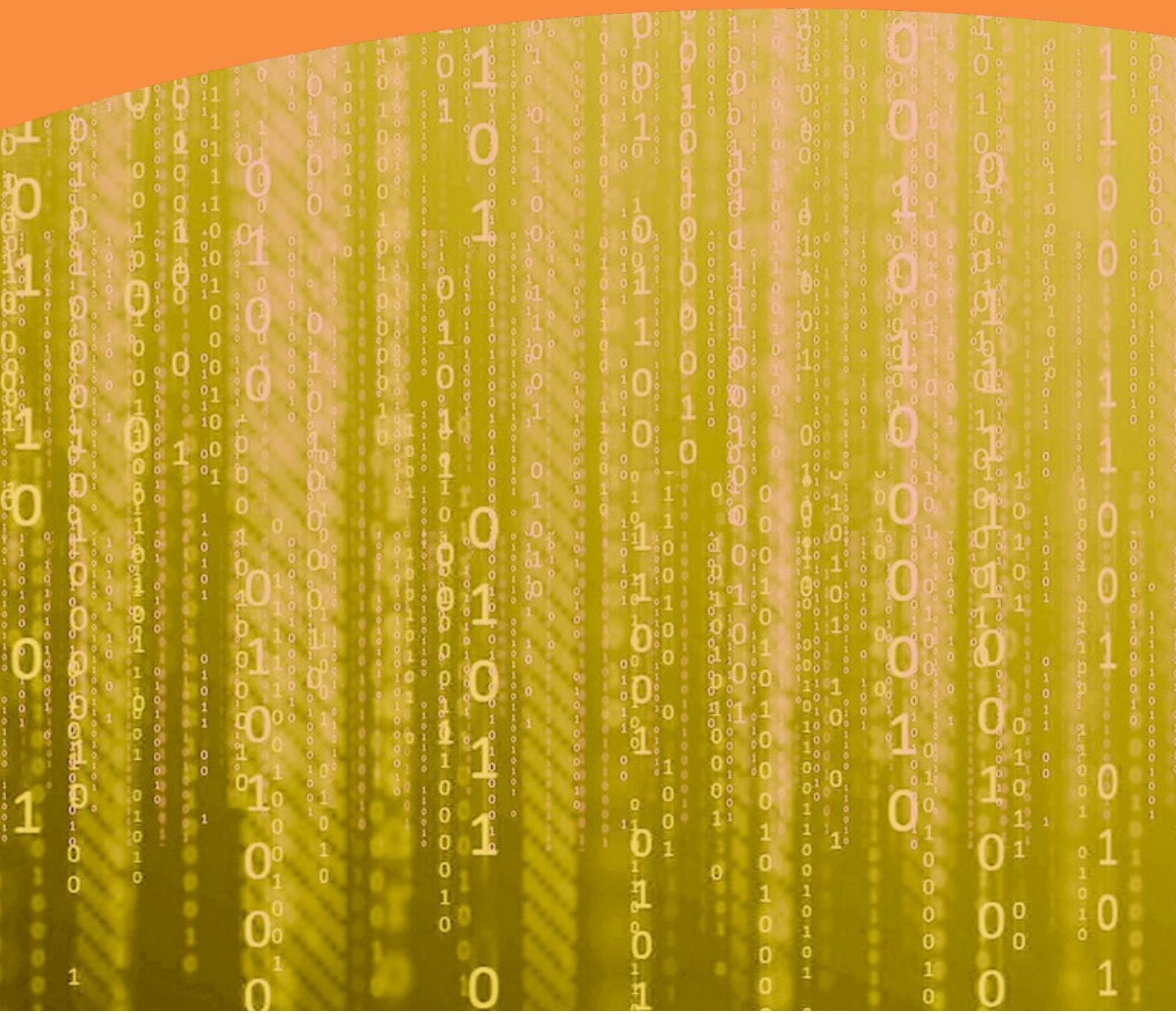
SharePoint Development

Understanding the Server Side Object Model (SSOM)

Mujthaba Hassan



bookboon
The eBook company

MUJTHABA HASSAN

SHAREPOINT DEVELOPMENT

UNDERSTANDING THE SERVER SIDE OBJECT MODEL (SSOM)

SharePoint Development: Understanding the Server Side Object Model (SSOM)
1st edition

ISBN 978-87-403-2100-5

CONTENTS

AUTHOR BIOGRAPHY

Mujthaba Hasan is a SharePoint consultant and a technology enthusiast. He has worked for some of the best companies around the world like Procter and Gamble (P&G), Exceed IT Services (Microsoft partner of the year, Gulf) & Infotouch (SME 40, Dubai). He is a consultant who specializes in developing enterprise solutions using SharePoint. He is Microsoft Certified Solutions Developer in SharePoint Applications & App Builder. He completed his Masters in Computer Application from SJCE, India with distinction.

Mujthaba can be contacted through email at mujthabahassan@gmail.com or reached by phone on +971-561882934.

ACKNOWLEDGEMENT

Firstly thanks to Allah for having provided me with wonderful opportunities to meet and collaborate with some brilliant minds. I would like to dedicate this book to my grandmother, Rahmathunissa for being a constant support and my parents, Hayath and Imrana for their constant love and encouragement.

I would like to thank my family, Abu Mohammad, the late Najmunissa, Naila, Fahad, Mohaab, Safa and Marwa for bringing joy into my life.

A special thanks to all my teachers, professors and mentors. I would also like to thank my managers and colleagues who believed in me and supported me. I wish to thank all my friends who had a positive influence on my life.

The staff at Bookboon has been a pleasure to work with. Karin Jakobsen has been attentive and patient to my questions. I couldn't have imagined a better first experience with authoring.

1 INTRODUCTION

1.1 A BRIEF OVERVIEW OF SHAREPOINT

SharePoint is a business collaboration platform developed by Microsoft, it was initially released on 2001 as SharePoint Portal Server 2001 and the current version being released is SharePoint 2016. SharePoint can be described as a complex CMS with high level of features and customizations. SharePoint's high customizability makes it a desirable platform for the development of a wide variety of business solutions such as intranet portals, public facing websites, financial applications, document management system, project management tool etc.

SharePoint is used across the organizations for ease of collaboration, efficient management of information, automation of business processes and social interaction to name a few.

SharePoint offers many tools for solving business problems like out-of-the-box solutions, Server Side Object Model, Client Side Object Model, SharePoint Designer Workflows, Search Service, etc. The flexibility of the platform and an array of out of the box tools led to its wide adoption by many of the organization across the globe.

1.2 DIFFERENT DEVELOPMENT OPTIONS IN SHAREPOINT

SharePoint offers two development options:

- Server Side Object Model:
  Server Side Object Model (SSOM) sometimes referred to as Server Object Model (SOM), is used to develop solutions that are deployed on the server. It provides a programming interface to interact with SharePoint's data and configuration.
  We usually make use of the C# language to develop using SSOM.

- Client Side Object Model:
  Client Side Object Model is commonly referred to as CSOM. CSOM provides an API to write code that runs on client side. CSOM usually runs on the browser. Solutions are usually built by using JavaScript or one of the many JavaScript frameworks.

1.3 WHAT IS SERVER SIDE OBJECT MODEL

An object model refers to the group of objects that are related to each other. Server Object Model refers to the API that provides reference to the objects on the SharePoint Server.

SOM cannot be accessed remotely and has to be used by an application on the same Application pool as the one used by SharePoint.

Microsoft.SharePoint is the assembly that majorly contains the components and Microsoft. SharePoint.dll needs to be added as reference to in your solution to use it. Other prerequisites are the Target framework setting needs to be set to .Net Framework 4.5 and also the platform needs to be set to x64.

The purpose of SOM is when you want to programmatically make changes to the components or data in SharePoint or when you want to build custom solutions handled by code.

When you use SOM, you can use all the features of SharePoint that has been exposed programmatically.

For example a user wants to add data into a SharePoint List but doesn't have the permission to add items. A custom WebPart could be created which provides a form, and saves data to the list once the user enters the details and submits.

1.4 HOW TO USE THIS GUIDE

The purpose of this book is to provide a brief overview of SSOM and to serve as a code reference. In this book we would be making use of the C# language & .NET to interact with SSOM.

The prerequisites to develop SharePoint solutions using SSOM are an understanding of ASP.NET, Visual Studio and C#.

