.webpart file – helps to define title and description of webpart
Elements.xml – Define the category to which webpart should be added
Webpart code behind
User control code behind

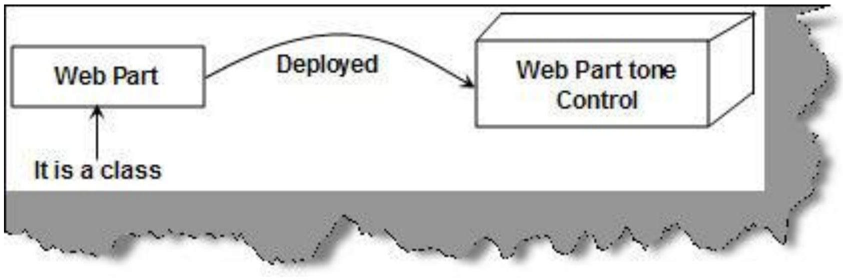
What are WebParts and in what ways do they vary in the SharePoint environment?

It helps to build reusable components which can be customized and personalized according to the business user. We can either make our own WebPart or reuse existing ones from SharePoint.

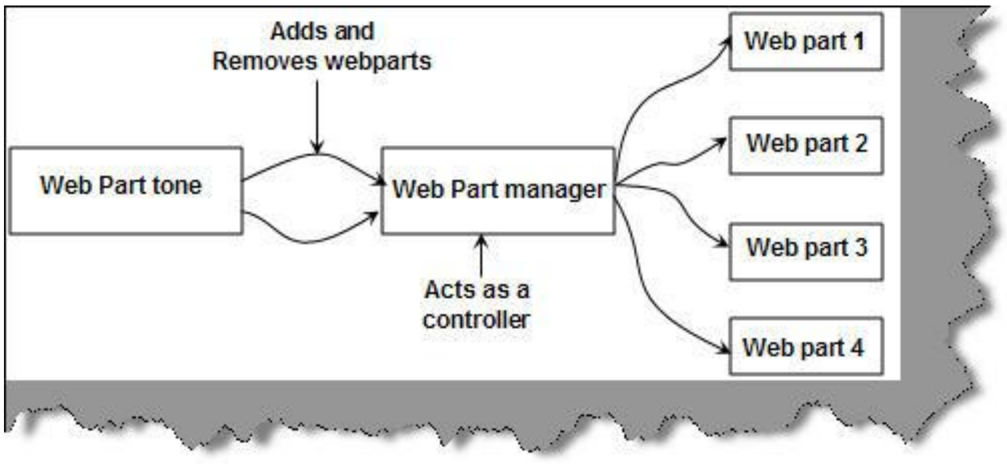
Following WebParts are available with WSS:

- DataView Web Part: Displays data with rich design support through the Microsoft SharePoint Designer.
- ListView Web Part: Helps us display list content for any list in the SharePoint site.
- Image Web Part: Helps us display image files.
- Content Editor Web Part: Use this to display static HTML content using a WYSIWYG editor or link to a text file.
- Members Web Part: Helps us display members of the site.
- PageViewer Web Part: Displays web pages in an iFrame.

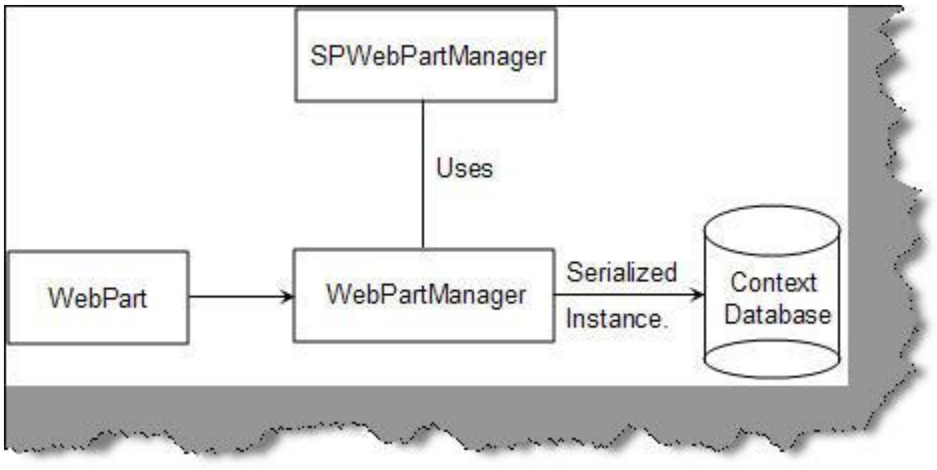
WebPart is nothing but a simple class which inherits from `System.Web.UI.WebControls.WebParts`. In other words you can say WebPart is kind of a WebControl which can be deployed in a `WebPartZoneControl`.



The WebPart manager control is like a controller which maintains instances of WebParts. It adds WebParts to the WebPartZone when the ASP.NET pages get initialized.



Now let's try to see how 'WebParts' work in a SharePoint environment. `WebPartManager` uses the`SPWebPartManager`. `WebPartZone` interacts with `WebPartManager` which in-turn stores a serialized instance of the WebPart in the content database. In other words, the code is stored in the content database for customization.



The big difference is that the 'WebPart' code is stored in a content database. So some of the differences as compared to normal ASP.NET is that, the code is deployed in a content database and is parsed using 'Safe Mode Parser'.

What are the different life cycle events that WebPart goes through?

A 'WebPart' control goes through various events and has a typical life cycle.

- `OnInit`: This is the initialization event and is the first event to occur.
- `OnLoad`: The load event.
- `CreateChildControls`: When child controls are added to a composite control, this event is fired.
- `EnsureChildControls`: This event makes sure that `CreateChildControls` fires.

- `OnPreRender`: This fires just before the render event.
- `Page.PreRenderComplete`: When all controls have executed, the `OnPreRender` event fires.
- `Render`: Render the full control.
- `RenderContents`: Renders the contents of the control only.

What's the difference between WebParts in WSS 2.0 and 3.0?

First let's talk about a bit of history. 'WebParts' was born when WSS 2.0 was first introduced. In the ASP.NET 2.0 framework, a new version of 'WebPart' was built which could run without WSS. WSS 3.0 uses the ASP.NET 2.0 'WebPart' framework.

Here are some quick points:

- If you are doing new development using WSS 3.0, you should use the ASP.NET 2.0 'WebPart' framework.
- If you want backward compatibility, you use the `Microsoft.SharePoint.WebPartPages.WebPart` class.

So the decision will be more based on what level of backward compatibility you are looking for.

The table here shows the new WebPart classes as compared to the SharePoint backward compatible class:

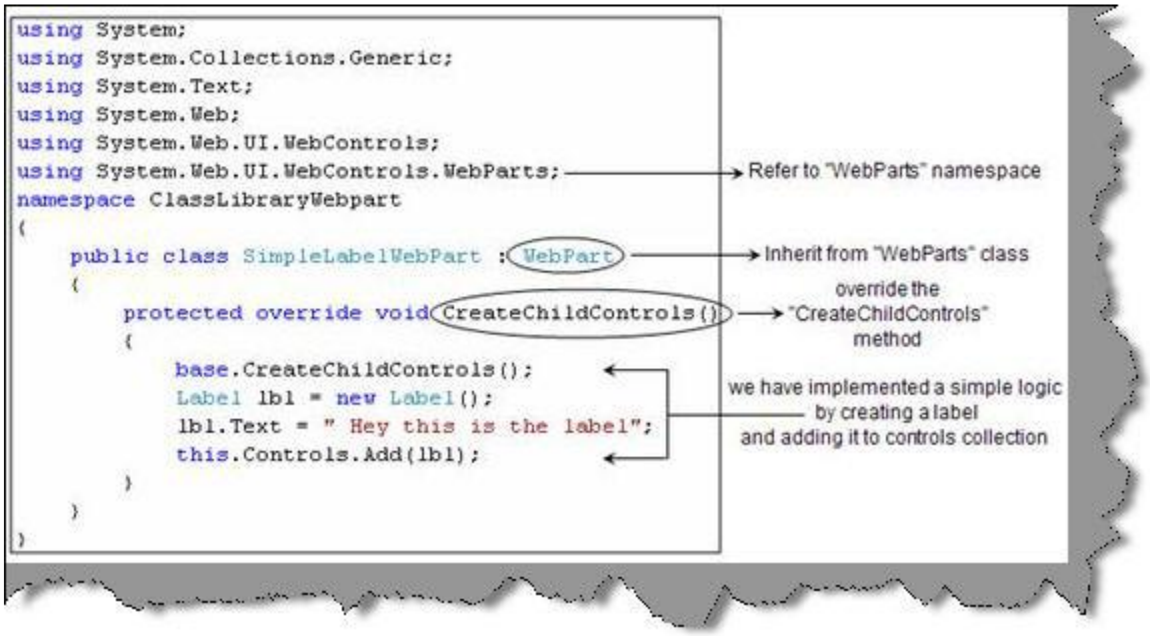
ASP.NET Web Parts classes	SharePoint Backward Compatibility classes
`WebBrowsableAttribute`	`BrowsableAttribute`
`WebDisplayName`	`FriendlyName`
`WebDescription`	`Description`
`Personalizable`	`WebPartStorage`
`PersonalizationScope`	`Storage`
`EditorPart`	`ToolPart`
`EditorPartCollection`	`ToolPart[]`
`CreateEditorParts()`	`GetToolParts()`
`RenderContents()`	`RenderWebPart()`
`SetPersonalizationDirty()`	`SaveProperties`

Can you explain the six steps we need to create a WebPart in SharePoint?

Now that we have understood the basics of WebPart, let's deploy a WebPart practically. There are three steps to deploy WebParts:

Step 1: Create the Webpart

The first step is to create the WebPart. Below is the code snippet of a WebPart. We need to reference `WebParts`, the custom webpart class should inherit from the `WebPart` class, and finally we need to override the `CreateChildControls` method with our custom implementation.



