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.

Namespace

ApexPages

Usage

The `StandardSetController` class also contains a *prototype object*. This is a single `sObject` contained within the Visualforce StandardSetController class. If the prototype object's fields are set, those values are used during the save action, meaning that the values are applied to every record in the set controller's collection. This is useful for writing pages that perform mass updates (applying identical changes to fields within a collection of objects).

Note

Fields that are required in other Salesforce objects will keep the same requiredness when used by the prototype object.

Instantiation

You can instantiate a StandardSetController in either of the following ways:

- From a list of sObjects:

```
List<account> accountList = [SELECT Name FROM Account LIMIT 20];
ApexPages.StandardSetController ssc = new ApexPages.StandardSetController(accountLi
```

- From a query locator:

```
ApexPages.StandardSetController ssc =
new ApexPages.StandardSetController(Database.getQueryLocator([SELECT Name,CloseDate
```

Note

The maximum record limit for StandardSetController is 10,000 records. Instantiating StandardSetController using a query locator returning more than 10,000 records causes a LimitException to be thrown. However, instantiating StandardSetController with a list of more than 10,000 records doesn't throw an exception, and instead truncates the records to the limit.

```
public class opportunityList2Con {
    // ApexPages.StandardSetController must be instantiated
    // for standard list controllers
    public ApexPages.StandardSetController setCon {
        get {
            if(setCon == null) {
                setCon = new ApexPages.StandardSetController(Database.getQueryLocator(
                    [SELECT Name, CloseDate FROM Opportunity]));
            }
            return setCon;
        }
        set;
    }

    // Initialize setCon and return a list of records
    public List<Opportunity> getOpportunities() {
        return (List<Opportunity>) setCon.getRecords();
    }
}
```

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

```
<apex:page controller="opportunityList2Con">
    <apex:pageBlock>
        <apex:pageBlockTable value="{!opportunities}" var="o">
            <apex:column value="{!o.Name}"/>
            <apex:column value="{!o.CloseDate}"/>
        </apex:pageBlockTable>
    </apex:pageBlock>
</apex:page>
```

- StandardSetController Constructors
- StandardSetController Methods

StandardSetController Constructors

The following are constructors for `StandardSetController`.

- **StandardSetController(queryLocator)**
  Creates an instance of the `ApexPages.StandardSetController` class for the list of objects returned by the query locator.
- **StandardSetController(controllerSObjects)**
  Creates an instance of the `ApexPages.StandardSetController` class for the specified list of standard or custom objects.

StandardSetController(queryLocator)

Creates an instance of the `ApexPages.StandardSetController` class for the list of objects returned by the query locator.

Signature

```
public StandardSetController(Database.QueryLocator queryLocator)
```

Parameters

queryLocator

Type: Database.QueryLocator

A query locator representing a list of sObjects.

Signature

```
public StandardSetController(List<sObject> controllerSObjects)
```

Parameters

controllerSObjects

Type: List<sObject>

A List of standard or custom objects.

Example

```
List<account> accountList = [SELECT Name FROM Account LIMIT 20];
ApexPages.StandardSetController ssc = new ApexPages.StandardSetController(accountList);
```

StandardSetController Methods

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

- **cancel()**
  Returns the PageReference of the original page, if known, or the home page.
- **first()**
  Changes the set of records that the controller returns to the first page of records.
- **getCompleteResult()**
  Indicates whether there are more records in the set than the maximum record limit. If this is false, there are more records than you can process using the list controller. The maximum record limit is 10,000 records.
- **getFilterId()**
  Returns the ID of the filter that is currently in context.
- **getHasNext()**
  Indicates whether there are more records after the current page set.
- **getHasPrevious()**
  Indicates whether there are more records before the current page set.
- **getListViewOptions()**
  Returns a list of the listviews available to the current user.
- **getPageNumber()**
  Returns the page number of the current page set. Note that the first page returns 1.
- **getPageSize()**
  Returns the number of records included in each page set.
- **getRecord()**
  Returns the sObject that represents the changes to the selected records. This retrieves the prototype object contained within the class, and is used for performing mass updates.
- **getRecords()**
  Returns the list of sObjects in the current page set. This list is immutable, i.e. you can't call `clear`() on it.
- **getResultSize()**
  Returns the number of records in the set.
- **getSelected()**
  Returns the list of sObjects that have been selected.
- **last()**
  Changes the set of records that the controller returns to the last page of records.
- **next()**
  Changes the set of records that the controller returns to the next page of records.

operation is finished, it returns a PageReference to the original page, if known, or the home page.

- **setFilterID(filterId)**
  Sets the filter ID of the controller.
- **setpageNumber(pageNumber)**
  Sets the page number.
- **setPageSize(pageSize)**
  Sets the number of records in each page set.
- **setSelected(selectedRecords)**
  Set the selected records to the records specified in the *selectedRecords* argument.

cancel()

Returns the PageReference of the original page, if known, or the home page.

Signature

```
public System.PageReference cancel()
```

Return Value

Type: System.PageReference

See Also

- *Visualforce Developer Guide*: Standard List Controller Actions

first()

Changes the set of records that the controller returns to the first page of records.