2 FARM & SERVER

SharePoint's logical architecture consists of the farm, which comprises of different SharePoint servers and SQL Servers working in conjunction. A farm can be best described as a collection of servers.

These servers are of different types based on their usage, but are mainly categorized as web front end server and application server. The purpose of the web front end server is to receive and process the end users request and return the data. An application server hosts SharePoint services such as Metadata service, Search Service, Workflow Service, etc.

During development, it is common to use a single server that hosts both webå applications and services.

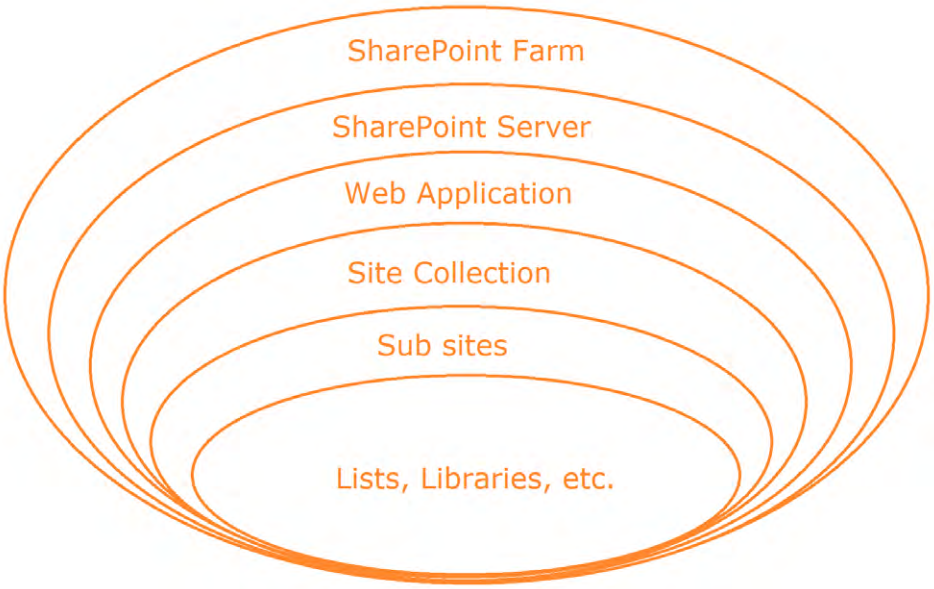


Figure 2.1: SharePoint Logical Architecture

2.1 FARM

The SPFarm is the class that refers to a SharePoint farm.

Using this class you can perform operations to the farm like create a new farm.

Some of the ways to create a farm using the SPFarm class are:

1. Create(SqlConnectionStringBuilder):

This method creates a server farm, its associated configuration database and a farm account on the local computer. The SqlConnectionStringBuilder object specifies the connection string of the configuration database for the new farm.

Example:

```
string connectString = "Server=(local);Database=AppWorld;UID=abcd;Pwd
=P@ssw0rd";

SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder(co
nnectString);

SPFarm.Create(builder);
```

2. Create(String):

This method creates a server farm, its associated database and a farm account on the local computer. The string object contains the connection string for the configuration database of the new server farm.

Example:

```
SPFarm farm = SPFarm.Create("Server = (local); Database = AppWorld; UID
= abcd; Pwd = P@ssw0rd");
```

3. Create (SqlConnectionStringBuilder, String, SecureString, SecureString):

This method is used to create a server farm & it's associated content database, using the user and password specified.

Syntax:

```
public static SPFarm Create(

      SqlConnectionStringBuilder configurationDatabase,

      string farmUser,

      SecureString farmPassword,

      SecureString masterPassphrase

)
```

Example:

```
string connectString = "Server=(local);Database=AppWorld;UID=abcd;Pwd
=P@ssw0rd";

SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder(co
nnectString);

string farmUser;

farmUser = "AppWorld/SPFarm";

SecureString farmPwd = new SecureString();

SecureString masterPassphrase = new SecureString();

ConsoleKeyInfo key;

Console.Write("Enter password: ");

do

{

   key = Console.ReadKey(true);

   // Ignore any key out of range.

   if (((int)key.Key) >= 65 && ((int)key.Key <= 90))
```

```
    {
        // Append the character to the password.
        farmPwd.AppendChar(key.KeyChar);
        Console.Write("*");
    }
   // Exit if Enter key is pressed.
} while (key.Key != ConsoleKey.Enter);
Console.WriteLine();
Console.Write("Enter Master Passphrase: ");
do
      {
        key = Console.ReadKey(true);
        // Ignore any key out of range.
        if (((int)key.Key) >= 65 && ((int)key.Key <= 90))
        {
          // Append the character to the password.
          masterPassphrase.AppendChar(key.KeyChar);
          Console.Write("*");
        }
        // Exit if Enter key is pressed.
      } while (key.Key != ConsoleKey.Enter);
Console.WriteLine();
SPFarm.Create(builder, farmUser, farmPwd, masterPassphrase);
```

4. Create(SqlConnectionStringBuilder, SqlConnectionStringBuilder, String, SecureString, SecureString):

This method creates a server farm and its associated configuration database, based on the user credentials provided.

Syntax:

```
public static SPFarm Create(

      SqlConnectionStringBuilder configurationDatabase,

      SqlConnectionStringBuilder administrationContentDatabase,

      string farmUser,

      SecureString farmPassword,

      SecureString masterPassphrase
)
```

Example:

```
string connectStringConfigDb = "Server=(local);Database=AppWorld;UID=ab
cd;Pwd=P@ssw0rd";

        SqlConnectionStringBuilder configDbBuilder = new SqlConnectionSt
        ringBuilder(connectStringConfigDb);

        string connectStringAdminContentDb = "Server=(local);Database=A
        ppWorld_Admin;UID=abcd;Pwd=P@ssw0rd";

        SqlConnectionStringBuilder AdminContentDbbuilder = new SqlConne
        ctionStringBuilder(connectStringAdminContentDb);

        string farmUser = "Appworld/SPFarm";

        string strfarmPwd = "P@ssw0rd";

        string strmasterPassphrase = "Phr@se1";

        SecureString farmPwd = new SecureString();

        SecureString masterPassphrase = new SecureString();
```

```
foreach (char c in strfarmPwd)
  farmPwd.AppendChar(c);

foreach (char c in strmasterPassphrase)
  masterPassphrase.AppendChar(c);

SPFarm.Create(configDbBuilder, AdminContentDbbuilder, farmUser,
farmPwd, masterPassphrase);
```

In the above example, it is demonstrated how a password stored in string can be appended to a Secure String.

However, this is for demonstration purpose and can be used to automate and simplify the task, but this shouldn't be used in practice as it defeats the purpose of using secure string, which ensures that the password is not stored as a plain text, which results in a security flaw.

5. Open(String)

This method is used to connect to an existing farm, you provide the SQL Server Connection in string format. The connection string corresponds to the farm configuration database that was created during the configuration of the farm. The prerequisites for this method to work are you need access to the remote configuration database, required permissions must be granted, and the database should be on the same network as the local machine.

Syntax:

```
public static SPFarm Open(

        string connectionString

)
```

Code:

```
SPFarm farm = SPFarm.Open("Server = (local); Database = AppWorld; UID
= abcd; Pwd = P@ssw0rd");
```

6. Open(SqlConnectionStringBuilder, SecureString)

Another way to connect an existing farm.

Syntax:

```
public static SPFarm Open(

        SqlConnectionStringBuilder connectionString,

        SecureString passphrase

)
```

Code:

```
string connectString = "Server=(local);Database=AppWorld;UID=abcd;Pwd
=P@ssw0rd";

SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder(co
nnectString);

string strmasterPassphrase = "Phr@se1";

SecureString masterPassphrase = new SecureString();

        foreach (char c in strmasterPassphrase)

          masterPassphrase.AppendChar(c);

SPFarm farm = SPFarm.Open(builder, masterPassphrase);
```

7. SPFarm.Local

Using Local, the static property of SPFarm class, one can connect directly to the local farm.

Code:

```
SPFarm farm = SPFarm.Local;
```

8. SPFarm.Joined

This property is used to check if the local server is joined to any server farm. It returns a Boolean value, true if it is connected and false if it is not connected.