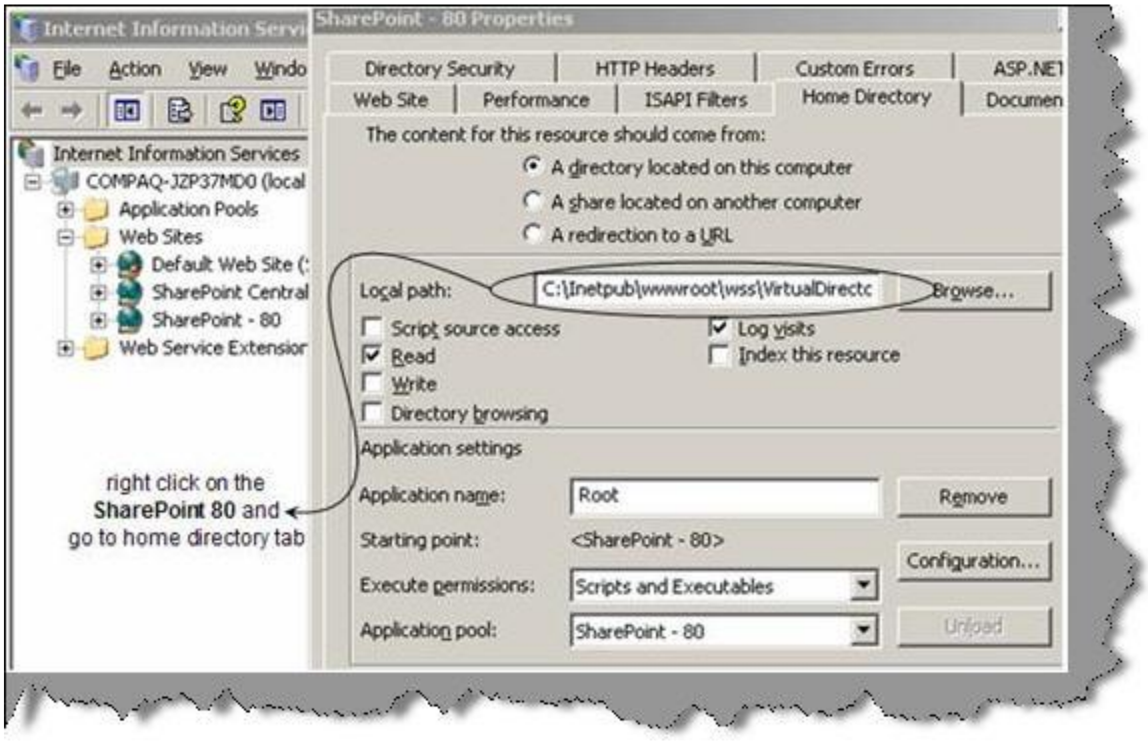
```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
namespace ClassLibraryWebpart
{
    public class SimpleLabelWebPart :WebPart
    {
        protected override void CreateChildControls()
        {
            base.CreateChildControls();
            Label lbl = new Label();
            lbl.Text = " Hey this is the label";
            this.Controls.Add(lbl);
        }
    }
}
```

Refer to "WebParts" namespace
Inherit from "WebParts" class
override the "CreateChildControls" method
we have implemented a simple logic by creating a label and adding it to controls collection

Step 2: Copy the compiled DLL to the virtual directory in the Bin folder

The second step is to copy the compiled DLL to the virtual directory of the SharePoint site.

You can get the virtual directory from IIS from the home directory tab in IIS. For instance, the current website is hosted on the SharePoint 80 virtual site which is actually located at '*C:\Inetpub\wwwroot\wss\VirtualDirectories\80*'.



The content for this resource should come from:
- (●) A directory located on this computer
- ( ) A share located on another computer
- ( ) A redirection to a URL

Local path: C:\Inetpub\wwwroot\wss\VirtualDirecto

- [ ] Script source access
- [✓] Read
- [ ] Write
- [ ] Directory browsing
- [✓] Log visits
- [ ] Index this resource

Application settings

Application name: Root

Starting point: <SharePoint - 80>

Execute permissions: Scripts and Executables

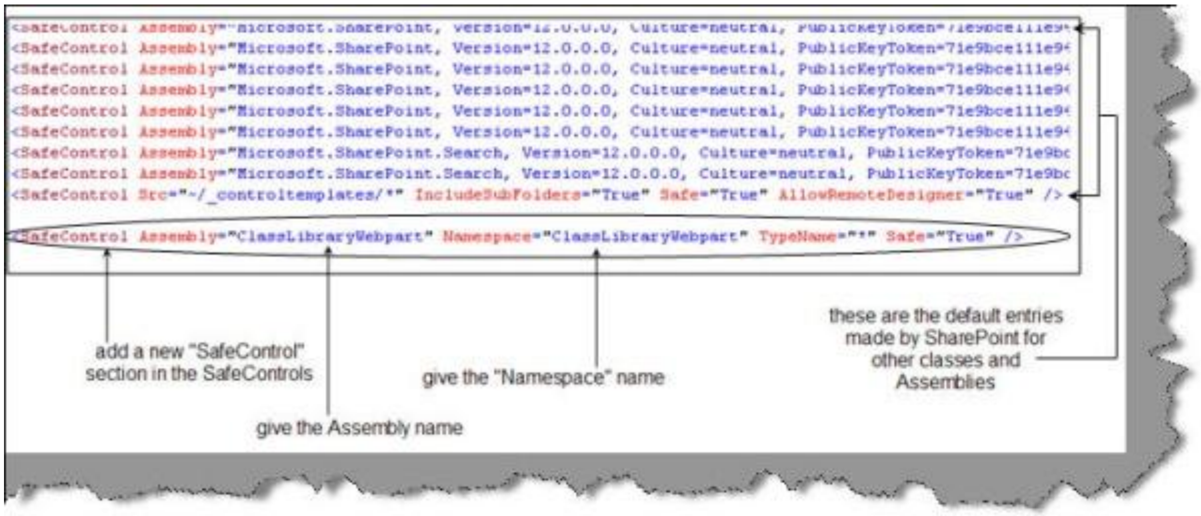
Application pool: SharePoint - 80

So now let's browse to the '*C: \Inetpub\wwwroot\wss\VirtualDirectories\80\bin*' directory and copy the DLL in the*bin* folder.

Step 3: Make the entry of the WebPart in the web.config file

Now that we have copied the DLL to the '*bin*' folder, we need to make a '`SafeControl`' entry in the '*Web.Config*' file of the virtual directory.

Let's browse to '*C:\Inetpub\wwwroot\wss\VirtualDirectories\80*', open the *web.config* file, and add the class library name in the '`SafeControls`' section of the *web.config* file. The code snippet below shows this in a visual manner.



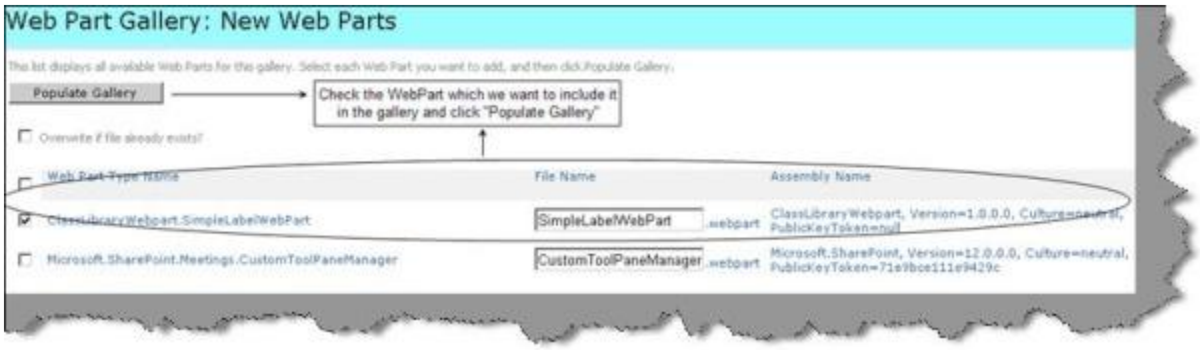
```
<SafeControl Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9
<SafeControl Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9
<SafeControl Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9
<SafeControl Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9
<SafeControl Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9
<SafeControl Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9
<SafeControl Assembly="Microsoft.SharePoint.Search, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bc
<SafeControl Assembly="Microsoft.SharePoint.Search, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bc
<SafeControl Src="~/_controltemplates/*" IncludeSubFolders="True" Safe="True" AllowRemoteDesigner="True" />

<SafeControl Assembly="ClassLibraryWebpart" Namespace="ClassLibraryWebpart" TypeName="*" Safe="True" />
```

add a new "SafeControl" section in the SafeControls
give the Assembly name
give the "Namespace" name
these are the default entries made by SharePoint for other classes and Assemblies

Step 4: Add it to the WebPart gallery

Now that we have added the WebPart in the '*web.config*' file, it is time to make it available in the web part gallery. Click on Site settings -> click Webparts -> click New button, you should see the figure shown below.

You can also see the WebPart which you recently added is the gallery. To make it available, click the check box and click Populate Gallery.



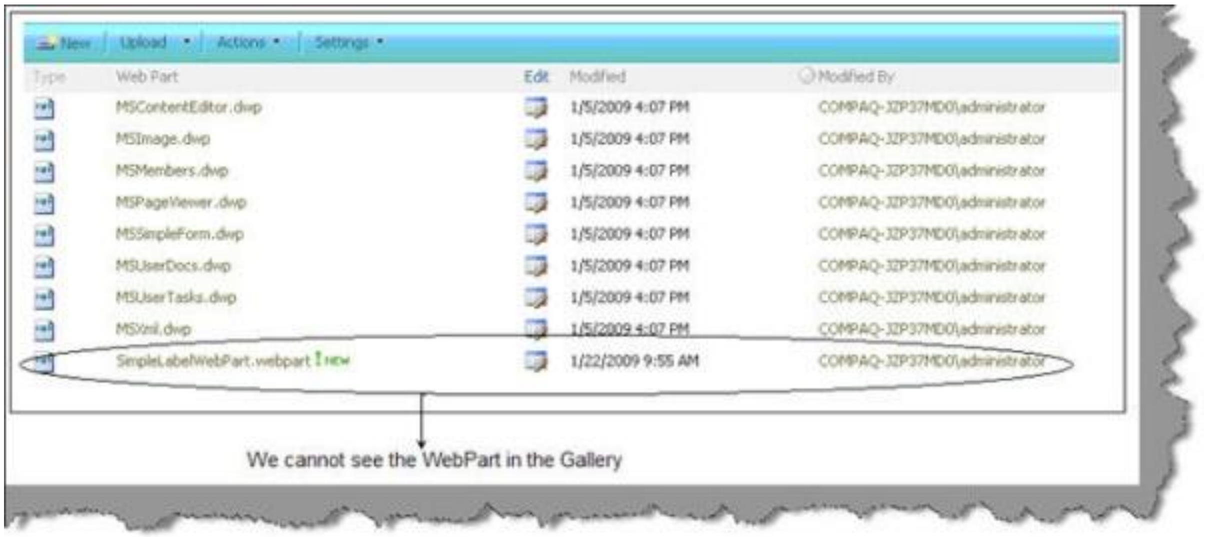
Web Part Gallery: New Web Parts

This list displays all available Web Parts for this gallery. Select each Web Part you want to add, and then click Populate Gallery.

Populate Gallery
Check the WebPart which we want to include it in the gallery and click "Populate Gallery"
Overwrite if file already exists?

	Web Part Type Name	File Name		Assembly Name
✓	ClassLibraryWebpart.SimpleLabelWebPart	SimpleLabelWebPart	.webpart	ClassLibraryWebpart, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
☐	Microsoft.SharePoint.Meetings.CustomToolPaneManager	CustomToolPaneManager	.webpart	Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c

You should now see your WebPart (for this context, it is SimpleLabelWebPart) in the WebPart gallery.

