**Building a Custom Controller**

A custom controller is an Apex class that uses the default, no-argument constructor for the outer, top-level class. You cannot create a custom controller constructor that includes parameters.

To create a custom controller:

1. From Setup, enter Apex Classes in the Quick Find box, then select **Apex Classes**.
2. Click **New**.
3. Click **Version Settings** to specify the version of Apex and the API used with this class. If your organization has installed managed packages from the AppExchange, you can also specify which version of each managed package to use with this class. Use the default values for all versions. This associates the class with the most recent version of Apex and the API, as well as each managed package. You can specify an older version of a managed package if you want to access components or functionality that differs from the most recent package version. You can specify an older version of Apex and the API to maintain specific behavior.
4. In the class editor, enter the Apex code for the class. A single class can be up to 1 million characters in length, not including comments, test methods, or classes defined using @isTest.
5. Click **Save** to save your changes and return to the class detail screen, or click **Quick Save** to save your changes and continue editing your class. Your Apex class must compile correctly before you can save your class.

The following class is a simple example of a custom controller:

|  |  |  |
| --- | --- | --- |
| 01 | public class MyController { | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | private final Account account; | |
| 04 |  |

|  |  |
| --- | --- |
| 05 | public MyController() { |
| 06 | account = [SELECT Id, Name, Site FROM Account | |

|  |  |  |
| --- | --- | --- |
| 07 | WHERE Id = :ApexPages.currentPage().getParameters().get('id')]; | |
| 08 | } |

|  |  |
| --- | --- |
| 09 |  |
| 10 | public Account getAccount() { | |

|  |  |  |
| --- | --- | --- |
| 11 | return account; | |
| 12 | } |

|  |  |
| --- | --- |
| 13 |  |
| 14 | public PageReference save() { | |

|  |  |  |
| --- | --- | --- |
| 15 | update account; | |
| 16 | return null; |

|  |  |  |
| --- | --- | --- |
| 17 | } | |
| 18 | } |

The following Visualforce markup shows how the custom controller above can be used in a page:

|  |  |  |
| --- | --- | --- |
| 1 | <apex:page controller="myController" tabStyle="Account"> | |
| 2 | <apex:form> |

|  |  |
| --- | --- |
| 3 | <apex:pageBlock title="Congratulations {!$User.FirstName}"> |
| 4 | You belong to Account Name: <apex:inputField value="{!account.name}"/> | |

|  |  |  |
| --- | --- | --- |
| 5 | <apex:commandButton action="{!save}" value="save"/> | |
| 6 | </apex:pageBlock> |

|  |  |  |
| --- | --- | --- |
| 7 | </apex:form> | |
| 8 | </apex:page> |

The custom controller is associated with the page because of the controller attribute of the <apex:page> component.

As with standard controllers and controller extensions, custom controller methods can be referenced with {! } notation in the associated page markup. In the example above, the getAccount method is referenced by the <apex:inputField> tag's value attribute, while the <apex:commandButton> tag references the save method with its action attribute.

Note

Like other Apex classes, all custom controllers run in system mode. Consequently, the current user's credentials are not used to execute controller logic, and the user's permissions and field-level security do not apply.

You can choose whether a custom controller respects a user's organization-wide defaults, role hierarchy, and sharing rules by using the with sharing keywords in the class definition. For information, see “Using the with sharing or without sharing Keywords” in the [Apex Developer Guide](https://developer.salesforce.com/docs/atlas.en-us.214.0.apexcode.meta/apexcode/apex_classes_keywords_sharing.htm).

A custom controller can also be used to create new records. For example:

|  |  |  |
| --- | --- | --- |
| 01 | public class NewAndExistingController { | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | public Account account { get; private set; } | |
| 04 |  |

|  |  |
| --- | --- |
| 05 | public NewAndExistingController() { |
| 06 | Id id = ApexPages.currentPage().getParameters().get('id'); | |

|  |  |
| --- | --- |
| 07 | account = (id == null) ? new Account() : |
| 08 | [SELECT Name, Phone, Industry FROM Account WHERE Id = :id]; | |

|  |  |  |
| --- | --- | --- |
| 09 | } | |
| 10 |  |

|  |  |  |
| --- | --- | --- |
| 11 | public PageReference save() { | |
| 12 | try { |

|  |  |
| --- | --- |
| 13 | upsert(account); |
| 14 | } catch(System.DMLException e) { | |

|  |  |  |
| --- | --- | --- |
| 15 | ApexPages.addMessages(e); | |
| 16 | return null; |

|  |  |
| --- | --- |
| 17 | } |
| 18 | //  After successful Save, navigate to the default view page | |

|  |  |  |
| --- | --- | --- |
| 19 | PageReference redirectSuccess = new ApexPages.StandardController(Account).view(); | |
| 20 | return (redirectSuccess); |

|  |  |  |
| --- | --- | --- |
| 21 | } | |
| 22 | } |

The following Visualforce markup shows how the custom controller above can be used in a page:

[view source](https://developer.salesforce.com/docs/#viewSource)

[print](https://developer.salesforce.com/docs/#printSource)[?](https://developer.salesforce.com/docs/#about)

|  |  |  |
| --- | --- | --- |
| 01 | <apex:page controller="NewAndExistingController" tabstyle="Account"> | |
| 02 | <apex:form> |

|  |  |  |
| --- | --- | --- |
| 03 | <apex:pageBlock mode="edit"> | |
| 04 | <apex:pageMessages/> |

|  |  |
| --- | --- |
| 05 | <apex:pageBlockSection> |
| 06 | <apex:inputField value="{!Account.name}"/> | |

|  |  |
| --- | --- |
| 07 | <apex:inputField value="{!Account.phone}"/> |
| 08 | <apex:inputField value="{!Account.industry}"/> | |

|  |  |
| --- | --- |
| 09 | </apex:pageBlockSection> |
| 10 | <apex:pageBlockButtons location="bottom"> | |

|  |  |  |
| --- | --- | --- |
| 11 | <apex:commandButton value="Save" action="{!save}"/> | |
| 12 | </apex:pageBlockButtons> |

|  |  |  |
| --- | --- | --- |
| 13 | </apex:pageBlock> | |
| 14 | </apex:form> |

|  |  |
| --- | --- |
| 15 | </apex:page> |

# Building a Controller Extension

A controller extension is any Apex class containing a constructor that takes a single argument of type ApexPages.StandardController or CustomControllerName, where CustomControllerName is the name of a custom controller you want to extend.

The following class is a simple example of a controller extension:

|  |  |  |
| --- | --- | --- |
| 01 | public class myControllerExtension { | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | private final Account acct; | |
| 04 |  |

|  |  |
| --- | --- |
| 05 | // The extension constructor initializes the private member |
| 06 | // variable acct by using the getRecord method from the standard | |

|  |  |
| --- | --- |
| 07 | // controller. |
| 08 | public myControllerExtension(ApexPages.StandardController stdController) { | |

|  |  |  |
| --- | --- | --- |
| 09 | this.acct = (Account)stdController.getRecord(); | |
| 10 | } |

|  |  |
| --- | --- |
| 11 |  |
| 12 | public String getGreeting() { | |

|  |  |  |
| --- | --- | --- |
| 13 | return 'Hello ' + acct.name + ' (' + acct.id + ')'; | |
| 14 | } |

|  |  |
| --- | --- |
| 15 | } |

The following Visualforce markup shows how the controller extension from above can be used in a page:

|  |  |  |
| --- | --- | --- |
| 1 | <apex:page standardController="Account" extensions="myControllerExtension"> | |
| 2 | {!greeting} <p/> |

|  |  |
| --- | --- |
| 3 | <apex:form> |
| 4 | <apex:inputField value="{!account.name}"/> <p/> | |

|  |  |  |
| --- | --- | --- |
| 5 | <apex:commandButton value="Save" action="{!save}"/> | |
| 6 | </apex:form> |

|  |  |
| --- | --- |
| 7 | </apex:page> |

The extension is associated with the page using the extensions attribute of the <apex:page> component.

As with all controller methods, controller extension methods can be referenced with {! } notation in page markup. In the example above, the {!greeting} expression at the top of the page references the controller extension's getGreetingmethod.

Because this extension works in conjunction with the Account standard controller, the standard controller methods are also available. For example, the value attribute in the <apex:inputField> tag retrieves the name of the account using standard controller functionality. Likewise, the <apex:commandButton> tag references the standard account save method with its action attribute.

Multiple controller extensions can be defined for a single page through a comma-separated list. This allows for overrides of methods with the same name. For example, if the following page exists:

|  |  |
| --- | --- |
| 1 | <apex:page standardController="Account" |
| 2 | extensions="ExtOne,ExtTwo" showHeader="false"> | |

|  |  |  |
| --- | --- | --- |
| 3 | <apex:outputText value="{!foo}" /> | |
| 4 | </apex:page> |

with the following extensions:

|  |  |
| --- | --- |
| 1 | public class ExtOne { |
| 2 | public ExtOne(ApexPages.StandardController acon) { } | |

|  |  |
| --- | --- |
| 3 |  |
| 4 | public String getFoo() { | |

|  |  |  |
| --- | --- | --- |
| 5 | return 'foo-One'; | |
| 6 | } |

|  |  |
| --- | --- |
| 7 | } |
| 1 | public class ExtTwo { | |

|  |  |  |
| --- | --- | --- |
| 2 | public ExtTwo(ApexPages.StandardController acon) { } | |
| 3 |  |

|  |  |  |
| --- | --- | --- |
| 4 | public String getFoo() { | |
| 5 | return 'foo-Two'; |

|  |  |  |
| --- | --- | --- |
| 6 | } | |
| 7 | } |