Code:

```
bool isJoined = SPFarm.Local.Joined;
```

9. Join

This method adds the local computer the server farm. If there is no configuration database then a new farm is created.

Code:

```
SPFarm.Local.Join();
```

10. SPFarm.Unjoin

This method removes the local server from the farm.

Code:

```
SPFarm.Local.Unjoin();
```

2.2 SERVER & SERVICES

Within a server, we have web applications that contains IIS web site and is the logical unit for site collections. Each web application has an associated IIS web site with dedicated or shared application pool.

Once you obtain the reference to a SharePoint Farm using SPFarm class, you can access and manage the various servers and services that belong to that farm. The following classes can be assessed:

- Servers: You can enumerate through the physical servers that belong to the farm using SPServer object.
- Services: You can access all services using the common base class SPService. To access specific services you need to make use of the specific classes corresponding to that service, such as SPWindowsService for windows services, SPWebService for web services. A web service is composed of a single or multiple web applications.

The following code shows how to make use of the SPFarm and loop through the details of the server on the farm using SPServer object:

```
SPFarm farm = SPFarm.Local;

Console.WriteLine("Servers on this Farm");

foreach (Microsoft.SharePoint.Administration.SPServer server in farm.
Servers)

{

        Console.WriteLine("Server Name: {0}", server.Name);

        Console.WriteLine("Server Address: {0}", server.Address);

        Console.WriteLine("Server Role: {0}", server.Role);

        Console.WriteLine("Server GUID: {0}", server.Id);

}
```

You can access the services on the farm through the Farm object. The web services running on the farm can be accessed through the objects of type SPWebService, the windows services can be accessed through the objects of type SPWindowsService, the web service comprising of one or multiple web applications can be accessed through the objects of type SPWebApplication. All these services share a common base class called SPService.

The following code demonstrates how to make use of these classes:

```
      SPFarm farm = SPFarm.Local;

        Console.WriteLine("Servers on this Farm");

        foreach (SPService service in farm.Services)

        {

          if (service is SPWindowsService) {

            Console.WriteLine("Windows Service Name: {0}", service.
            DisplayName);

            Console.WriteLine("Type: {0}", service.TypeName);

            Console.WriteLine("Instances: {0}", service.Instances.Count);

            Console.WriteLine("Status: {0}", service.Status);
```

```
            Console.WriteLine("Applications: {0}", service.Applications);
            Console.WriteLine("GUID: {0}", service.Id);
            Console.WriteLine("Running Jobs: {0}", service.RunningJobs);
          }
          else if (service is SPWebService)
          {
            Console.WriteLine("Web Service Name: {0}", service.
            DisplayName);
            Console.WriteLine("Type: {0}", service.TypeName);
            Console.WriteLine("Instances: {0}", service.Instances.Count);
            Console.WriteLine("Status: {0}", service.Status);
            Console.WriteLine("Applications: {0}", service.Applications);
            Console.WriteLine("GUID: {0}", service.Id);
            Console.WriteLine("Running Jobs: {0}", service.RunningJobs);
          }
          else
          {
            Console.WriteLine("General Service Name: {0}", service.
            DisplayName);
            Console.WriteLine("Type: {0}", service.TypeName);
            Console.WriteLine("Instances: {0}", service.Instances.Count);
            Console.WriteLine("Status: {0}", service.Status);
            Console.WriteLine("Applications: {0}", service.Applications);
            Console.WriteLine("GUID: {0}", service.Id);
            Console.WriteLine("Running Jobs: {0}", service.RunningJobs);
          }
        }
Console.ReadKey();
```

3 WEB APPLICATION

A web application is defined as an IIS website configured to run SharePoint. Each web application has one IIS website associated with it, and there can be up to five IIS websites associated to a single web application. Each web application has a minimum of one content database. Microsoft has limited the number of content databases on a SharePoint farm to five hundred.

Code example for looping through the web applications and it's associated content databases in a farm:

```
SPFarm farm = SPFarm.Local;

foreach (SPService service in farm.Services)

  {

    if (service is SPWebService)

    {

      SPWebService webService = service as SPWebService;

      if (webService != null)

      {

        foreach (SPWebApplication webApplication in

        webService.WebApplications)

        {

          Console.WriteLine("Name of Web Application: {0}",
          webApplication.DisplayName);

          Console.WriteLine("Content Databases of the web application");

          foreach (SPContentDatabase db in webApplication.
          ContentDatabases)

          {
```

```
                Console.WriteLine("Content Database: {0}", db.Name);

                Console.WriteLine("Connection String: {0}",
                db.DatabaseConnectionString);

              }

            }

          }

        }

Console.ReadKey();
```

Some of the commonly used properties and methods for the web application are:

1. SPWebApplication.Lookup method:
   This method finds the web application located at the specified URL, and returns the SPWebApplication object.

Example:

```
SPWebApplication oWebApplication = SPWebApplication.Lookup(new
Uri("http://MyTeamServer:123"));
```

2. SPWebApplication.AlertsEnabled property
   The AlertsEnabled property is used to get or set whether alerts are allowed in the web application. True indicates that it is allowed and false indicates that it is not allowed.

Example:

```
SPWebApplication oWebApplication = SPWebApplication.Lookup(new
Uri("http://MyTeamServer:123"));

bool isAlertEnabled = oWebApplication.AlertsEnabled;

Console.WriteLine("Alert Enabled: {0}", isAlertEnabled);
```

3. SPWebApplication.AllowDesigner property
   This property is used to get or set whether the websites within this web application can be edited using SharePoint Designer.

Example:

```
SPWebApplication oWebApplication = SPWebApplication.Lookup(new
Uri("http://MyTeamServer:123"));

oWebApplication.AllowDesigner = true;
```

4. SPWebApplication.CanSelectForBackup property
   This property is used to get or set whether the web application can be backed up. If true then the web application can be backed up, otherwise false. The default value is true.

Example:

```
SPWebApplication oWebApplication = SPWebApplication.Lookup(new
Uri("http://MyTeamServer:123"));

oWebApplication.CanSelectForBackup = true;
```

5. SPWebApplication.CanSelectForRestore property
   This property is used to get or set whether the web application can be restored. If true then the web application can be restored, otherwise false. The default value is true.

Example:

```
SPWebApplication oWebApplication = SPWebApplication.Lookup(new
Uri("http://MyTeamServer:123"));

oWebApplication.CanSelectForRestore = true;
```

6. SPWebApplication.CanRenameOnRestore property
   This property is used to get the value whether the web application can be renamed when it is restored. If true then the web application can be renamed, otherwise false.

Example:

```
SPWebApplication oWebApplication = SPWebApplication.Lookup(new
Uri("http://MyTeamServer:123"));

bool canRename = oWebApplication.CanRenameOnRestore;
```

7. SPWebApplication.MaximumFileSize property
   This property is used to get or set the size of the largest file that can be uploaded. The value is mentioned in MB and the default value is 50MB.

Example:

```
SPWebApplication oWebApplication = SPWebApplication.Lookup(new
Uri("http://MyTeamServer:123"));

oWebApplication.MaximumFileSize = 25;
```

8. SPWebApplication.Delete method
   This method is used to delete the web application and some of its resources. This method however doesn't delete the associated IIS websites.

Example:

```
SPWebApplication oWebApplication = SPWebApplication.Lookup(new
Uri("http://intranet.xyzcomp.com/"));

oWebApplication.Delete();
```

4 APPLICATION POOL

Application pool allows for isolation of different websites on the same server. In this way, if one website crashes in an application pool, the other application pool websites are not affected. It is a means to share the resources of the server and an efficient mechanism to fail proof the server from total failure. In the context of SharePoint each web application and service is associated to an application pool. A web application or service can be associated to a single application pool, but an application pool can be associated to multiple web applications. This enables for better security, performance, availability and reliability.