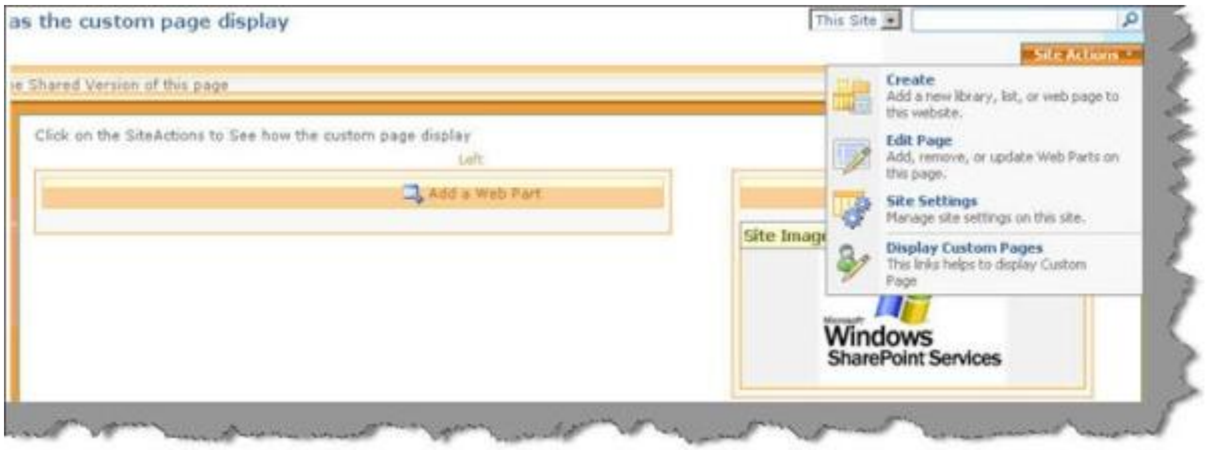


Type	Web Part	Edit	Modified	Modified By
	MSContentEditor.dwp		1/5/2009 4:07 PM	COMPAQ-32P37MD0\administrator
	MSImage.dwp		1/5/2009 4:07 PM	COMPAQ-32P37MD0\administrator
	MSMembers.dwp		1/5/2009 4:07 PM	COMPAQ-32P37MD0\administrator
	MSPageViewer.dwp		1/5/2009 4:07 PM	COMPAQ-32P37MD0\administrator
	MSSimpleForm.dwp		1/5/2009 4:07 PM	COMPAQ-32P37MD0\administrator
	MSUserDocs.dwp		1/5/2009 4:07 PM	COMPAQ-32P37MD0\administrator
	MSUserTasks.dwp		1/5/2009 4:07 PM	COMPAQ-32P37MD0\administrator
	MSXml.dwp		1/5/2009 4:07 PM	COMPAQ-32P37MD0\administrator
	SimpleLabelWebPart.webpart ! NEW		1/22/2009 9:55 AM	COMPAQ-32P37MD0\administrator

We cannot see the WebPart in the Gallery

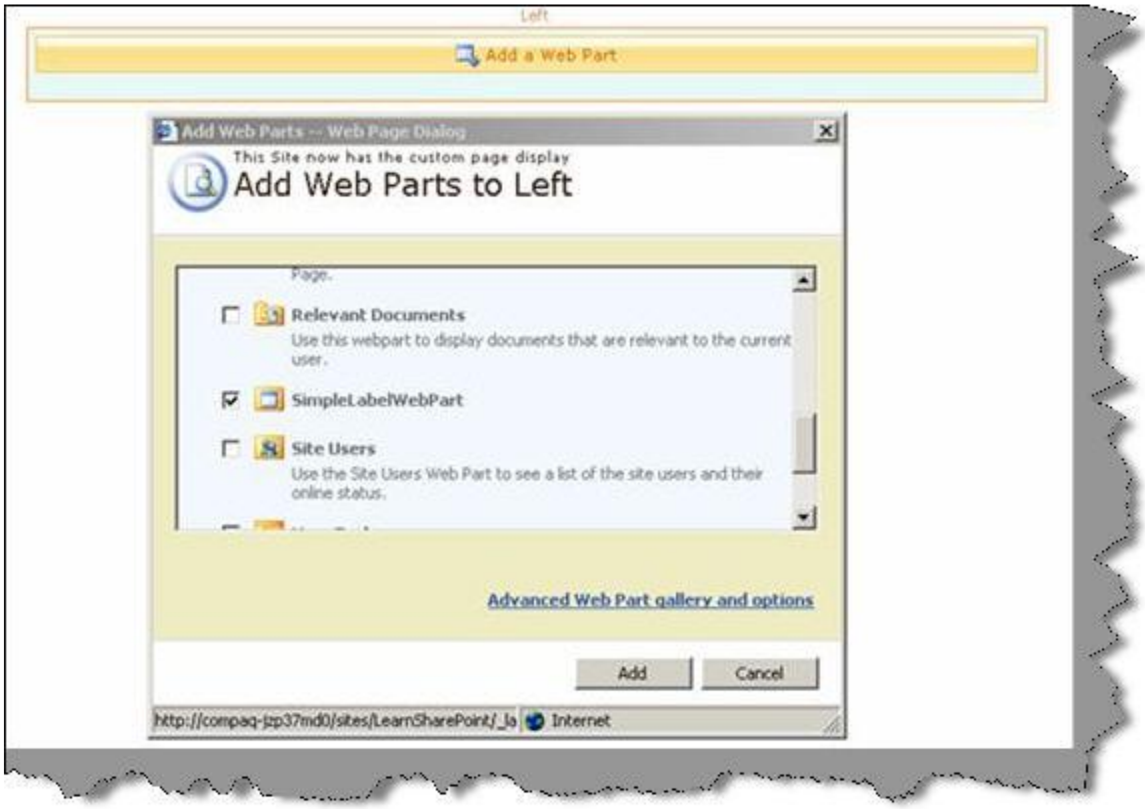
Step 5: Add the WebPart to site pages

Now that our WebPart is available in the gallery, it's time to add this WebPart to a SharePoint page. Go to the page where you want to add this WebPart, click on the site action menu, and click Edit page as shown in the below figure.

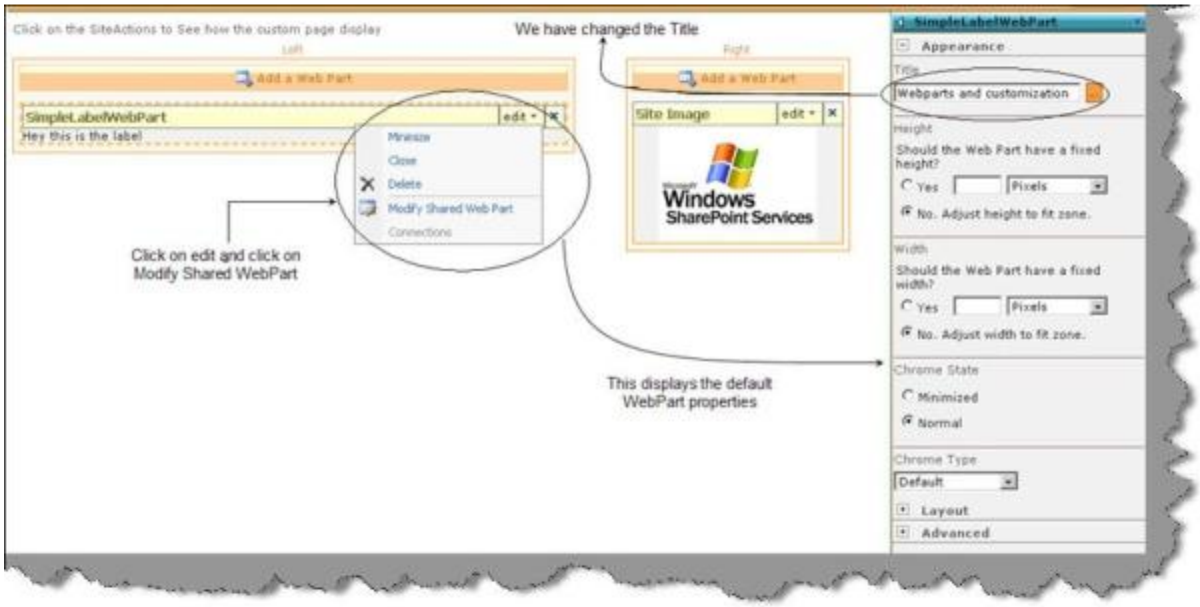


as the custom page display
This Site
Site Actions
e Shared Version of this page
Click on the SiteActions to See how the custom page display
Left
Add a Web Part
Site Imag
Create
Add a new library, list, or web page to this website.
Edit Page
Add, remove, or update Web Parts on this page.
Site Settings
Manage site settings on this site.
Display Custom Pages
This links helps to display Custom Page
Windows SharePoint Services

On the page, you should see an 'Add a WebPart' button. Click on it and it will populate with the existing Web Parts. Browse to your web part and click Add.

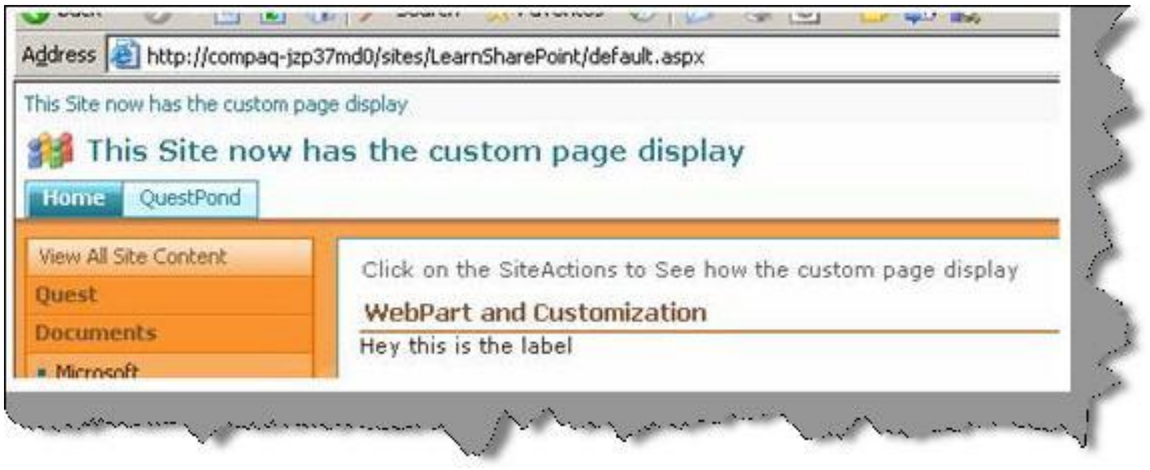

Left
Add a Web Part
Add Web Parts -- Web Page Dialog
This Site now has the custom page display
Add Web Parts to Left
Page.
Relevant Documents
Use this webpart to display documents that are relevant to the current user.
SimpleLabelWebPart
Site Users
Use the Site Users Web Part to see a list of the site users and their online status.
Advanced Web Part gallery and options
Add
Cancel
http://compaq-jzp37md0/sites/LearnSharePoint/_la
Internet

Once done, your WebPart is now added to the site page. You can edit the default WebPart properties by clicking on 'Modify Shared WebPart'. For simplicity's sake, we have changed the title.