Signature

```
public Void first()
```

Return Value

Type: Void

See Also

- *Visualforce Developer Guide*: Standard List Controller Actions

getCompleteResult()

Indicates whether there are more records in the set than the maximum record limit. If this is false, there are more records than you can process using the list controller. The maximum record limit is 10,000 records.

Signature

```
public Boolean getCompleteResult()
```

Return Value

Type: Boolean

Note

The `getFilterID()` method doesn't support list views without filter IDs, such as the Recently Viewed list view. In these cases, the method returns the first filter ID of the object's available list views. If called within an `<apex:enhancedList>` component, the method returns the filter ID of the last used list view.

Signature

```
public String getFilterId()
```

Return Value

Type: String

See Also

- *Visualforce Developer Guide*: Standard List Controller Actions
- *Visualforce Developer Guide*: List Views with Standard List Controllers

getHasNext()

Indicates whether there are more records after the current page set.

Signature

```
public Boolean getHasNext()
```

Return Value

Type: Boolean

getHasPrevious()

Indicates whether there are more records before the current page set.

Signature

```
public Boolean getHasPrevious()
```

Return Value

Type: Boolean

getListViewOptions()

Returns a list of the listviews available to the current user.

Signature

```
public System.SelectOption getListViewOptions()
```

Return Value

Type: System.SelectOption[]

See Also

- *Visualforce Developer Guide*: Standard List Controller Actions

Returns the page number of the current page set. Note that the first page returns 1.

Signature

```
public Integer getPageNumber()
```

Return Value

Type: Integer

getPageSize()

Returns the number of records included in each page set.

Signature

```
public Integer getPageSize()
```

Return Value

Type: Integer

getRecord()

Returns the sObject that represents the changes to the selected records. This retrieves the prototype object contained within the class, and is used for performing mass updates.

Signature

```
public sObject getRecord()
```

Return Value

Type: sObject

See Also

- *Visualforce Developer Guide*: Building a Custom List Controller

getRecords()

Returns the list of sObjects in the current page set. This list is immutable, i.e. you can't call `clear()` on it.

Signature

```
public sObject[] getRecords()
```

Return Value

Type: sObject[]

See Also

- *Visualforce Developer Guide*: Building a Custom List Controller

getResultSize()

Return Value

Type: Integer

getSelected()

Returns the list of sObjects that have been selected.

Signature

```
public sObject[] getSelected()
```

Return Value

Type: sObject[]

last()

Changes the set of records that the controller returns to the last page of records.

Signature

```
public Void last()
```

Return Value

Type: Void

See Also

- *Visualforce Developer Guide*: Standard List Controller Actions

next()

Changes the set of records that the controller returns to the next page of records.

Signature

```
public Void next()
```

Return Value

Type: Void

See Also

- *Visualforce Developer Guide*: Standard List Controller Actions

previous()

Changes the set of records that the controller returns to the previous page of records.

Signature

```
public Void previous()
```

Return Value

save()

Inserts new records or updates existing records that have been changed. After this operation is finished, it returns a PageReference to the original page, if known, or the home page.

Signature

```
public System.PageReference save()
```

Return Value

Type: System.PageReference

See Also

- *Visualforce Developer Guide*: Standard List Controller Actions

setFilterID(filterId)

Sets the filter ID of the controller.

Signature

```
public Void setFilterID(String filterId)
```

Parameters

filterId
Type: String

Return Value

Type: Void

setpageNumber(pageNumber)

Sets the page number.

Signature

```
public Void setpageNumber(Integer pageNumber)
```

Parameters

pageNumber
Type: Integer

Return Value

Type: Void

setPageSize(pageSize)

Sets the number of records in each page set.

Signature

```
public Void setPageSize(Integer pageSize)
```

Return Value

Type: Void

setSelected(selectedRecords)

Set the selected records to the records specified in the *selectedRecords* argument.

Signature

`public Void setSelected(sObject[] selectedRecords)`

Parameters

selectedRecords

Type: sObject[]

Return Value

Type: Void

Usage

Use the `setSelected()` method in your Apex controller or controller extension to manually set the records displayed on a Visualforce page. The `setSelected()` method overwrites any previously selected records with the records specified in the *selectedRecords* argument.

Example

`AccountNamePage` shows a table of account names. `MyControllerExtension`'s constructor contains a SOQL query that returns a list of accounts. This list is passed into `setSelected()` so that the account records in the list are selected and displayed in the table.

```
<!-- AccountNamePage.page -->
<apex:page standardController="Account" recordSetVar="accounts" extensions="MyControllerEx
    <apex:pageBlock>
        <apex:pageBlockTable value="{!accounts}" var="acc">
            <apex:column value="{!acc.name}"/>
        </apex:pageBlockTable>
    </apex:pageBlock>
</apex:page>


// MyControllerExtension.cls
public with sharing class MyControllerExtension {
    private ApexPages.StandardSetController setController;

    public MyControllerExtension(ApexPages.StandardSetController setController) {
        this.setController = setController;

        Account [] records = [SELECT Id, Name FROM Account LIMIT 30];
        setController.setSelected(records);
    }
}
```

See Also

- *Visualforce Developer Guide*: Accessing Data with List Controllers

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

Privacy Information Terms of Service Legal Use of Cookies Trust Cookie Preferences

Your Privacy Choices Responsible Disclosure Contact