The following code shows how to create a new application pool:

```
SPFarm farm = SPFarm.Local;

SPWebService service = farm.Services.GetValue<SPWebService>();

SPApplicationPool appPool = new SPApplicationPool("TestAppPool",
service);

appPool.CurrentIdentityType = IdentityType.SpecificUser;

appPool.Username = "ewaytronics\\administrator"; //Domain\\UserName

SecureString Password = new SecureString();

ConsoleKeyInfo key;

Console.Write("Enter password: ");

do

{

  key = Console.ReadKey(true);

  // Append the character to the password & ignore enter character.

  if (key.Key != ConsoleKey.Enter)

  {

    Password.AppendChar(key.KeyChar);

    Console.Write("*");

  }

  // Exit if Enter key is pressed.

} while (key.Key != ConsoleKey.Enter);

appPool.SetPassword(Password);

appPool.Update();

appPool.Deploy();

Console.ReadKey();
```

The following code shows how to change the application pool of a web application.

```
SPWebApplication oWebApplication = SPWebApplication.Lookup(new
Uri("http:// intranet.botronics.com/"));

  SPFarm farm = SPFarm.Local;

  SPWebService service = farm.Services.GetValue<SPWebService>();

  SPApplicationPool appPool = service.ApplicationPools["Name of
  the Application Pool"];

  oWebApplication.ApplicationPool = appPool;

  oWebApplication.Update();

  oWebApplication.ProvisionGlobally();

  Console.WriteLine("{0}",       oWebApplication.ApplicationPool.
  DisplayName);
```

5 SITES

In SharePoint, we have site collection, which refers to a group of sites that share functionalities & features like security, permissions, contents types etc. There is a top level site for every site collection. A sub site is a site that is located within another site. Sites in SharePoint are used as a container of information and data through making use of lists, document libraries and pages to name a few. In Server object model, a site collection is represented through the SPSite class and a site is represented through SPWeb class.

The following code demonstrates to loop through the site collection within a web application:

```
SPWebApplication oWebApplication = SPWebApplication.Lookup(new
Uri("http:// intranet.tream.com/"));

foreach (SPSite site in oWebApplication.Sites)

{

  Console.WriteLine("Sites URL:{0}", site.Url);

  Console.WriteLine("Sites Zone:{0}", site.Zone);

  Console.WriteLine("Root: {0}", site.RootWeb);

}

Console.ReadKey();
```

Details like site url, the zone used to access the site collection and the root web site of the site collection are displayed in the above code.

The SPSite object can be initialized through one of the following constructor methods:

1. public SPSite(Guid id):
   In this method, the GUID of the the site collection is specified.

2. public SPSite(string requestUrl):
   In this method, the requested url for the site collection is specified.

3. public SPSite(Guid id, SPUrlZone zone):
   In this method, the GUID of the site collection and the SPUrlZone is specified. The SPUrlZone is an enumeration and contains the following:
   public enum SPUrlZone {
   Default,
   Intranet,
   Internet,
   Custom,
   Extranet
   }

4. public SPSite(Guid id, SPUserToken userToken):
   In this method, the GUID of the site collection is specified and SPUserToken is specified. SPUserToken is a binary object that contains the identification of a user. This method is usually used to access the site collection with a user that has higher privileges and more control on the application.

5. public SPSite(string requestUrl, SPUserToken userToken):
   In this method, the url of the site collection is specified and SPUserToken is specified. SPUserToken is a binary object that contains the identification of a user. This method is usually used to access the site collection with a user that has higher privileges and more control on the application.

6. public SPSite(Guid id, SPUrlZone zone, SPUserToken userToken):
   In this method, the GUID of the site collection, the SPUrlZone from the enum and SPUserToken is specified.

The SPWeb object can be accessed only through the SPSite object. The SPWeb object is initialized through the OpenWeb method of the SPSite object.

Example:

```
using (SPSite site = new SPSite("http://win-kmp:46466"))
{
  Console.WriteLine("Current Site URL: {0}", site.Url);
  using (SPWeb web = site.OpenWeb("en"))
  {
    Console.WriteLine(web.Title);
  }
}
```

The OpenWeb method consists of the following overloads:

1. public SPWeb OpenWeb():
   This constructor returns the web site that is associated with the url in the SPSite constructor. For Example if the root site is associated in the SPSite, then the root site is returned.

2. public SPWeb OpenWeb(Guid gWebId):
   The GUID of the site is passed in the constructor, and the relevant website is returned.

3. public SPWeb OpenWeb(string strUrl):
   The string contains the server or site relative url. The server-relative url begins with a forward slash ("/") whereas the site relative url doesn't begin with a forward slash.

4. public SPWeb OpenWeb(string strUrl, bool requireExactUrl):
   The string contains the server or site relative url. The requireExactUrl is used to specify whether the exact URL must be supplied. If an exact URL must be supplied then true else false.

5. public SPWeb OpenWeb(string strUrl, SPSiteOpenWebOptions options):
   The string contains the server or site relative url. The SPSiteOpenWebOptions is an enum used to specify whether you wish to initialize the navigation cache.

Some of the popular properties of SPWeb are:

1. AllowAnonymousAccess:
   This property is used to set or get the anonymous access to the website. True indicates that anonymous access is allowed otherwise it is false.

2. AllowUnsafeUpdates:
   This property is used to fetch or set to accept the updates to the database without requiring a security validation for a GET request.
   Setting this property to true opens security risks like cross site scripting vulnerabilities.

3. AllProperties:
   This returns a hash table containing the metadata of the website, such as default language, total visits, total bandwidth etc.

4. AllUsers:
   Gets a collection of users who have browsed through the website as authenticated users.

5. AnonymousPermMask64:
   Gets or sets the permissions for the anonymous users of the website.

6. AnonymousState:
   Gets or sets the level of access granted for the anonymous users of the website.

7. ContentTypes:
   Gets the collection of all content types that are used for the website.

8. Description:
   Gets or sets the description of the website.

9. EventReceivers:
   Gets the collection of event receivers that are currently used in the website.

10. Features:
    Gets the collection of features that are currently activated in the website.

11. Fields:
    Gets the collection of all site columns of the website.

12. Files:
    Gets the collection of all files in the root directory of the website.

13. Folders:
    Gets the collection of all the top level folders on the website.

14. ID:
    Gets the GUID of the website.

15. language:
    Gets the locale identified (LCID) for the website and is in numeric form.

16. Lists:
    Gets the collection of all the lists that are in the website.

17. ListTemplates:
    Gets the collection of all the available list definitions and list templates on the website.

18. Locale:
    Gets or sets the locale of the website. Locale refers to the region specific details used for rendering information such as time, currency and numeric fields.

19. MasterUrl:
    Gets or sets the master page url of the website.

20. Name:
    Gets or sets the name of the website.

21. SiteAdministrators:
    Gets the collection of administrators of the website.

22. SiteGroups:
    Gets a collection of all the groups in the website.

23. Url:
    Gets the absolute url of the website.

24. Users:
    Gets a collection of all the users that have been granted permissions explicitly in the website.

25. Webs:
    Gets a collection of all the websites underneath the website, excluding children of those websites.

26. WebTemplate:
    Gets the name of the site template or definition that was used for the creation of the site.

27. WorkflowAssociations:
    Gets the collection of all the workflows associated to the website.

28. Workflows:
    Gets the collection of all the workflows instances that have run or are running on the website.

Some of the popular members of SPWeb are:

1. public void AddProperty(Object key, Object Property):
   Used to add property to the list of metadata for the website.

2. public void Delete():
   This method is used to delete the website.

3. public void DeleteProperty(Object key);
   This method is used to delete a particular metadata property of the website.

4. public SPFile GetFile(string strUrl):
   This method is used to retrieve the file object from the specified URL.

5. public SPFolder GetFolder(string strUrl):
   This method is used to retrieve the folder located at the specified URL.

6. public SPList GetList(string strUrl):
   This method is used to fetch the list mentioned at the specific URL.

7. public void Recycle():
   This method is used to send the website to the recycle bin.

8. public void Update():
   This method is used to update the databases with the changes that were made on the website.