Click on the SiteActions to See how the custom page display
We have changed the Title
Left
Add a Web Part
SimpleLabelWebPart
edit
Hey this is the label
Minimize
Close
Delete
Modify Shared Web Part
Connections
Click on edit and click on Modify Shared WebPart
Right
Add a Web Part
Site Image
edit
Windows SharePoint Services
This displays the default WebPart properties
SimpleLabelWebPart
Appearance
Title
Webparts and customization
Height
Should the Web Part have a fixed height?
Yes
Pixels
No. Adjust height to fit zone.
Width
Should the Web Part have a fixed width?
Yes
Pixels
No. Adjust width to fit zone.
Chrome State
Minimized
Normal
Chrome Type
Default
Layout
Advanced

Step 6: View the WebPart

You can see the changed title display when viewing the page normally. With a few lines of code, you can see how we have achieved a huge customization.

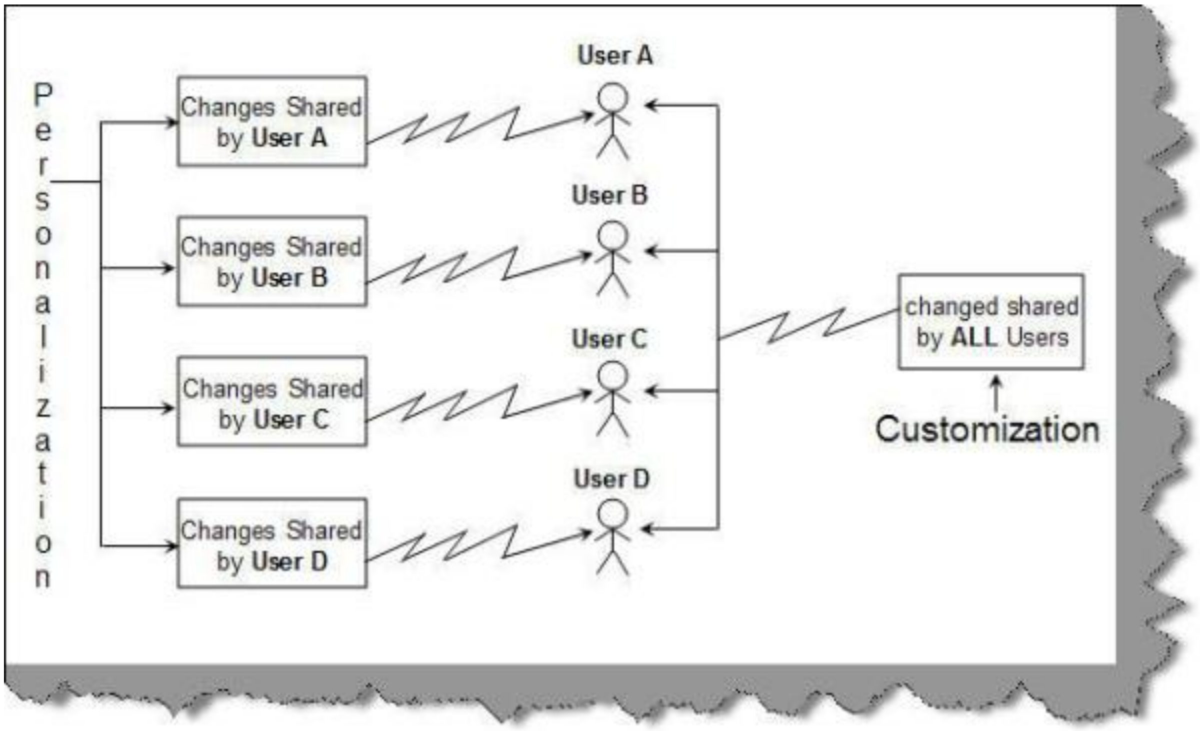

Address http://compaq-jzp37md0/sites/LearnSharePoint/default.aspx
This Site now has the custom page display
This Site now has the custom page display
Home
QuestPond
View All Site Content
Quest
Documents
Microsoft
Click on the SiteActions to See how the custom page display
WebPart and Customization
Hey this is the label

How can we implement customization and personalization in WebParts?

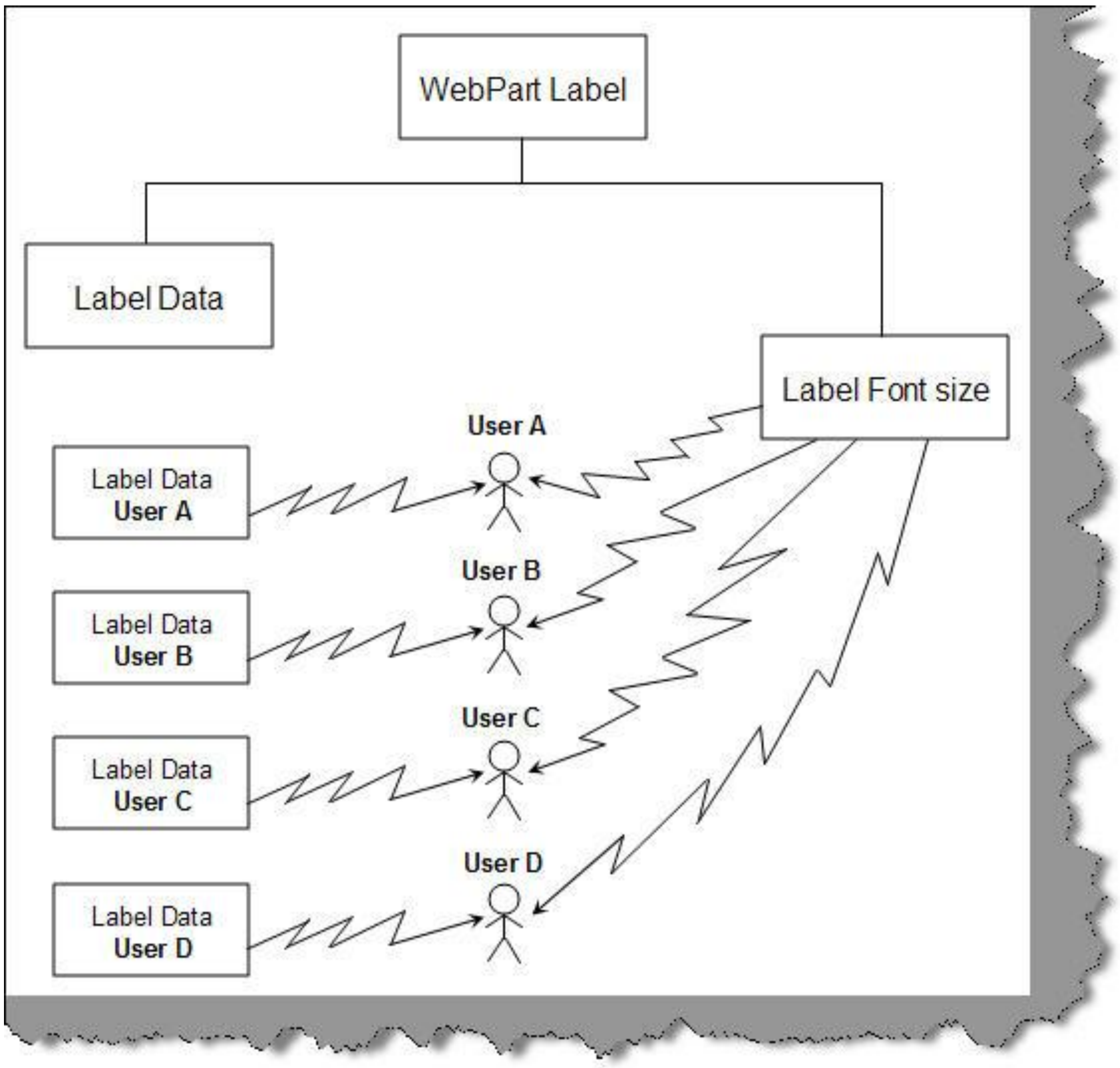
Personalization and customization is all about improving individual user experience as per personal choices. Every human is different and everybody wants that the world moves as per their choice. The same holds true for websites. For instance, your website can have features like weather forecast, business news, and sports news. As an admin, you have the capability to change the look and feel of the website which gets displayed to all users. Some individual users would like to only see news and not weather forecast. So the website should provide some kind of functionality where the users can go and personalize their choices so that the website behaves according to their choices.

The capability where the admin can change the look and feel is called customization. This change is reflected for every viewer who comes to the site. The capability that individual users can either choose to view the news with weather forecast or not is called personalization.

Just cutting it short, when changes made to a site are shared by all users, it’s called customization. When we make changes to some feature which is only seen by individual users, it is termed as personalization.



Let’s try to understand personalization and customization in SharePoint using a small sample. Let’s make a label whose text / data can be personalized by every user, but the font size can only be customized by the admin. The font size customization activity is only limited to the admin and cannot be personalized by the end user.



Let's create the label WebPart which we discussed. We will create a webpart whose label data can be personalized and the font size can only be customized by the admin.

The first thing we need to do is create a class for the webpart. Below is the code snippet of the webpart class. We have created two private properties for font size and label data.

Hide Copy Code

```
public class SimpleLabelCustomization : WebPart
{
    private int _intFontSize;
    private string _strLblData = "";
}
```

The second thing we need to do is define the personalization scope for both of these properties.

We want the label data changes to be personalized by the user. So we need to attribute personalizable topersonalizationscope.user. In order to display the description and display name, we have specifiedWebDisplayName and WebDescription.

