the `upsert` statement uses the sObject's ID to match the sObject with existing records in Salesforce. Alternatively, you can specify a field to use for matching. For custom objects, specify a custom field marked as external ID. For standard objects, you can specify any field that has the `idLookup` attribute set to true. For example, the Email field of Contact or User has the `idLookup` attribute set. To check a field's attribute, see the [Object Reference for Salesforce](#).

Also, you can use foreign keys to upsert sObject records if they have been set as reference fields. For more information, see [Field Types](#) in the *Object Reference for Salesforce*.

The optional field parameter, `opt_field`, is a field token (of type `Schema.SObjectField`). For example, to specify the `MyExternalId` custom field, the statement is:

```
upsert sObjectList Account.Fields.MyExternalId__c;
```

If the field used for matching doesn't have the `Unique` attribute set, the context user must have the "View All" object-level permission for the target object or the "View All Data" permission so that `upsert` doesn't accidentally insert a duplicate record.

 **Note:** Custom field matching is case-insensitive only if the custom field has the **Unique** and **Treat "ABC" and "abc" as duplicate values (case insensitive)** attributes selected as part of the field definition. If so, "ABC123" is matched with "abc123." For more information, see "Create Custom Fields" in the Salesforce online help.

How Upsert Chooses to Insert or Update

Upsert uses the sObject record's primary key (the ID), an `idLookup` field, or an external ID field to determine whether it should create a record or update an existing one:

- If the key isn't matched, a new object record is created.
- If the key is matched once, the existing object record is updated.
- If the key is matched multiple times, an error is generated and the object record isn't inserted or updated.

Example

This example performs an upsert of a list of accounts.

```
List<Account> acctList = new List<Account>();
// Fill the accounts list with some accounts

try {
    upsert acctList;
} catch (DmlException e) {

}
```

This next example performs an upsert of a list of accounts using a foreign key for matching existing records, if any.

```
List<Account> acctList = new List<Account>();
// Fill the accounts list with some accounts

try {
    // Upsert using an external ID field
    upsert acctList myExtIDField__c;
} catch (DmlException e) {

}
```

Delete Statement

The `delete` DML operation deletes one or more existing sObject records, such as individual accounts or contacts, from your organization's data. `delete` is analogous to the `delete ()` statement in the SOAP API.

Syntax

```
delete sObject
```

```
delete sObject[]
```

Example

The following example deletes all accounts that are named 'DotCom':

```
Account[] doomedAccts = [SELECT Id, Name FROM Account
                          WHERE Name = 'DotCom'];

try {
    delete doomedAccts;
} catch (DmlException e) {
    // Process exception here
}
```



Note: For more information on processing `DmlExceptions`, see [Bulk DML Exception Handling](#).

Undelete Statement

The **undelete** DML operation restores one or more existing sObject records, such as individual accounts or contacts, from your organization's Recycle Bin. **undelete** is analogous to the UNDELETE statement in SQL.

Syntax

```
undelete sObject | ID
undelete sObject[] | ID[]
```

Example


The following example undeletes an account named 'Universal Containers'. The **ALL ROWS** keyword queries all rows for both top level and aggregate relationships, including deleted records and archived activities.

```
Account[] savedAccts = [SELECT Id, Name FROM Account WHERE Name = 'Universal Containers'
ALL ROWS];
try {
    undelete savedAccts;
} catch (DmlException e) {
    // Process exception here
}
```

 **Note:** For more information on processing DmlExceptions, see [Bulk DML Exception Handling](#).

Merge Statement

The **merge** statement merges up to three records of the same sObject type into one of the records, deleting the others, and re-parenting any related records.

 **Note:** This DML operation does not have a matching Database system method.

Syntax

```
merge sObject sObject
merge sObject sObject[]
merge sObject ID
merge sObject ID[]
```

The first parameter represents the master record into which the other records are to be merged. The second parameter represents the one or two other records that should be merged and then deleted. You can pass these other records into the **merge** statement as a single sObject record or ID, or as a list of two sObject records or IDs.