The following code will show how to create a new site collection:

```
using (SPSite rootSite = new SPSite("http://win-spserv:46466"))
{
 SPWebApplication webApplication = rootSite.WebApplication;
 using (SPSite newSiteCollection = webApplication.Sites.Add(
 "sites/CodeGenerated", // Site URL
 "Generated by Code", // Site Collection Title
 "Sample Site", // Site Collection Description
 1033, // LCID
 15, // Compatibility level (can be 14 for 2010 or 15 for 2013
 or 16 for 2016)
 "CMSPUBLISHING#0", // Web Site Template for a Publishing Site
 "MUJTRONICS\\Administrator", // Owner Login
 "Administrator", // Owner DisplayName
 "administrator@mujtronics.com", // Owner Email
 "MUJTRONICS\\KO", // Secondary Administrator Login
 "Kevin Owens", // Secondary Admin DisplayName
 "ko@mujtronics.com", // Secondary Admin Email
 "SQLSP13 ", // Database Server Name for Content Database
 "WSS_Content_CodeGenerated", // Content Database Name
 null, // Database Login Name
 null // Database Login Password
 ))
 {
```

```
            Console.WriteLine("Created Site Collection: {0}",
            newSiteCollection.Url);
          }
        }
```

The definition of the Add method used in the example above:

public SPSite Add(

string siteUrl,

string title,

string description,

uint nLCID,

uint compatibilityLevel,

string webTemplate,

string ownerLogin,

string ownerName,

string ownerEmail,

string secondaryContactLogin,

string secondaryContactName,

string secondaryContactEmail,

string databaseServer,

string databaseName,

string userName,

string password

)

The web template codes for the most commonly used templates are:

Template Name	Description
GLOBAL#0	Global template (1033)
STS#0	Team Site (1033)
STS#1	Blank Site (1033)
STS#2	Document Workspace (1033)
MPS#0	Basic Meeting Workspace (1033)
MPS#1	Blank Meeting Workspace (1033)
MPS#2	Decision Meeting Workspace (1033)

Template Name	Description
MPS#3	Social Meeting Workspace (1033)
MPS#4	Multipage Meeting Workspace (1033)
CENTRALADMIN#0	Central Admin Site (1033)
WIKI#0	Wiki Site (1033)
BLOG#0	Blog (1033)
BDR#0	Document Center (1033)
OFFILE#0	Records Center (1033)
OFFILE#1	Records Center (1033)
OSRV#0	Shared Services Administration Site (1033)
SPS#0	SharePoint Portal Server Site (1033)
SPSPERS#0	SharePoint Portal Server Personal Space (1033)
SPSMSITE#0	Personalization Site (1033)
SPSTOC#0	Contents area Template (1033)
SPSTOPIC#0	Topic area template (1033)
SPSNEWS#0	News Site (1033)
CMSPUBLISHING#0	Publishing Site (1033)
BLANKINTERNET#0	Publishing Site (1033)
BLANKINTERNET#1	Press Releases Site (1033)
BLANKINTERNET#2	Publishing Site with Workflow (1033)
SPSNHOME#0	News Site (1033)
SPSSITES#0	Site Directory (1033)
SPSCOMMU#0	Community area template (1033)
SPSREPORTCENTER#0	Report Center (1033)
SPSPORTAL#0	Collaboration Portal (1033)
SRCHCEN#0	Search Center with Tabs (1033)
PROFILES#0	Profiles (1033)

Template Name	Description
BLANKINTERNETCONTAINER#0	Publishing Portal (1033)
SPSMSITEHOST#0	My Site Host (1033)
SRCHCENTERLITE#0	Search Center (1033)
SRCHCENTERLITE#1	Search Center (1033)
SPSBWEB#0	SharePoint Portal Server Bucket Web Template (1033)

The following code will show how to create a new subsite of blog template:

```
using (SPSite site = new SPSite("http://win-spserv:46466/sites/
CodeGenerated/"))

{

  using (SPWeb newWeb = site.AllWebs.Add(

  "Blog Site", // Web Site Url

  "Blog Site Created Pragmatically", // Web Site Title

  "Blogging Site Created pragmatically", // Web Site Description

  1033, // LCID

  "BLOG#0", // Web Site Template Name

  true, // Use Unique Permissions

  false // Convert an existing folder

  ))

  {

    Console.WriteLine("New Web Site URL: {0}", newWeb.Url);

  }

}
```

6 LIST

The SharePoint list is one of the building blogs of SharePoint. It is the most widely used form of data storage on the SharePoint platform. SharePoint list can be described as a spreadsheet or a simple database. It is the most commonly used form of storing and managing information in SharePoint. In the list, data is stored in row, which is known as a list item. Lists are usually customized to store data in the format required. The data in the rows are defined by columns which are called as metadata, columns, fields or properties. For example, we could have a list having name Employee which consists of the columns: Name, Address, Contact Number. The data in the list would contain the values for these fields like:

Name: Jack Drew
Address: 12 Street, xyz Avenue, abc City
Contact Number: 0123456

Some of the SharePoint column types are:

Column Type	Description	Enum Value
Single Line of Text	It contains strings and can store upto 255 characters.	Text
Multiple Lines of Text	It can contain Plain Text, Rich Text, or Enhanced Rich Text	Note
Choice	A list of defined choice is provided	Choice
Number	It can contain numerical values	Number
Currency	Monetary values are stored using this type	Currency
Date and Time	Date & Time values are stored using this type	DateTime
Lookup	Values from another list are referred in this type	Lookup
Yes/No	Boolean values are used	Boolean
Person or Group	The user name or group name	User

Column Type	Description	Enum Value
Hyperlink or Picture	Hyperlink (URL) or picture	URL
Calculated	Value based on some calculation	Calculated
External Data	Data from some external source like a database, spreadsheets or CRM apps.	
Task Outcome	This field is usually used for storing the outcome of workflow.	
Managed Metadata	Values selected from a specific term set of managed terms.	

Retrieve all the lists at a site

The following code demonstrates how to fetch all the lists on a site.

```
using (SPSite site = new SPSite("http://winspserver:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    foreach (SPList list in web.Lists)
    {
      Console.WriteLine("List Name: " + list.Title);
      Console.WriteLine("List GUID: " + list.ID);
    }
    Console.ReadKey();
  }
}
```

Creating a new SharePoint List

The following code demonstrates how to create a new list.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPListCollection lists = web.Lists;
    string listTitle = "Test List";
    string listDescription = "Creating list pragmatically";
    SPListTemplateType listTemplateType = new SPListTemplateType();
    listTemplateType = SPListTemplateType.GenericList;
    lists.Add(listTitle, listDescription, listTemplateType);
  }
}
```

In the above example we have created a SharePoint List having the name Test List and by making use of the generic list for list template. The SPListTemplateType is an enumeration used to select the list template. Some of the widely used list templates are:

Template Name	Description
GenericList	Custom list. Value = 100.
DocumentLibrary	Document library. Value = 101.
Survey	Survey. Value = 102.
Links	Links. Value = 103.
Announcements	Announcements. Value = 104.
Contacts	Contacts. Value = 105.
Events	Calendar. Value = 106.
Tasks	Tasks. Value = 107.

The list can be fetched either by using GUID of the list as web.Lists[guid] or by using the URL as web.GetList(url), but fetching List using the GUID is most performance efficient and quicker when compared to GetList method.

Deleting a list

List is deleted by making use of the Delete method of the SPList object.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPList list = web.Lists["Test List"];
    list.Delete();
  }
}
```

Retrieving all the fields from the list

The following code fetches all the fields from a list and displays its internal name.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
  {
    using (SPWeb web = site.OpenWeb())
    {
      SPList list = web.Lists["Test List"];
      foreach (SPField field in list.Fields)
      {
        Console.WriteLine("Field Name: " + field.InternalName);
      }
    }
  }
```

Adding new field to an existing list

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPList list = web.Lists["Test List"];
    string fieldInternalName = list.Fields.Add("Description",
    SPFieldType.Text, false);
    list.Update();
  }
}
```

We have created a new field called Description, the column type used is single line of text and the required property has been set to false.