Hide Copy Code

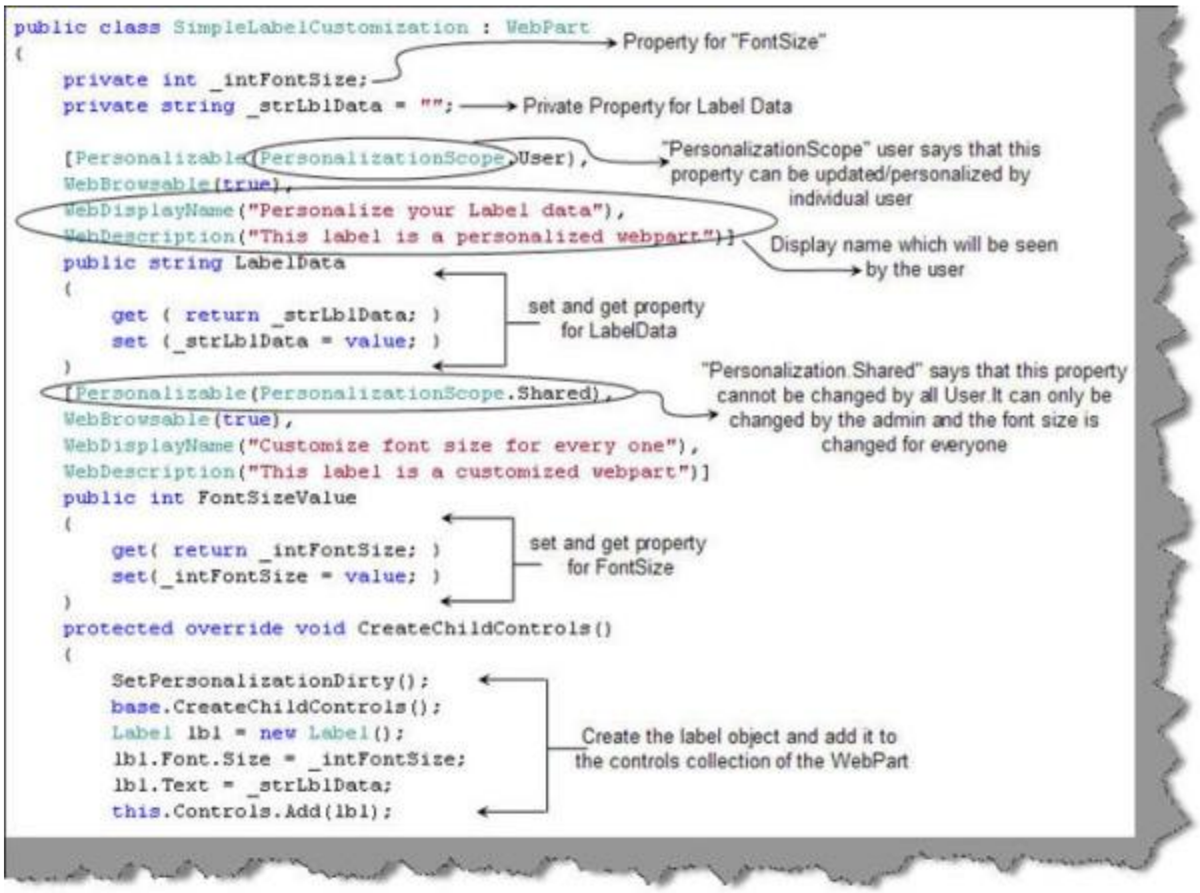
```
[Personalizable(PersonalizationScope.User),
WebBrowsable(true),
WebDisplayName("Personalize your Label data"),
WebDescription("This label is a personalized webpart")]
public string LabelData
{
    get { return _strLblData; }
    set {_strLblData = value; }
}
```

We want font size changes to be viewed by all users so we need to attribute the personalizable value topersonalizationscope.Shared.

Hide Copy Code

```
[Personalizable(PersonalizationScope.Shared),
WebBrowsable(true),
WebDisplayName("Customize font size for every one"),
WebDescription("This label is a customized webpart")]
public int FontSizeValue
{
    get{ return _intFontSize; }
    set{_intFontSize = value; }
}
```

Below is the complete code snippet with explanation.

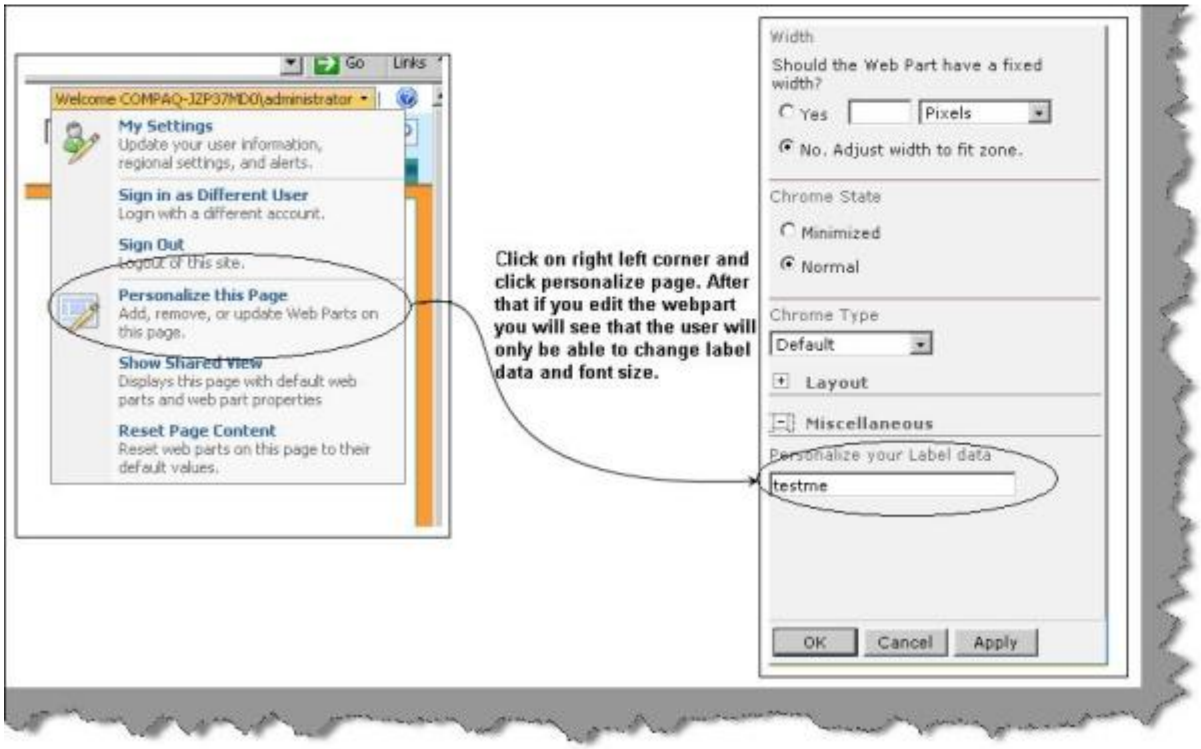

Property for "FontSize"
Private Property for Label Data
"PersonalizationScope" user says that this property can be updated/personalized by individual user
Display name which will be seen by the user
set and get property for LabelData
"Personalization.Shared" says that this property cannot be changed by all User.It can only be changed by the admin and the font size is changed for everyone
set and get property for FontSize
Create the label object and add it to the controls collection of the WebPart

```
public class SimpleLabelCustomization : WebPart
{
    private int _intFontSize;
    private string _strLblData = "";

    [Personalizable(PersonalizationScope.User),
    WebBrowsable(true),
    WebDisplayName("Personalize your Label data"),
    WebDescription("This label is a personalized webpart")]
    public string LabelData
    {
        get { return _strLblData; }
        set { _strLblData = value; }
    }
    [Personalizable(PersonalizationScope.Shared),
    WebBrowsable(true),
    WebDisplayName("Customize font size for every one"),
    WebDescription("This label is a customized webpart")]
    public int FontSizeValue
    {
        get{ return _intFontSize; }
        set{_intFontSize = value; }
    }
    protected override void CreateChildControls()
    {
        SetPersonalizationDirty();
        base.CreateChildControls();
        Label lbl = new Label();
        lbl.Font.Size = _intFontSize;
        lbl.Text = _strLblData;
        this.Controls.Add(lbl);
```

So now that we are done with our webpart, we need to deploy it. You can see the previous section to understand how to deploy a webpart.

Now let's see the fun of personalization. To personalize the page, you need to click on the top right hand corner and then click Personalize page. Now if you edit the webpart, you will see only the label data can be changed.

This is because we have set label data as a user based personalization and font size as a shared personalization.


Welcome COMPAQ-JZP37MD0\administrator
My Settings
Update your user information, regional settings, and alerts.
Sign in as Different User
Login with a different account.
Sign Out
Logout of this site.
Personalize this Page
Add, remove, or update Web Parts on this page.
Show Shared View
Displays this page with default web parts and web part properties
Reset Page Content
Reset web parts on this page to their default values.
Width
Should the Web Part have a fixed width?
Yes
Pixels
No. Adjust width to fit zone.
Chrome State
Minimized
Normal
Chrome Type
Default
Layout
Miscellaneous
Personalize your Label data
OK
Cancel
Apply

Click on right left corner and click personalize page. After that if you edit the webpart you will see that the user will only be able to change label data and font size.

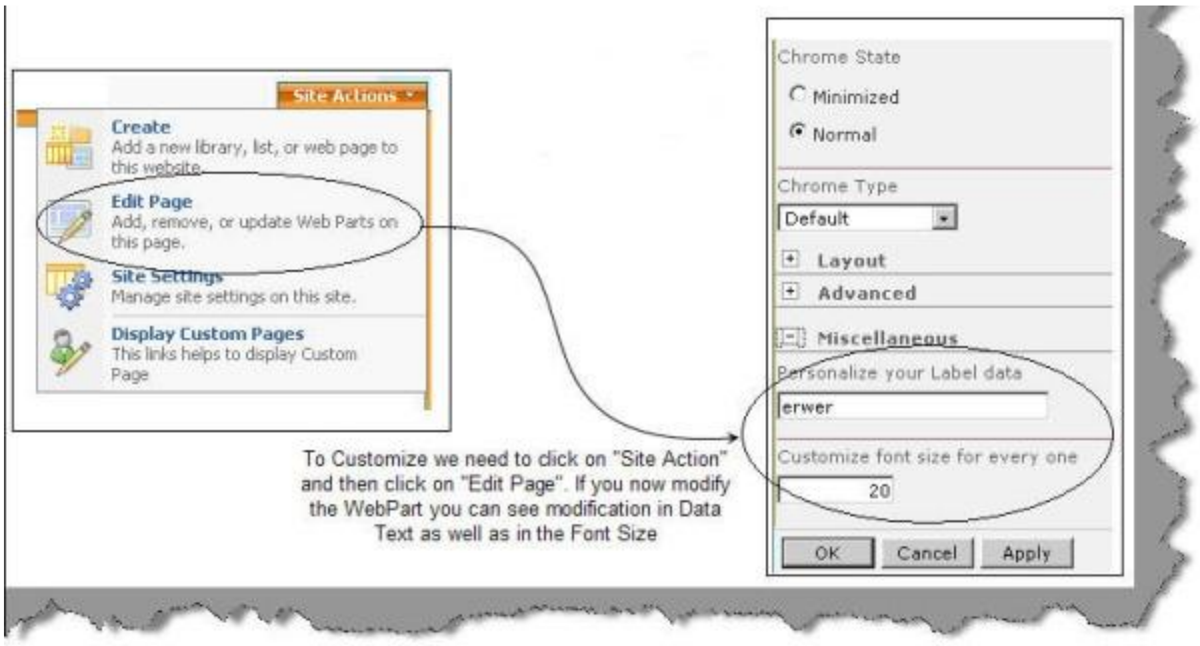
Width: No. Adjust width to fit zone.

Chrome State: Normal

Chrome Type: Default

Personalize your Label data: testme

To customize the webpart, we need to click on Site Action and click Edit Page. You can now see both the properties. The Site Actions menu can be restricted to administrators to have better control on customization and personalization.