Example

The following example merges two accounts named 'Acme Inc.' and 'Acme' into a single record:

```
List<Account> ls = new List<Account>{new Account(name='Acme Inc.'), new Account(name='Acme')};
insert ls;
Account masterAcct = [SELECT Id, Name FROM Account WHERE Name = 'Acme Inc.' LIMIT 1];
```



```
Account mergeAcct = [SELECT Id, Name FROM Account WHERE Name = 'Acme' LIMIT 1];
try {
    merge masterAcct mergeAcct;
} catch (DmlException e) {
    // Process exception here
}
```

 **Note:** For more information on processing `DmlExceptions`, see [Bulk DML Exception Handling](#).

ApexPages Namespace

The `ApexPages` namespace provides classes used in Visualforce controllers.

The following are the classes in the `ApexPages` namespace.

IN THIS SECTION:

[Action Class](#)

You can use `ApexPages.Action` to create an action method that you can use in a Visualforce custom controller or controller extension.

[Component Class](#)

Represents a dynamic Visualforce component in Apex.

[IdeaStandardController Class](#)

`IdeaStandardController` objects offer Ideas-specific functionality in addition to what is provided by the `StandardController`.

[IdeaStandardSetController Class](#)

`IdeaStandardSetController` objects offer Ideas-specific functionality in addition to what is provided by the `StandardSetController`.

[KnowledgeArticleVersionStandardController Class](#)

`KnowledgeArticleVersionStandardController` objects offer article-specific functionality in addition to what is provided by the `StandardController`.

[Message Class](#)

Contains validation errors that occur when the user saves the page that uses a standard controller.

[StandardController Class](#)

Use a `StandardController` when defining an extension for a standard controller.

[StandardSetController Class](#)

`StandardSetController` objects allow you to create list controllers similar to, or as extensions of, the pre-built Visualforce list controllers provided by Salesforce.

Action Class

You can use `ApexPages.Action` to create an action method that you can use in a Visualforce custom controller or controller extension.

Namespace

[ApexPages](#)

Usage

For example, you could create a `saveOver` method on a controller extension that performs a custom save.

Instantiation

The following code snippet illustrates how to instantiate a new `ApexPages.Action` object that uses the save action:

```
ApexPages.Action saveAction = new ApexPages.Action('{!save}');
```

IN THIS SECTION:

[Action Constructors](#)

[Action Methods](#)

Action Constructors

The following are constructors for `Action`.

IN THIS SECTION:

[Action\(action\)](#)

Creates a new instance of the `ApexPages.Action` class using the specified action.

Action(action)

Creates a new instance of the `ApexPages.Action` class using the specified action.

Signature

```
public Action(String action)
```

Parameters

action

Type: [String](#)

The action.

Action Methods

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

IN THIS SECTION:

[getExpression\(\)](#)

Returns the expression that is evaluated when the action is invoked.

[invoke\(\)](#)

Invokes the action.

getExpression()

Returns the expression that is evaluated when the action is invoked.

Signature

```
public String getExpression()
```

Return Value

Type: [String](#)

invoke()

Invokes the action.

Signature

```
public System.PageReference invoke()
```

Return Value

Type: [System.PageReference](#)

Component Class

Represents a dynamic Visualforce component in Apex.

Namespace

[ApexPages](#)

Dynamic Component Properties

The following are properties for `Component`.

IN THIS SECTION:

[childComponents](#)

Returns a reference to the child components for the component.

expressions

Sets the content of an attribute using the expression language notation. The notation for this is `expressions.name_of_attribute`.

facets

Sets the content of a facet to a dynamic component. The notation is `facet.name_of_facet`.

childComponents

Returns a reference to the child components for the component.