Modifying column from a list

In the following example we will be changing the field from single line of text to multi line of text. GetList method has been used to demonstrate how it can be used to fetch list.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPList list = web.GetList("/Lists/Test%20List/AllItems.aspx");
    SPField field = list.Fields["Description"];
    field.Type = SPFieldType.Note;
    field.Update();
  }
}
```

Deleting column from a list

In the following example we will be deleting the field from the list. To do so we need to specify the field name and execute the delete method from the SPField object.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPList list = web.Lists["Test List"];
    SPField field = list.Fields["desc"];
    field.Delete();
  }
}
```

Adding a new list item to the list

In the following example we will be adding a new list item to the list.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPListItemCollection listItems = web.Lists["Test List"].Items;
    SPListItem item = listItems.Add();
    item["Title"] = "Sample Data";
    item.Update();
  }
}
```

Retrieving all List Items in a list

In the following example we will be fetching all the list items in a list and display the Title of the list item.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPList list = web.Lists["Test List"];
    SPListItemCollection listItems = list.Items;
    foreach (SPListItem item in listItems)
    {
      Console.WriteLine(item["Title"]);
    }
    Console.ReadKey();
  }
}
```

Fetching List Item

In the following example we will be fetching a particular list item from the list. The GetItemById method is used, and it fetches the list item based on the id. Once we have the list item we access the required field by mentioning the field name in the array. Another way to fetch list items is by making use of CAML query which would be explored in detail further in the book.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
      SPList list = web.Lists["Test List"];
      SPListItem item = list.GetItemById(1);
      Console.WriteLine(item["Title"]);
      Console.ReadKey();
    }
  }
```

Updating List Item

In the following example we will be updating a particular value of the list item from the list, i.e. the Title value of the list item having ID one is modified.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPList list = web.Lists["Test List"];
    SPListItem item = list.GetItemById(1);
    item["Title"] = "Modified the title value";
    item.Update();
  }
}
```

Deleting List Item

The following example demonstrates how to delete a list item based on its ID.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPList list = web.Lists["Test List"];
    SPListItem item = list.GetItemById(2);
    item.Delete();
  }
}
```

Creating a new view for the list

The following example demonstrates how to create a new view for the list. List view is used to view the list in a different more customized format than the default list view. By making use of views you can choose to display or hide certain columns, filter or sort data, apply grouping. This enables a user to fetch and view data in a format that is more useful to him. The following code demonstrates how to create a new custom view. We are creating a view having the name CustomView, which would display the Title and Description columns. We have made use of CAML query for this code, which would be explained in detail further in the book. The CAML query is fetching data that has "Modified Data" as it's title.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPList list = web.Lists["Test List"];
    SPViewCollection viewCollection = list.Views;
    string strViewName = "CustomView";
    StringCollection viewFields = new StringCollection();
    viewFields.Add("Title");
    viewFields.Add("Description");
    string query = "<Where><Eq><FieldRef Name=\"Title\"/>" +"<Value
    Type=\"Text\">Modified Data</Value></Eq></Where>";// here you
    can filter your items using the selected item in the dropdownlist
    viewCollection.Add(strViewName, viewFields, query, 100, true,
    false);
    web.Update();
  }
}
```

The syntax used for creating the view is:

```
public SPView Add(
     string strViewName,
     StringCollection strCollViewFields,
     string strQuery,
     uint iRowLimit,
     bool bPaged,
     bool bMakeViewDefault
)
```

strViewName refers to the name of the view.

strCollViewFields is the collection of that contains the internal names of the fields in the view.

Strquery is the CAML query that contains the where clause.

iRowLimit stands for the maximum number of items that the view will return.

bPages is to specify if the view supports displaying more items page by page if set to true, otherwise false.

bMakeViewDefault is set to true if this view needs to be set as the default view, otherwise false.

Retrieving existing List view

The following example demonstrates how to retrieve an existing list view and display its GUID.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPList list = web.Lists["Test List"];
    SPView view = list.Views["CustomView"];
    Console.WriteLine("View GUID: " + view.ID);
    Console.ReadKey();
  }
}
```

Modifying List View

The following code demonstrates how to edit an existing view. In the example we will be adding the country field to the view and removing the title field from the view.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPList list = web.Lists["Test List"];
    SPView view = list.Views["CustomView"];
    view.ViewFields.Add("Country");
    view.ViewFields.Delete("Title");
    view.Update();
    Console.ReadKey();
  }
}
```

Deleting List View

The following code demonstrates how to delete an existing list view, we use the Delete method and pass the GUID of the list view to do so.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPList list = web.Lists["Test List"];
    SPView view = list.Views["CustomView"];
    SPViewCollection oViewCollection = list.Views;
    oViewCollection.Delete(view.ID);
    list.Update();
  }
}
```

7 DOCUMENT LIBRARY

A document library is a location on the site where you can create, store and share files. A document library contains the file and information related to that file, such as creation date, created by, etc. Document library provides the means to manage files with features like check out, check in, publish to name a few. At the time of this righting the maximum limit for the number of items in the list or document library for SharePoint online and SharePoint 2016 are 30,000,000 per library/list and the maximum file size limit is 10 GB, although the default value is 2 GB.

Retrieve All Document Libraries in a site

The following code retrieves all the document libraries at a particular site.

```
using (SPSite oSite = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb oWeb = oSite.OpenWeb())
  {
    SPListCollection docLibraryCol = oWeb.
    GetListsOfType(SPBaseType.DocumentLibrary);
    foreach (SPList docLib in docLibraryCol)
    {
      Console.WriteLine("Document Library Name:" + docLib.Title
      + "<br>");
    }
  }
}
```

Retrieve All Files from all Document Libraries in a site

```
using (SPSite oSite = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb oWeb = oSite.OpenWeb())
  {
    foreach (SPList list in oWeb.Lists)
    {
      SPDocumentLibrary doclibrary = list as SPDocumentLibrary;
      if (doclibrary != null)
      {
        foreach (SPListItem item in doclibrary.Items)
        {
          Console.WriteLine("File Name:" + item.Name);
          Console.WriteLine("File URL:" + item.Url);
        }
      }
    }
  }
}
```

Create new document library

```
using (SPSite oSite = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb oWeb = oSite.OpenWeb())
  {
    oWeb.Lists.Add("New Doc Library", "Created using SSOM",
    SPListTemplateType.DocumentLibrary);
    oWeb.Update();
  }
}
```

Fetch and update existing Document Library

The following code demonstrates for to fetch an existing document library and perform changes to it.

```
using (SPSite oSite = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb oWeb = oSite.OpenWeb())
  {
    SPList list = oWeb.Lists["New Doc Library"];
    list.Title = "Doc Library";
    list.Update();
  }
}
```

Display all items and folders at the top level of a particular document library

The following code fetches all the files and folders at the top level of a specific document library.

```
using (SPSite oSite = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb oWeb = oSite.OpenWeb())
  {
    SPList list = oWeb.Lists["Doc Library"];
    foreach (SPListItem item in list.Items)
    {
      Console.WriteLine("File Name:" + item.Name);
    }
      foreach (SPFolder folder in list.RootFolder.SubFolders)
    {
      Console.WriteLine("Folder Name:" + folder.Name);
    }
  }
}
```

Delete a document library

```
using (SPSite oSite = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb oWeb = oSite.OpenWeb())
  {
    SPList docLib = oWeb.Lists["Doc Library"];
    docLib.Delete();
    oWeb.Update();
  }
}
```

Creating a folder within a document library

```
using (SPSite oSite = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb oWeb = oSite.OpenWeb())
  {
    SPList docLib = oWeb.Lists["Doc Library"];
    SPFolderCollection folderCol = docLib.RootFolder.SubFolders;
    SPFolder subFolder = folderCol.Add("/Doc Library/subfolder");
    docLib.Update();
  }
}
```

Retrieve all items and subfolders at a particular folder

In the following code we specify the document library name and the name of the subfolder and fetch all the items and folders at that folder.

```
using (SPSite oSite = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb oWeb = oSite.OpenWeb())
  {
    SPFolder parentFolder = oWeb.Lists["Doc Library"].RootFolder.
    SubFolders["subfolder"];
    foreach (SPFile file in parentFolder.Files)
    {
      Console.WriteLine("File Name:" + file.Name);
    }
    foreach (SPFolder folder in parentFolder.SubFolders)
    {
      Console.WriteLine("Folder Name:" + folder.Name);
    }
    Console.ReadKey();
  }
}
```

Modify folder values

In this code, it will be demonstrated how to modify the properties of the folder. In this particular example we are renaming the name of the sub folder.

```
using (SPSite oSite = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb oWeb = oSite.OpenWeb())
  {
    SPFolder folder = oWeb.GetFolder("Doc Library/subfolder");
    folder.Item["Name"] = "newFolder";
    folder.Item.SystemUpdate();
  }
}
```

Delete folder from the document library

```
using (SPSite oSite = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb oWeb = oSite.OpenWeb())
  {
    SPFolder folder = oWeb.Lists["Doc Library"].RootFolder.
    SubFolders["newfolder"];
    folder.Delete();
    oWeb.Update();
  }
}
```

Create a new file in the document library

In the following code we will be saving a file into the document library. The file is in the physical location "C:\Test.txt".