Site Actions
Create
Add a new library, list, or web page to this website.
Edit Page
Add, remove, or update Web Parts on this page.
Site Settings
Manage site settings on this site.
Display Custom Pages
This links helps to display Custom Page
Chrome State
Minimized
Normal
Chrome Type
Default
Layout
Advanced
Miscellaneous
OK
Cancel
Apply

To Customize we need to click on "Site Action" and then click on "Edit Page". If you now modify the WebPart you can see modification in Data Text as well as in the Font Size

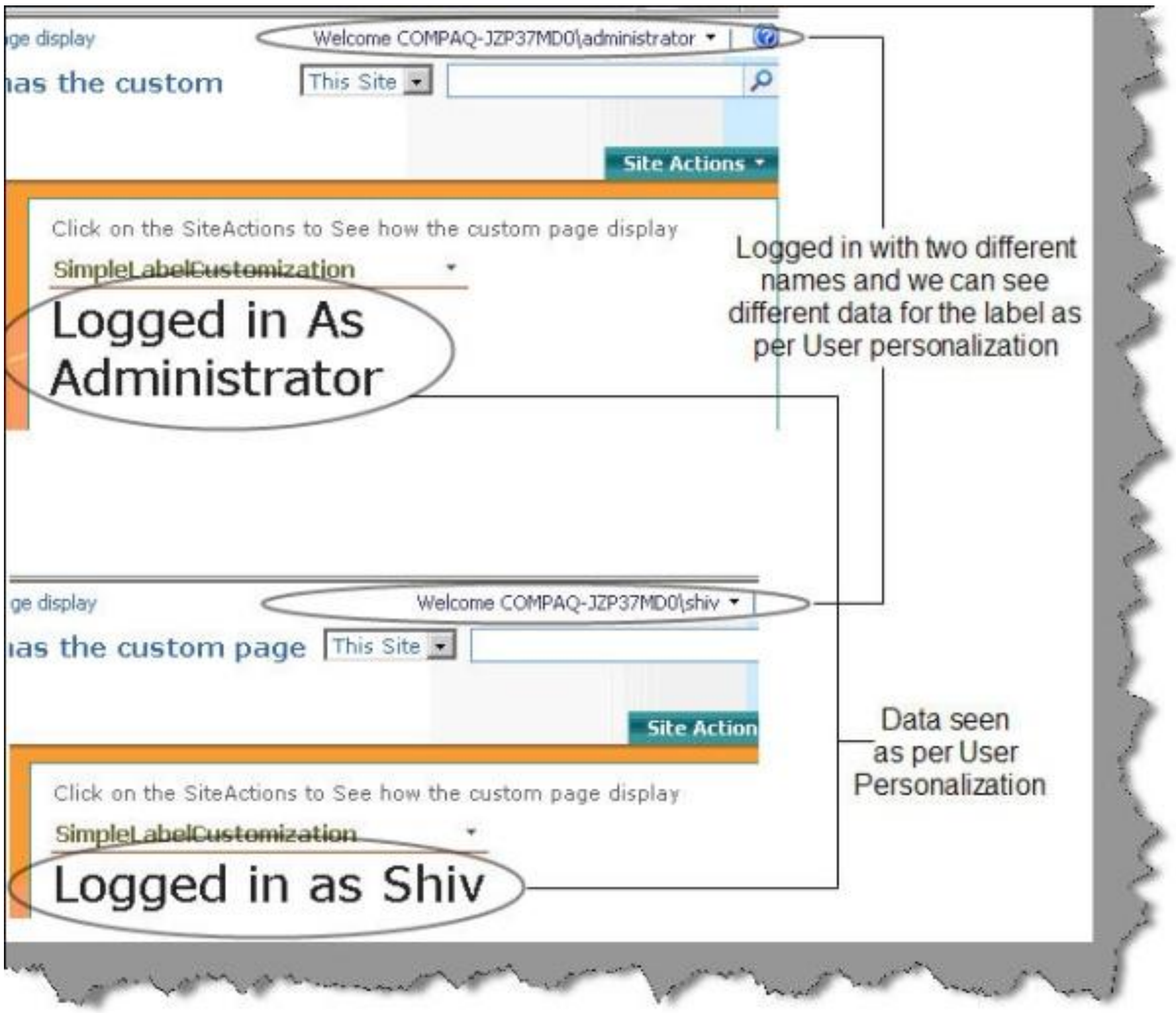
Chrome State: Normal

Chrome Type: Default

Personalize your Label data: erwer

Customize font size for every one: 20

To see the actual action, create two different logins and change data with each user logged in. You will see that depending on what username you have logged in, you will see the label data as per user personalization.


ge display
Welcome COMPAQ-JZP37MD0\administrator
has the custom
This Site
Site Actions
Click on the SiteActions to See how the custom page display
~~SimpleLabelCustomization~~
Logged in As Administrator
Logged in with two different names and we can see different data for the label as per User personalization
ge display
Welcome COMPAQ-JZP37MD0\shiv
has the custom page
This Site
Site Action
Data seen as per User Personalization
Click on the SiteActions to See how the custom page display
~~SimpleLabelCustomization~~
Logged in as Shiv

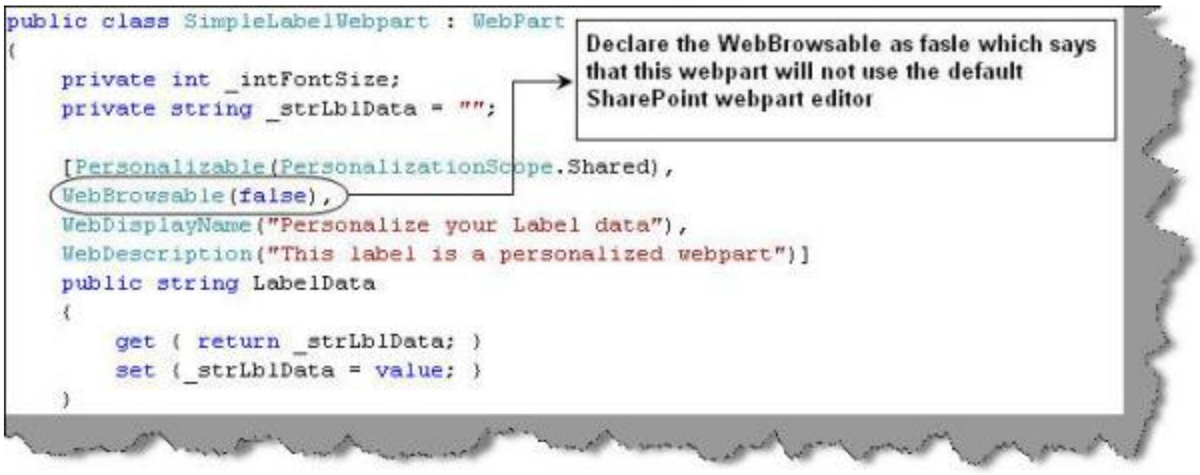
How can we create a custom editor for a WebPart?

There are situations where you want to use your own custom WebPart editor rather than use the default WebPart editor.

Note: In case you do not know how to deploy a WebPart, we advice you to read the answer for the question "Can you explain the six steps we need to create a WebPart in SharePoint?

To make your own custom web part editor is a four step procedure.

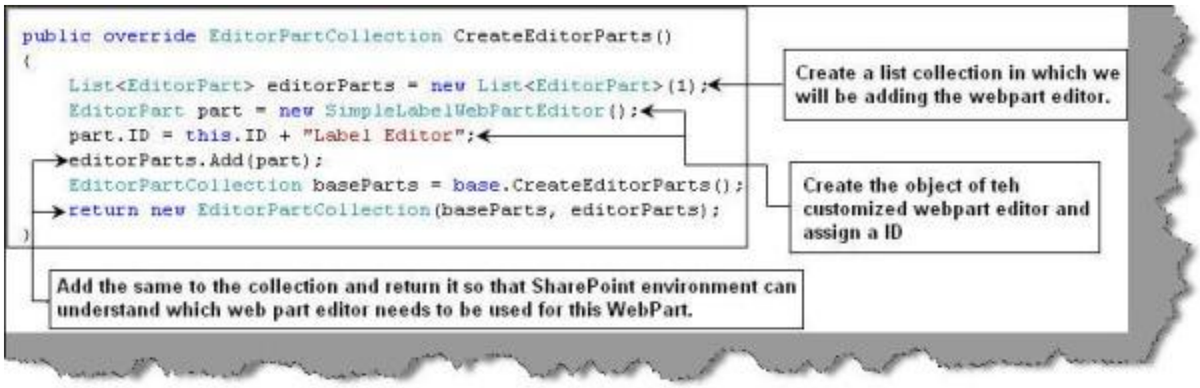
Step 1: The first the thing is to create the WebPart. We need to make a slight change, i.e., '`WebBrowsable (false)`" so that it does not use the default 'WebPart' editor. Below is the code snippet for the same font size 'WebPart' which has '`WebBrowsable`' value as `false`.



```
public class SimpleLabelWebpart : WebPart
{
    private int _intFontSize;
    private string _strLblData = "";

    [Personalizable(PersonalizationScope.Shared),
    WebBrowsable(false),
    WebDisplayName("Personalize your Label data"),
    WebDescription("This label is a personalized webpart")]
    public string LabelData
    {
        get { return _strLblData; }
        set {_strLblData = value; }
    }
```

Declare the WebBrowsable as fasle which says that this webpart will not use the default SharePoint webpart editor

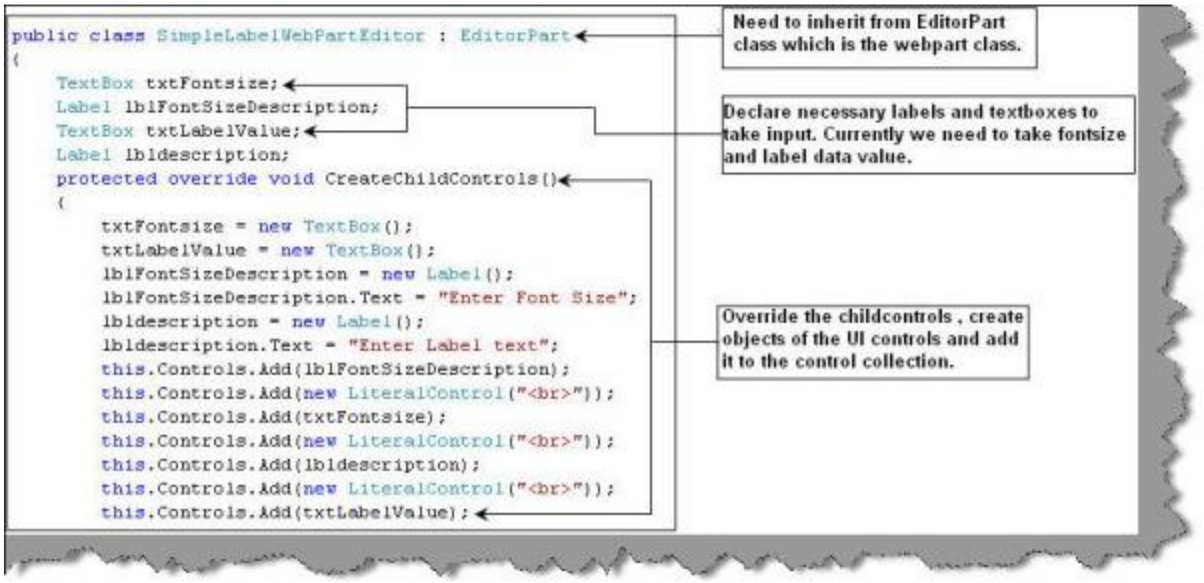
Step 2: The other thing which we need to do in the WebPart is override the '`CreateEditorParts`' method and add our custom WebPart editor. We will be seeing in the next step how we can create the web part editor. For this example, we have created '`SimpleWebPartEditor`'.



```
public override EditorPartCollection CreateEditorParts()
{
    List<EditorPart> editorParts = new List<EditorPart>(1);
    EditorPart part = new SimpleLabelWebPartEditor();
    part.ID = this.ID + "Label Editor";
    editorParts.Add(part);
    EditorPartCollection baseParts = base.CreateEditorParts();
    return new EditorPartCollection(baseParts, editorParts);
}
```

Create a list collection in which we will be adding the webpart editor.
Create the object of teh customized webpart editor and assign a ID
Add the same to the collection and return it so that SharePoint environment can understand which web part editor needs to be used for this WebPart.