Signature

```
public List <ApexPages.Component> childComponents {get; set;}
```

Property Value

Type: `List<ApexPages.Component>`

Example

```
Component.Apex.PageBlock pageBlk = new Component.Apex.PageBlock();

Component.Apex.PageBlockSection pageBlkSection = new
Component.Apex.PageBlockSection(title='dummy header');

pageBlk.childComponents.add(pageBlkSection);
```

expressions

Sets the content of an attribute using the expression language notation. The notation for this is `expressions.name_of_attribute`.

Signature

```
public String expressions {get; set;}
```

Property Value

Type: `String`

Example

```
Component.Apex.InputField inpFld = new
Component.Apex.InputField();
inpFld.expressions.value = '{!Account.Name}';
inpFld.expressions.id = '{!$User.FirstName}';
```

facets

Sets the content of a facet to a dynamic component. The notation is `facet.name_of_facet`.

Signature

```
public String facets {get; set;}
```

Property Value

Type: [String](#)

Usage



Note: This property is only accessible by components that support facets.

Example

```
Component.Apex.DataTable myDT = new  
Component.Apex.DataTable();  
Component.Apex.OutputText footer = new  
Component.Apex.OutputText(value='Footer Copyright');  
myDT.facets.footer = footer;
```

IdeaStandardController Class

`IdeaStandardController` objects offer Ideas-specific functionality in addition to what is provided by the `StandardController`.

Namespace

[ApexPages](#)

Usage

A method in the `IdeaStandardController` object is called by and operated on a particular instance of an `IdeaStandardController`.



Note: The `IdeaStandardSetController` and `IdeaStandardController` classes are currently available through a limited release program. For information on enabling these classes for your organization, contact your Salesforce representative.

In addition to the methods listed in this class, the `IdeaStandardController` class inherits all the methods associated with the `StandardController` class.

Instantiation

An `IdeaStandardController` object cannot be instantiated. An instance can be obtained through a constructor of a custom extension controller when using the standard ideas controller.

Example

The following example shows how an `IdeaStandardController` object can be used in the constructor for a custom list controller. This example provides the framework for manipulating the comment list data before displaying it on a Visualforce page.

```
public class MyIdeaExtension {

    private final ApexPages.IdeaStandardController ideaController;

    public MyIdeaExtension(ApexPages.IdeaStandardController controller) {
        ideaController = (ApexPages.IdeaStandardController)controller;
    }

    public List<IdeaComment> getModifiedComments() {
        IdeaComment[] comments = ideaController.getCommentList();
        // modify comments here
        return comments;
    }

}
```

The following Visualforce markup shows how the `IdeaStandardController` example shown above can be used in a page. This page must be named `detailPage` for this example to work.



Note: For the Visualforce page to display the idea and its comments, in the following example you need to specify the ID of a specific idea (for example, `/apex/detailPage?id=<ideaID>`) whose comments you want to view.

```
<!-- page named detailPage -->
<apex:page standardController="Idea" extensions="MyIdeaExtension">
    <apex:pageBlock title="Idea Section">
        <ideas:detailOutputLink page="detailPage" ideaId="{!idea.id}">{!idea.title}
        </ideas:detailOutputLink>
        <br/><br/>
        <apex:outputText >{!idea.body}</apex:outputText>
    </apex:pageBlock>
    <apex:pageBlock title="Comments Section">
        <apex:dataList var="a" value="{!modifiedComments}" id="list">
            {!a.commentBody}
        </apex:dataList>
        <ideas:detailOutputLink page="detailPage" ideaId="{!idea.id}"
            pageOffset="-1">Prev</ideas:detailOutputLink>
        |
        <ideas:detailOutputLink page="detailPage" ideaId="{!idea.id}"
            pageOffset="1">Next</ideas:detailOutputLink>
    </apex:pageBlock>
</apex:page>
```

SEE ALSO:

[StandardController Class](#)

IdeaStandardController Methods

The following are instance methods for `IdeaStandardController`.

IN THIS SECTION:[getCommentList\(\)](#)

Returns the list of read-only comments from the current page.

getCommentList()

Returns the list of read-only comments from the current page.