```
String fileToUpload = @"C:\Test.txt";

String site = "http://win-kmp:46466/";

String documentLibraryName = "Doc Library";

using (SPSite oSite = new SPSite(site))

{

  using (SPWeb oWeb = oSite.OpenWeb())

  {

    SPFolder docLibrary = oWeb.Folders[documentLibraryName];

    Boolean replaceExistingFiles = true;

    String fileName = System.IO.Path.GetFileName(fileToUpload);

    FileStream fileStream = File.OpenRead(fileToUpload);

    SPFile spfile = docLibrary.Files.Add(fileName, fileStream,
    replaceExistingFiles);

    docLibrary.Update();

  }

}
```

Retrieve specific file from a document library

```
using (SPSite oSite = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb oWeb = oSite.OpenWeb())
  {
    SPFolder folder = oWeb.GetFolder("Doc Library");
    SPFile file = folder.Files["Test.txt"];
    Console.WriteLine("File Name:" + file.Name);
  }
}
```

Updating/Modifying the file in a document library

The following code will demonstrate how to update a file within a document library.

```
using (SPSite oSite = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb oWeb = oSite.OpenWeb())
  {
    SPList list = oWeb.Lists["Doc Library"];
    SPListItem item = list.Items[0];
    item.File.CheckOut();
    item["Name"] = "New Name";
    item.Update();
    item.File.CheckIn("file name changed successfully");
  }
}
```

Deleting file from a document library

```
using (SPSite oSite = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb oWeb = oSite.OpenWeb())
  {
    SPList list = oWeb.Lists["Doc Library"];
    SPListItem item = list.Items[0];
    item.Delete();
    oWeb.Update();
  }
}
```

8 CONTENT TYPES

Content types are one of the core building blocks of SharePoint. According to Microsoft's definition, "content type is a reusable collection of metadata (columns), workflow, behavior, and other settings for a category of items or documents in a Microsoft SharePoint Foundation list or document library. Content types enable you to manage the settings for a category of information in a centralized, reusable way". Content types can be described as a collection of metadata about an item or document.

Create a new Content Type

In the following example we will be creating a new content type named "HR Report" and it will be used for content types of Document. The content type has been added to the group Custom Content Types.

```
using (SPSite oSite = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPContentType contentType = new SPContentType(web.
    ContentTypes["Document"], web.ContentTypes, "HR Report");
    web.ContentTypes.Add(contentType);
    contentType.Group = "Custom Content Types";
    contentType.Description = "HR report content type";
    contentType.FieldLinks.Add(new SPFieldLink(web.Fields.
    GetField("Name")));
    contentType.FieldLinks.Add(new SPFieldLink(web.Fields.
    GetField("Due Date")));
    contentType.Update();
  }
}
```

Retrieve an existing Content Type

In the following example, we will be retrieving a content type named "HR Report" and display some of its details.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPContentType contentType = web.AvailableContentTypes["HR
    Report"];
    Console.WriteLine("Name:" + contentType.Name);
    Console.WriteLine("Description:" + contentType.Description);
  }
}
```

Modifying an existing Content Type

In the following example, we will be retrieving a content type named "HR Report" and display some of its details. Note that we have used the ContentTypes method instead of AvailableContentTypes because the content types retrieved from this collection are read only.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPContentType contentType = web.ContentTypes["HR Report"];
    contentType.FieldLinks.Add(new SPFieldLink(web.Fields.
    GetField("Office")));
    contentType.Update();
  }
}
```

Deleting an existing Content Type

In the following example, we will be deleting a content type named "Test". Before you delete any content type, it is important to ensure that you delete any association to the content type in any list, document library or anywhere else in the site. The following code checks if the content type is still in use, if yes then it displays the places that still use it else it deletes the content type.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPContentType contentType = web.ContentTypes["Test"];
    IList<SPContentTypeUsage> usages = SPContentTypeUsage.
    GetUsages(contentType);
    if (usages.Count > 0)
    {
      Console.WriteLine("The content type is used in the following:");
      foreach (SPContentTypeUsage usage in usages)
      {
        Console.WriteLine(usage.Url);
      }
    }
    else
    {
      web.ContentTypes.Delete(contentType.Id);
    }
  }
}
```

Display all content types

In the following example, we will be display all the content types available at a site.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    SPContentTypeCollection allContentTypes = web.ContentTypes;
    foreach (SPContentType ct in allContentTypes)
    {
      Console.WriteLine("Name: " + ct.Name);
    }
    Console.ReadKey();
  }
}
```

9 SITE COLUMNS

A site column is a reusable column definition that can be assigned to a list or document library and used across a site. In simple words, a site column is a column in a list or a document library. In an organization there could be a column that is common across lists, rather than creating a new list column, it is much more effective to create a site column and use it across lists. This ensures consistency across the site for the data in that site column. Site columns are reusable as you have to use the same site column than to create a new list column.

Display all Site columns

In the following example, we will be displaying all the site columns available at a site.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    foreach (SPField field in web.Fields)
    {
      Console.WriteLine("Site Column: " + field.Title);
    }
    Console.ReadKey();
  }
}
```

Create a new site column

In the following example, we will be creating a new site column. The code creates a site column of name "DateOfBirth" which is of type DateTime and the required field property has been set to false.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    string siteColumnName = "DateOfBirth";
    if (!web.Fields.ContainsField(siteColumnName))
    {
      web.Fields.Add(siteColumnName, SPFieldType.DateTime, false);
      web.Update();
    }
  }
}
```

Fetch a site column

In the following example, it is demonstrated how to fetch a particular site column based on its name.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
{
  using (SPWeb web = site.OpenWeb())
  {
    string siteColumnName = "DateOfBirth";
    if (web.Fields.ContainsField(siteColumnName))
    {
      SPField field = web.Fields.GetField(siteColumnName);
      Console.WriteLine("Site Column:" + field.Title);
    }
    else
    {
      Console.WriteLine("Site Column doesn't exist");
    }
  }
}
```

Modify an existing Site Column

In the following code, the property of an existing site column will be modified. Changing the type of the site column can result in loss of data, exercise caution when performing such operations. In this example we will be converting the changing the field of the site column from DateTime to single line of text.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
  {
    using (SPWeb web = site.OpenWeb())
    {
      string siteColumnName = "DateOfBirth";
      if (web.Fields.ContainsField(siteColumnName))
      {
        SPField field = web.Fields.GetField(siteColumnName);
        field.Type = SPFieldType.Text;
        field.Update();
      }
      else
      {
        Console.WriteLine("Site Column doesn't exist");
      }
    }
  }
```

Delete Site Column

In the following example, we will be deleting the site column. Deleting a site column will not delete the site column from the lists or document libraries which currently use the site column.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
  {
    using (SPWeb web = site.OpenWeb())
    {
      string siteColumnName = "Xyz";
      if (web.Fields.ContainsField(siteColumnName))
      {
        SPField field = web.Fields.GetField(siteColumnName);
        field.Delete();
        web.Update();
      }
      else
      {
        Console.WriteLine("Site Column doesn't exist");
      }
    }
  }
```

10 SECURITY & PERMISSIONS

A SharePoint group is a collection of users who have certain permissions. These permissions determine the type of control that the user/group has on the site or list or library. Groups makes it easier to manage users who have the same permission level. Permissions are inherited from the parent object by default but the inheritance can be broken.

Default Permission Groups

The following table displays the default groups and their default permissions.

Group Name	Default Permission Level	Description
Visitors	Read	This group grants its users the Read permission.
Members	Edit	This group grants its users the Edit permission.
Owners	Full Control	This group grants its users the Full Control permission.
Viewers	View Only	This group grants its users the View Only permission.
Restricted Readers	Restricted Read to the site, plus Limited Access to specific lists	This group grants its users to view pages and documents, but cannot view historical versions or review user rights information.
Style Resource Readers	Read to the Master Page Gallery and Restricted Read to the Style Library.	This group grants its users the Read permission to the Master Page Gallery and Restricted Read permission to the Style Library. By default, all authenticated users are a member of this group.
Designers	Design, Limited Access	This group grants its users to view, add, update, delete, approve, and customize the layout of site pages by using the browser or SharePoint Designer 2013.
Approvers	Approve, plus Limited Access	This group grants its users to edit and approve pages, list items, and documents.
Hierarchy Managers	Manage Hierarchy, plus Limited Access	This group grants its users to create sites, lists, list items, and documents.