Step 3: In order to create your custom web part editor, the first step we need to do is inherit from the`EditorPart` class. Currently we have created a custom web part editor class called `SimpleLabelWebPart`. In this custom class, we have created two labels and two text boxes. The two textboxes will take the font size and label value while the labels will display the description for those text boxes. In other words, we have defined our UI which takes inputs which can customize our web part.



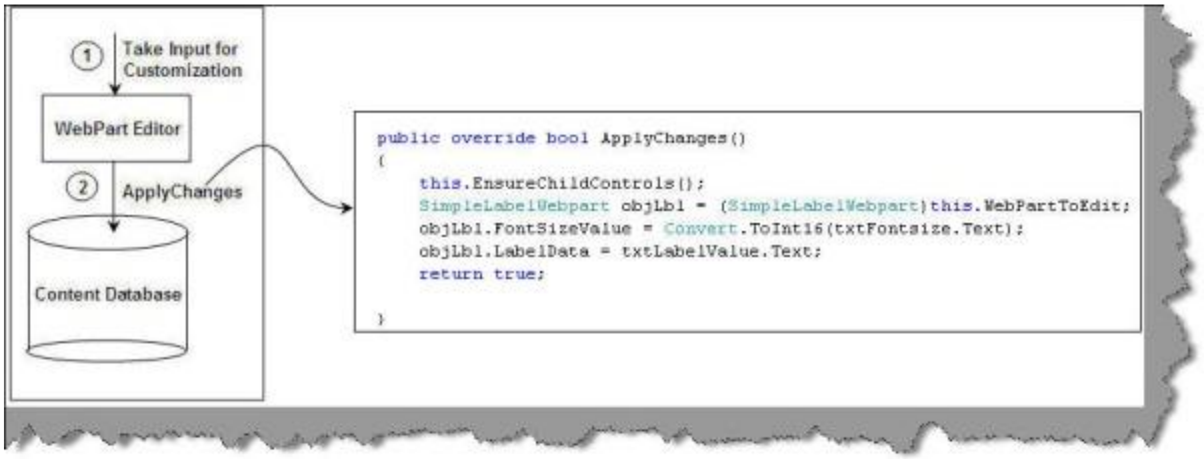
```
public class SimpleLabelWebPartEditor : EditorPart
{
    TextBox txtFontsize;
    Label lblFontSizeDescription;
    TextBox txtLabelValue;
    Label lbldescription;
    protected override void CreateChildControls()
    {
        txtFontsize = new TextBox();
        txtLabelValue = new TextBox();
        lblFontSizeDescription = new Label();
        lblFontSizeDescription.Text = "Enter Font Size";
        lbldescription = new Label();
        lbldescription.Text = "Enter Label text";
        this.Controls.Add(lblFontSizeDescription);
        this.Controls.Add(new LiteralControl("<br>"));
        this.Controls.Add(txtFontsize);
        this.Controls.Add(new LiteralControl("<br>"));
        this.Controls.Add(lbldescription);
        this.Controls.Add(new LiteralControl("<br>"));
        this.Controls.Add(txtLabelValue);
```

Need to inherit from EditorPart class which is the webpart class.

Declare necessary labels and textboxes to take input. Currently we need to take fontsize and label data value.

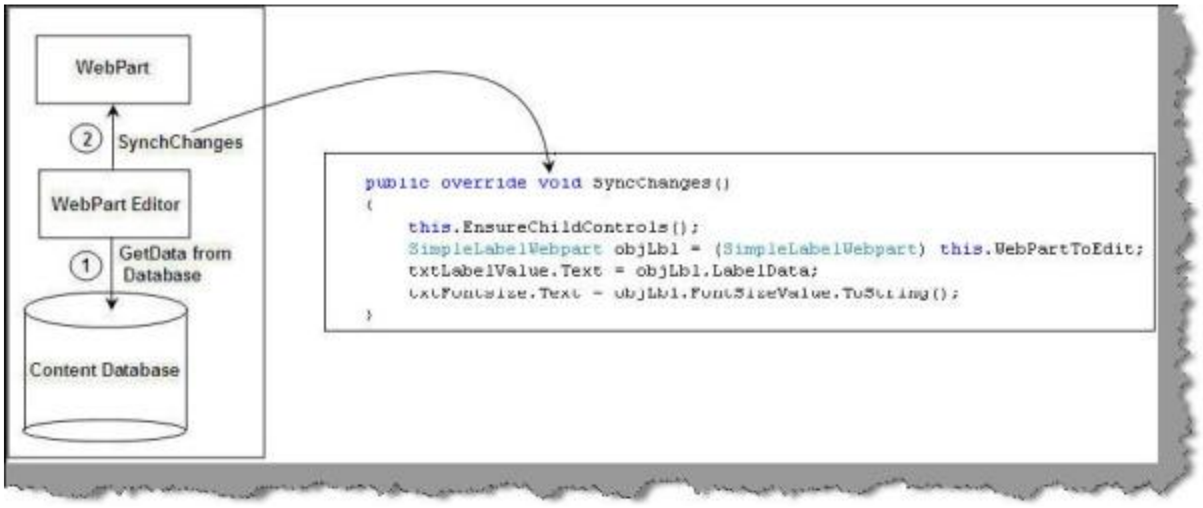
Override the childcontrols , create objects of the UI controls and add it to the control collection.

Step 4: Two activities take place in general, one is when we the user gives customization values which customizes the web part and the other when we need to synchronize the customization values with the web part.



```
public override bool ApplyChanges()
{
    this.EnsureChildControls();
    SimpleLabelWebpart objLbl = (SimpleLabelWebpart)this.WebPartToEdit;
    objLbl.FontSizeValue = Convert.ToInt16(txtFontsize.Text);
    objLbl.LabelData = txtLabelValue.Text;
    return true;

}
```

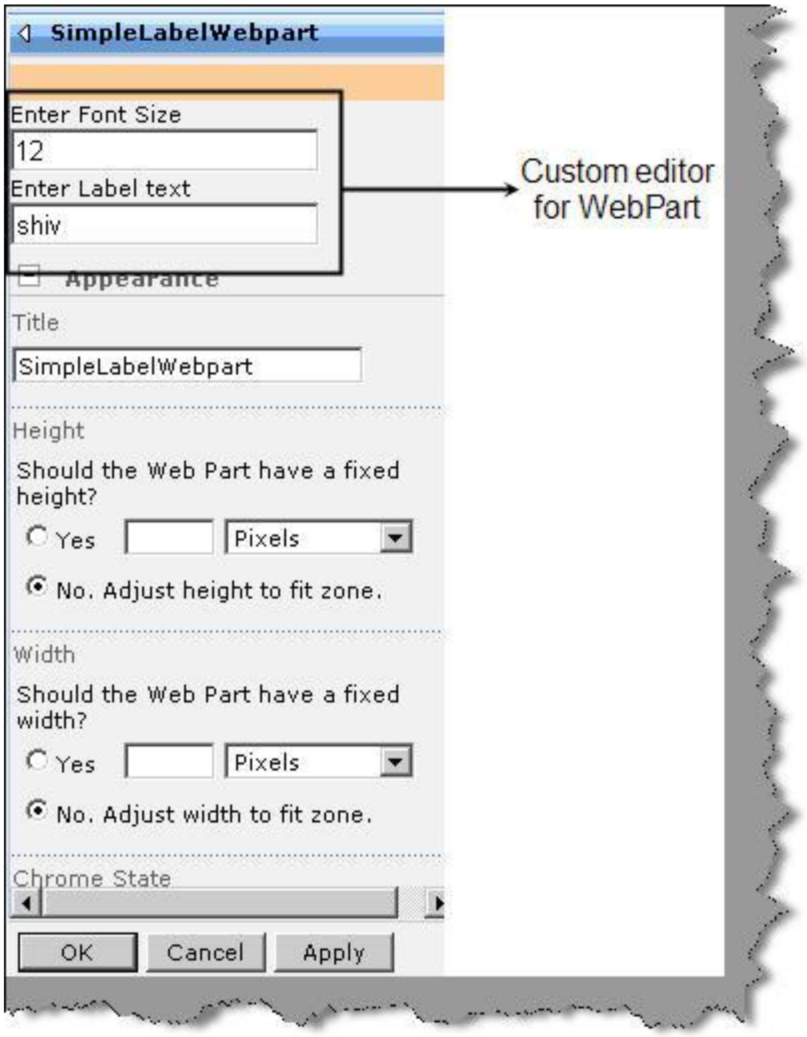
The web part customization values are stored in a content database according to the user logged in. So when customization data is provided for a web part through our custom web part editor, we need to override the`ApplyChanges` method and push the customization data to the web part object. In other words, the data is pushed from the web part editor to the content database on a per user basis.



```
public override void SyncChanges()
{
    this.EnsureChildControls();
    SimpleLabelWebpart objLbl = (SimpleLabelWebpart) this.WebPartToEdit;
    txtLabelValue.Text = objLbl.LabelData;
    txtFontsize.Text = objLbl.FontSizeValue.ToString();
}
```

Now comes the second aspect where we need to synch the web part data from the content database with the web part itself. For this, we need to override the `syncchanges` method and push the data from the web part object into the web part editor user interface.

Now if you edit your web part, you can see how the custom editor looks like.