Signature

```
public IdeaComment[] getCommentList()
```

Return Value

Type: `IdeaComment[]`

This method returns the following comment properties:

- `id`
- `commentBody`
- `createdDate`
- `createdBy.Id`
- `createdBy.communityNickname`

IdeaStandardSetController Class

`IdeaStandardSetController` objects offer Ideas-specific functionality in addition to what is provided by the `StandardSetController`.

Namespace

[ApexPages](#)

Usage

Note: The `IdeaStandardSetController` and `IdeaStandardController` classes are currently available through a limited release program. For information on enabling these classes for your organization, contact your Salesforce representative.

In addition to the method listed above, the `IdeaStandardSetController` class inherits the methods associated with the `StandardSetController`.



Note: The methods inherited from the `StandardSetController` cannot be used to affect the list of ideas returned by the `getIdeaList` method.

Instantiation

An `IdeaStandardSetController` object cannot be instantiated. An instance can be obtained through a constructor of a custom extension controller when using the standard list controller for ideas.

Example: Displaying a Profile Page

The following example shows how an `IdeaStandardSetController` object can be used in the constructor for a custom list controller:

```
public class MyIdeaProfileExtension {
    private final ApexPages.IdeaStandardSetController ideaSetController;

    public MyIdeaProfileExtension(ApexPages.IdeaStandardSetController controller) {
        ideaSetController = (ApexPages.IdeaStandardSetController)controller;
    }

    public List<Idea> getModifiedIdeas() {
        Idea[] ideas = ideaSetController.getIdeaList();
        // modify ideas here
        return ideas;
    }
}
```

The following Visualforce markup shows how the `IdeaStandardSetController` example shown above and the `<ideas:profileListOutputLink>` component can display a profile page that lists the recent replies, submitted ideas, and votes associated with a user. Because this example does not identify a specific user ID, the page automatically shows the profile page for the current logged in user. This page must be named `profilePage` in order for this example to work:

```
<!-- page named profilePage -->
<apex:page standardController="Idea" extensions="MyIdeaProfileExtension"
recordSetVar="ideaSetVar">
    <apex:pageBlock >
        <ideas:profileListOutputLink sort="recentReplies" page="profilePage">
            Recent Replies</ideas:profileListOutputLink>
        |
        <ideas:profileListOutputLink sort="ideas" page="profilePage">Ideas Submitted
        </ideas:profileListOutputLink>
        |
        <ideas:profileListOutputLink sort="votes" page="profilePage">Ideas Voted
        </ideas:profileListOutputLink>
    </apex:pageBlock>
    <apex:pageBlock >
        <apex:dataList value="{!modifiedIdeas}" var="ideadata">
            <ideas:detailOutputLink ideaId="{!ideadata.id}" page="viewPage">
                {!ideadata.title}</ideas:detailOutputLink>
            </apex:dataList>
        </apex:pageBlock>
    </apex:page>
```

In the previous example, the `<ideas:detailOutputLink>` component links to the following Visualforce markup that displays the detail page for a specific idea. This page must be named `viewPage` in order for this example to work:

```
<!-- page named viewPage -->
<apex:page standardController="Idea">
    <apex:pageBlock title="Idea Section">
        <ideas:detailOutputLink page="viewPage" ideaId="{!idea.id}">{!idea.title}
        </ideas:detailOutputLink>
    <br/><br/>
```




```

        <apex:outputText>{!idea.body}</apex:outputText>
    </apex:pageBlock>
</apex:page>

```

Example: Displaying a List of Top, Recent, and Most Popular Ideas and Comments

The following example shows how an `IdeaStandardSetController` object can be used in the constructor for a custom list controller:

 **Note:** You must have created at least one idea for this example to return any ideas.

```

public class MyIdeaListExtension {
    private final ApexPages.IdeaStandardSetController ideaSetController;

    public MyIdeaListExtension (ApexPages.IdeaStandardSetController controller) {
        ideaSetController = (ApexPages.IdeaStandardSetController)controller;
    }

    public List<Idea> getModifiedIdeas() {
        Idea[] ideas = ideaSetController.getIdeaList();
        // modify ideas here
        return ideas;
    }
}

```

The following Visualforce markup shows how the `IdeaStandardSetController` example shown above can be used with the `<ideas:listOutputLink>` component to display a list of recent, top, and most popular ideas and comments. This page must be named `listPage` in order for this example to work:

```

<!-- page named listPage -->
<apex:page standardController="Idea" extensions="MyIdeaListExtension"
recordSetVar="ideaSetVar">
    <apex:pageBlock >
        <ideas:listOutputLink sort="recent" page="listPage">Recent Ideas
        </ideas:listOutputLink>
        |
        <ideas:listOutputLink sort="top" page="listPage">Top Ideas
        </ideas:listOutputLink>
        |
        <ideas:listOutputLink sort="popular" page="listPage">Popular Ideas
        </ideas:listOutputLink>
        |
        <ideas:listOutputLink sort="comments" page="listPage">Recent Comments
        </ideas:listOutputLink>
    </apex:pageBlock>
    <apex:pageBlock >
        <apex:dataList value="{!modifiedIdeas}" var="ideadata">
            <ideas:detailoutputlink ideaId="{!ideadata.id}" page="viewPage">
                {!ideadata.title}</ideas:detailoutputlink>
            </apex:dataList>
        </apex:pageBlock>
    </apex:page>

```

In the previous example, the `<ideas:detailOutputLink>` component links to the following Visualforce markup that displays the detail page for a specific idea. This page must be named `viewPage`.

```
<!-- page named viewPage -->
<apex:page standardController="Idea">
    <apex:pageBlock title="Idea Section">
        <ideas:detailOutputLink page="viewPage" ideaId="{!idea.id}">{!idea.title}
        </ideas:detailOutputLink>
        <br/><br/>
        <apex:outputText>{!idea.body}</apex:outputText>
    </apex:pageBlock>
</apex:page>
```

SEE ALSO:

[StandardSetController Class](#)

IdeaStandardSetController Methods

The following are instance methods for `IdeaStandardSetController`.

IN THIS SECTION:

[getIdeaList\(\)](#)

Returns the list of read-only ideas in the current page set.

getIdeaList()

Returns the list of read-only ideas in the current page set.

Signature

```
public Idea[] getIdeaList()
```

Return Value

Type: `Idea[]`

Usage

You can use the `<ideas:listOutputLink>`, `<ideas:profileListOutputLink>`, and `<ideas:detailOutputLink>` components to display profile pages as well as idea list and detail pages (see the examples below). The following is a list of properties returned by this method:

- `Body`
- `Categories`
- `Category`
- `CreatedBy.CommunityNickname`
- `CreatedBy.Id`
- `CreatedDate`

- `Id`
- `LastCommentDate`
- `LastComment.Id`
- `LastComment.CommentBody`
- `LastComment.CreatedBy.CommunityNickname`
- `LastComment.CreatedBy.Id`
- `NumComments`
- `Status`
- `Title`
- `VoteTotal`

KnowledgeArticleVersionStandardController Class

`KnowledgeArticleVersionStandardController` objects offer article-specific functionality in addition to what is provided by the `StandardController`.

Namespace

[ApexPages](#)

Usage

In addition to the method listed above, the `KnowledgeArticleVersionStandardController` class inherits all the methods associated with `StandardController`.



Note: Though inherited, the `edit`, `delete`, and `save` methods don't serve a function when used with the `KnowledgeArticleVersionStandardController` class.

Example

The following example shows how a `KnowledgeArticleVersionStandardController` object can be used to create a custom extension controller. In this example, you create a class named `AgentContributionArticleController` that allows customer-support agents to see pre-populated fields on the draft articles they create while closing cases.