Enter Font Size
12
Enter Label text
shiv

Title
SimpleLabelWebpart

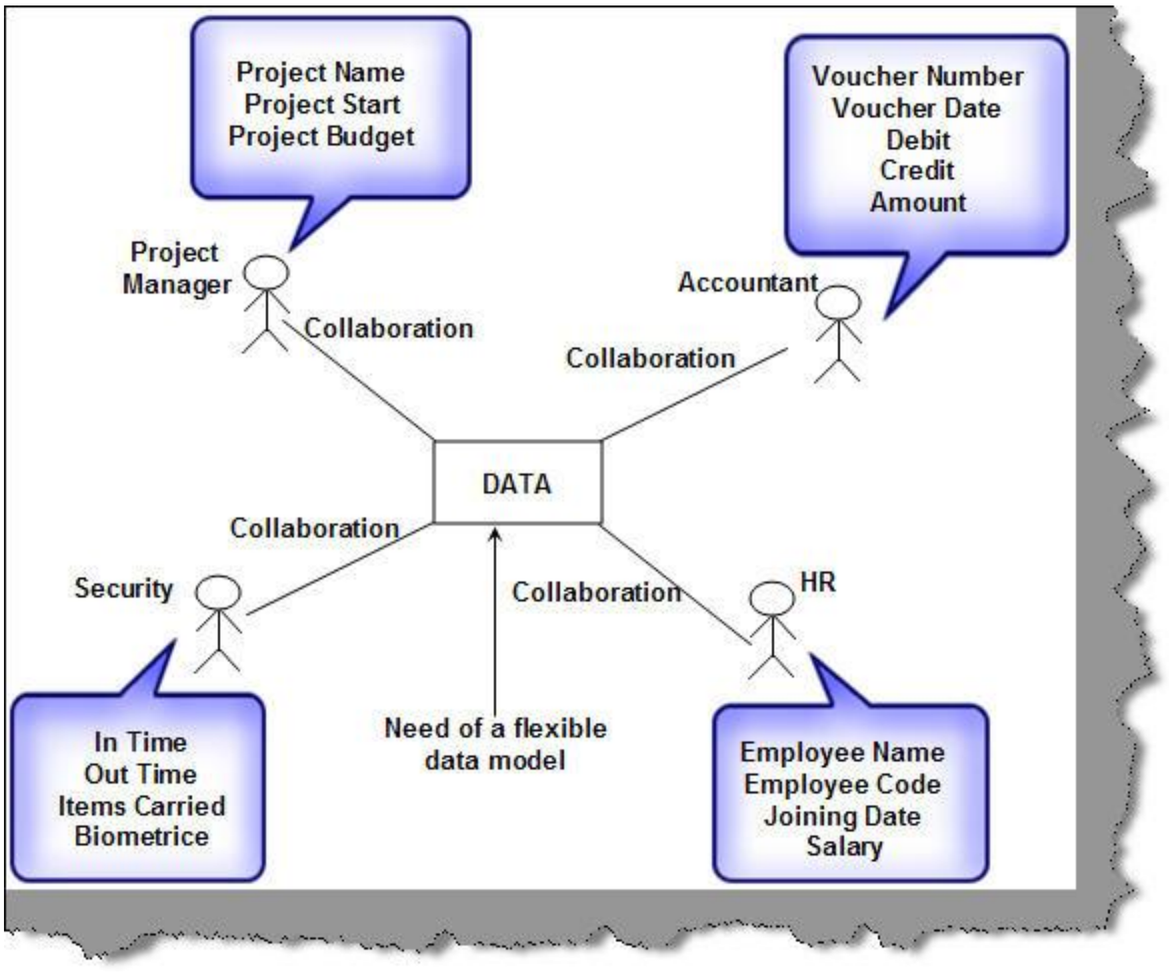
Height
Should the Web Part have a fixed height?
Yes [✓ No. Adjust height to fit zone.] Pixels

Width
Should the Web Part have a fixed width?
Yes [✓ No. Adjust width to fit zone.] Pixels

Chrome State

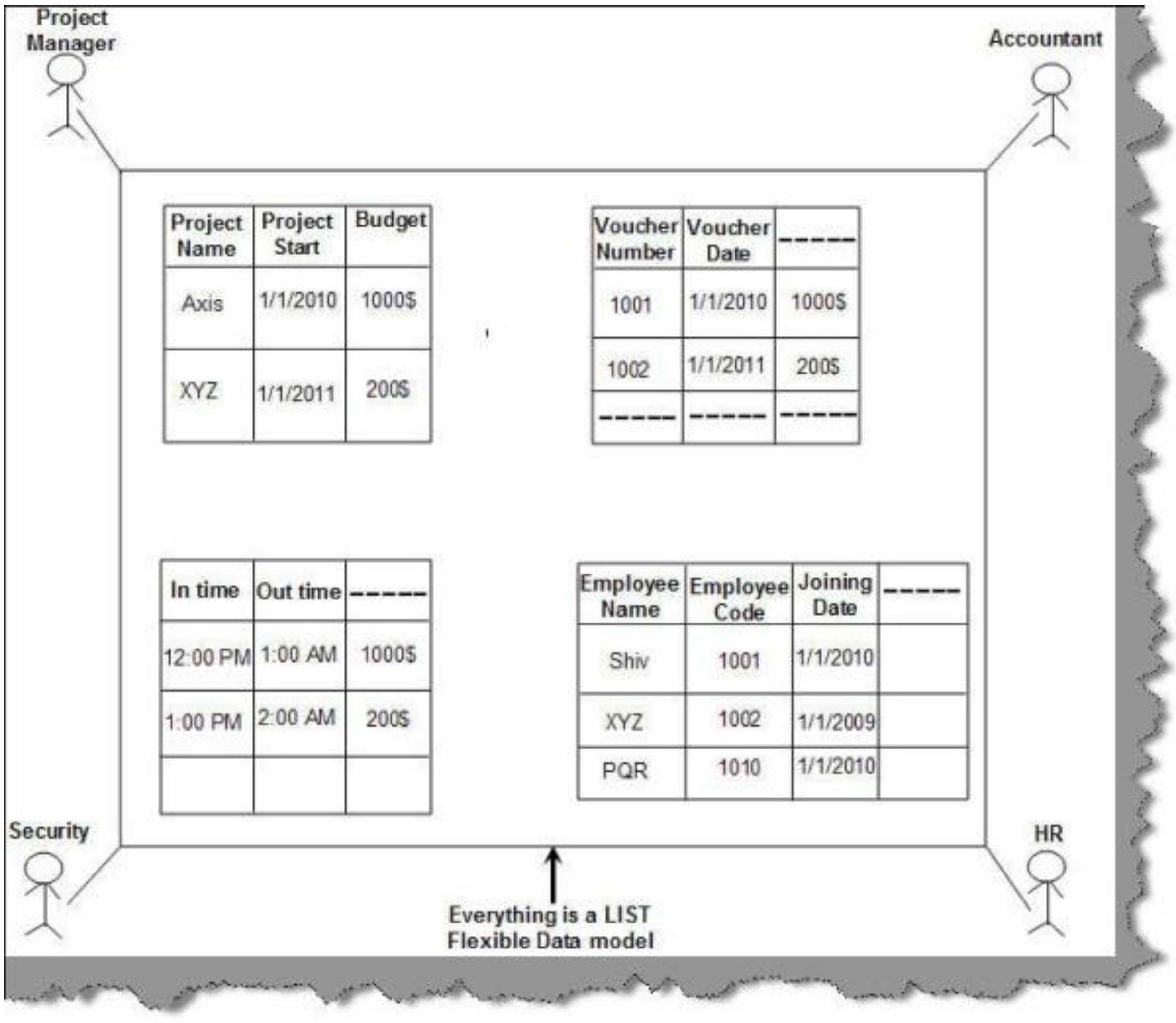
What are List and Content Types in SharePoint?

SharePoint is all about data and collaboration of data between different types of users. When we talk about data, it varies from user to user and different types of meta-data. For instance, below we have four kinds of users: project manager, accountant, security personnel, and HR.



All these four users have different kinds of data formats. For instance, the accountant needs to save data with different data types and fields like voucher number (20 character string), voucher date (MM/DD/YYYY), debit (Boolean true), credit (Boolean false), and amount (double). The security needs in-time, out-time, items carried, and biometrics data.

In order to bring a flexible collaborative data model which can be further extended, SharePoint brought in something called List and Content Types. Data is visualized in terms of List data as shown below. With this approach, you can add more fields and content type on a needed basis.



Project Name	Project Start	Budget
Axis	1/1/2010	1000$
XYZ	1/1/2011	200$

Voucher Number	Voucher Date	-----
1001	1/1/2010	1000$
1002	1/1/2011	200$
-----	-----	-----

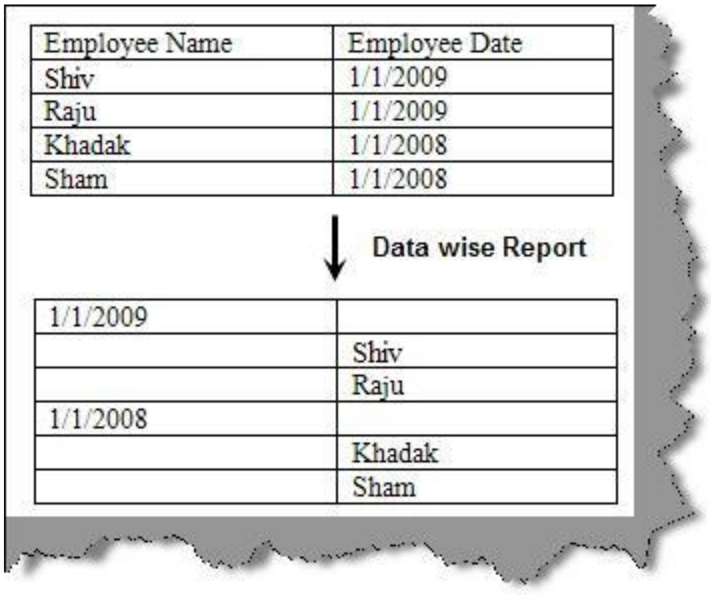
In time	Out time	-----
12:00 PM	1:00 AM	1000$
1:00 PM	2:00 AM	200$

Employee Name	Employee Code	Joining Date	-----
Shiv	1001	1/1/2010	
XYZ	1002	1/1/2009	
PQR	1010	1/1/2010	

The best part about organizing data in the list is the amount of flexibility and extensibility we can bring in. Some of the flexibility a list type of approach brings in are given below:

Different views of the list

For instance, an employee list can be viewed employee wise or date wise.



Employee Name	Employee Date
Shiv	1/1/2009
Raju	1/1/2009
Khadak	1/1/2008
Sham	1/1/2008

Data wise Report

1/1/2009	
	Shiv
	Raju
1/1/2008	
	Khadak
	Sham

Create content from external list storage

You can take Excel and instruct SharePoint to create a list of that type.

Which are the various readymade list types that SharePoint provides currently?

There are approximately thirteen different readymade list types, below mentioned are some of the most used ones.

Document library

Used for collaborating on documents with support for versioning, check-in and check-out, and workflow. Includes support for deep integration with Microsoft Office.

Form library

Used to store XML documents and forms for use with Microsoft Office InfoPath.

Wiki page library

Used for collaborative web pages based on wiki pages, which are dynamically generated and collaboratively edited web pages.

Picture library

A specialized document library enhanced for use with pictures. Includes support for slideshows, thumbnails, and simple editing through Microsoft Office Picture Manager.

Announcements

Used for simple sharing of timely news with support for expiration.

Contacts

A list for tracking people and contact information, with support for integration into Microsoft Office Outlook and other WSS-compatible contacts applications.

Discussions

A simple list for threaded discussions with support for approval and managing discussion threads.

Links

A list for managing hyperlinks.

Calendar

A list for tracking upcoming events and deadlines. Includes support for integration and synchronization with Office Outlook.

Tasks

A list of activity-based items that can integrate with workflow.

Project tasks

An enhanced tasks list with support for Gantt chart rendering and integration with Microsoft Office Project.

Issue tracking

A list for tracking issues and resolution, with support for prioritization.

Custom list

An empty list definition for extending with custom columns, or created using Microsoft Office Excel spreadsheets.