Prerequisites:

1. Create an article type called *FAQ*. For instructions, see “Create Article Types” in the Salesforce online help.
2. Create a text custom field called *Details*. For instructions, see “Add Custom Fields to Article Types” in the Salesforce online help.
3. Create a category group called *Geography* and assign it to a category called *USA*. For instructions, see “Create and Modify Category Groups” and “Add Data Categories to Category Groups” in the Salesforce online help.
4. Create a category group called *Topics* and assign it a category called *Maintenance*.

```
/** Custom extension controller for the simplified article edit page that
    appears when an article is created on the close-case page.
 */
public class AgentContributionArticleController {
    // The constructor must take a ApexPages.KnowledgeArticleVersionStandardController as
    an argument
```

```

public AgentContributionArticleController(
    ApexPages.KnowledgeArticleVersionStandardController ctl) {
    // This is the SObject for the new article.
    //It can optionally be cast to the proper article type.
    // For example, FAQ__kav article = (FAQ__kav) ctl.getRecord();
    SObject article = ctl.getRecord();
    // This returns the ID of the case that was closed.
    String sourceId = ctl.getSourceId();
    Case c = [SELECT Subject, Description FROM Case WHERE Id=:sourceId];

    // This overrides the default behavior of pre-filling the
    // title of the article with the subject of the closed case.
    article.put('title', 'From Case: '+c.subject);
    article.put('details__c',c.description);

    // Only one category per category group can be specified.
    ctl.selectDataCategory('Geography','USA');
    ctl.selectDataCategory('Topics','Maintenance');
}

}

/** Test class for the custom extension controller.
 */
@isTest
private class AgentContributionArticleControllerTest {
    static testMethod void testAgentContributionArticleController() {
        String caseSubject = 'my test';
        String caseDesc = 'my test description';

        Case c = new Case();
        c.subject= caseSubject;
        c.description = caseDesc;
        insert c;
        String caseId = c.id;
        System.debug('Created Case: ' + caseId);

        ApexPages.currentPage().getParameters().put('sourceId', caseId);
        ApexPages.currentPage().getParameters().put('sfdc.override', '1');

        ApexPages.KnowledgeArticleVersionStandardController ctl =
            new ApexPages.KnowledgeArticleVersionStandardController(new FAQ__kav());

        new AgentContributionArticleController(ctl);

        System.assertEquals(caseId, ctl.getSourceId());
        System.assertEquals('From Case: '+caseSubject, ctl.getRecord().get('title'));
        System.assertEquals(caseDesc, ctl.getRecord().get('details__c'));
    }
}

```

If you created the custom extension controller for the purpose described in the previous example (that is, to modify submitted-via-case articles), complete the following steps after creating the class:

1. Log into your Salesforce organization and from Setup, enter *Knowledge Settings* in the Quick Find box, then select **Knowledge Settings**.

2. Click **Edit**.
3. Assign the class to the `Use Apex customization` field. This associates the article type specified in the new class with the article type assigned to closed cases.
4. Click **Save**.

IN THIS SECTION:

[KnowledgeArticleVersionStandardController Constructors](#)

[KnowledgeArticleVersionStandardController Methods](#)

SEE ALSO:

[StandardController Class](#)

KnowledgeArticleVersionStandardController Constructors

The following are constructors for `KnowledgeArticleVersionStandardController`.

IN THIS SECTION:

[KnowledgeArticleVersionStandardController\(article\)](#)

Creates a new instance of the `ApexPages.KnowledgeArticleVersionStandardController` class using the specified knowledge article.

KnowledgeArticleVersionStandardController(article)

Creates a new instance of the `ApexPages.KnowledgeArticleVersionStandardController` class using the specified knowledge article.

Signature

```
public KnowledgeArticleVersionStandardController(SObject article)
```

Parameters

article

Type: `SObject`

The knowledge article, such as `FAQ_kav`.

KnowledgeArticleVersionStandardController Methods

The following are instance methods for `KnowledgeArticleVersionStandardController`.

IN THIS SECTION:

[getSourceId\(\)](#)

Returns the ID for the source object record when creating a new article from another object.