Default Permission Levels

The following table lists the default permission levels for sites in SharePoint 2013.

Permission level	Description	Permissions included by default
View Only	This permission enables users to view application pages. This permission is used for the Excel Services Viewers group.	• View Application Pages • View Items • View Versions • Create Alerts • Use Self Service Site Creation • View Pages • Browse User Information • Use Remote Interfaces • Use Client Integration Features • Open
Limited Access	This permission enables limited access to the user. Specific permissions to enable users to access a specific list, document library, folder, list item, or document, without enabling them to access the whole site. Limited Access cannot be edited or deleted.	• View Application Pages • Browse User Information • Use Remote Interfaces • Use Client Integration Features • Open
Read	This permission enables the users to view pages and list items, and to download documents.	Limited Access permissions, plus: • View Items • Open Items • View Versions • Create Alerts • Use Self-Service Site Creation • View Pages
Contribute	This permission enables the users to manage personal views, edit items and user information, delete versions in existing lists and document libraries, and add, remove, and update personal Web Parts.	Read permissions, plus: • Add Items • Edit Items • Delete Items • Delete Versions • Browse Directories • Edit Personal User Information • Manage Personal Views • Add/Remove Personal Web Parts • Update Personal Web Parts

Permission level	Description	Permissions included by default
Edit	This permission enables the users to manage lists.	Contribute permissions, plus: • Manage Lists
Design	This permission enables the users to view, add, update, delete, approve, and customize items or pages in the website.	Edit permissions, plus: • Add and Customize Pages • Apply Themes and Borders • Apply Style Sheets • Override List Behaviors • Approve Items
Full Control	This permission enables the users to have full control of the website.	All permissions
Restricted Read	This permission enables the users to view pages and documents. For publishing sites only.	• View Items • Open Items • View Pages • Open
Approve	This permission enables the users to edit and approve pages, list items, and documents. For publishing sites only.	Contribute permissions, plus: • Override List Behaviors • Approve Items
Manage Hierarchy	This permission enables the users to create sites; edit pages, list items, and documents, and change site permissions. For Publishing sites only.	Design permissions minus the Approve Items, Apply Themes and Borders, and Apply Style Sheets permissions, plus: • Manage permissions • View Web Analytics Data • Create Subsites • Manage Alerts • Enumerate Permissions • Manage Web Site

Permission Types

The following table lists the types of permissions for lists in SharePoint 2013.

Permission	Description	Dependant Permissions	Included in these permission levels by default
Manage Lists	This permission allows the user to create and delete lists, add or remove columns in a list, and add or remove public views of a list.	View Items, View Pages, Open	Edit, Design, Full Control, Manage Hierarchy
Override List Behaviors	This permission allows the user to discard or check in a document that is checked out to another user, and change or override settings that allow users to read/edit only their own items.	View Items, View Pages, Open	Design, Full Control
Add Items	This permission allows the user to add items to lists, and add documents to document libraries.	View Items, View Pages, Open	Contribute, Edit, Design, Full Control
Edit Items	This permission allows the user to edit items in lists, edit documents in document libraries, and customize Web Part pages in document libraries.	View Items, View Pages, Open	Contribute, Edit, Design, Full Control
Delete Items	This permission allows the user to delete items from a list, and documents from a document library.	View Items, View Pages, Open	Contribute, Edit, Design, Full Control
View Items	This permission allows the user to view items in lists, and documents in document libraries.	View Pages, Open	Read, Contribute, Edit, Design, Full Control
Approve Items	This permission allows the user to approve a minor version of list items or document.	Edit Items, View Items, View Pages, Open	Design, Full Control
Open Items	This permission allows the user to view the source of documents with server-side file handlers.	View Items, View Pages, Open	Read, Contribute, Edit, Design, Full Control

Permission	Description	Dependant Permissions	Included in these permission levels by default
View Versions	This permission allows the user to view past versions of a list item or document.	View Items, Open Items, View Pages, Open	Read, Contribute, Edit, Design, Full Control
Delete Versions	This permission allows the user to delete past versions of list items or documents.	View Items, View Versions, View Pages, Open	Contribute, Edit, Design, Full Control
Create Alerts	This permission allows the user to create alerts.	View Items, View Pages, Open	Read, Contribute, Edit, Design, Full Control
View Application Pages	This permission allows the user to view forms, views, and application pages. Enumerate lists.	Open	All

The following table lists the types of permissions for sites in SharePoint 2013.

Permission	Description	Dependent Permissions	Included in these permission levels by default
Manage Permissions	This permission allows the user to create and change permission levels on the web site and assign permissions to users and groups.	View Items, Open Items, View Versions, Browse Directories, View Pages, Enumerate Permissions, Browse User Information, Open	Full Control
View Web Analytics Data	This permission allows the user to view reports on Web site usage.	View Pages, Open	Full Control
Create Subsites	This permission allows the user to create subsites such as team sites, Meeting Workspace sites, and Document Workspace sites.	View Pages, Browse User Information, Open	Full Control

Permission	Description	Dependent Permissions	Included in these permission levels by default
Manage Web Site	This permission grants the user the ability to perform all administration tasks for the web site, as well as manage content.	View Items, Add and Customize Pages, Browse Directories, View Pages, Enumerate Permissions, Browse User Information, Open	Full Control
Add and Customize Pages	This permission allows the user to add, change, or delete HTML pages or Web Part pages, and edit the website.	View Items, Browse Directories, View Pages, Open	Design, Full Control
Apply Themes and Borders	This permission allows the user to apply a theme or borders to the whole website.	View Pages, Open	Design, Full Control
Apply Style Sheets	This permission allows the user to apply a style sheet (.css file) to the website.	View Pages, Open	Design, Full Control
Create Groups	This permission allows the user to create a group of users that can be used anywhere within the site collection.	View Pages, Browse User Information, Open	Full Control
Browse Directories	This permission allows the user to enumerate files and folders in a website by using SharePoint Designer 2013 and Web DAV interfaces.	View Pages, Open	Contribute, Edit, Design, Full Control
Use Self-Service Site Creation	This permission allows the user to create a website using Self-Service Site Creation.	View Pages, Browse User Information, Open	Read, Contribute, Edit, Design, Full Control
View Pages	This permission allows the user to view pages in a website.	Open	Read, Contribute, Edit, Design, Full Control
Enumerate Permissions	This permission allows the user to enumerate permissions on the website, list, folder, document, or list item.	Browse Directories, View Pages, Browse User Information, Open	Full Control

Permission	Description	Dependent Permissions	Included in these permission levels by default
Browse User Information	This permission allows the user to view information about users of the website.	Open	All
Manage Alerts	This permission allows the user to manage alerts for all users of the website.	View Items, View Pages, Open, Create Alerts	Full Control
Use Remote Interfaces	This permission allows the user to use SOAP, Web DAV, the Client Object Model, or SharePoint Designer 2013 interfaces to access the website.	Open	All
Use Client Integration Features	This permission allows the user to use features that launch client applications. Without this permission, users must work on documents locally and then upload their changes.	Use Remote Interfaces, Open, View Items	All
Open	This permission allows the user to enables users to open a website, list, or folder to access items inside that container.	None	All
Edit Personal User Information	This permission allows the user to enables users to change their own user information, such as adding a picture.	Browse User Information, Open	Contribute, Edit, Design, Full Control

The following table lists the types of personal permissions in SharePoint 2013.

Permission	Description	Dependent Permissions	Included in these permission levels by default
Manage Personal Views	This permission allows the user to create, change, and delete personal views of lists.	View Items, View Pages, Open	Contribute, Edit, Design, Full Control
Add/ Remove Personal Web Parts	This permission allows the user to add or remove personal Web Parts on a Web Part page.	View Items, View Pages, Open, Update Personal Web Parts	Contribute, Edit, Design, Full Control
Update Personal Web Parts	This permission allows the user to update Web Parts to display personalized information.	View Items, View Pages, Open	Contribute, Edit, Design, Full Control

Display all groups and their permission level at a site

The following code displays all the groups that have permissions at the site.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
  {
    using (SPWeb web = site.OpenWeb())
    {
      foreach (SPRoleAssignment role in web.RoleAssignments)
      {
        if (role.Member is SPGroup)
        {
          Console.WriteLine("Group Name: " + role.Member.Name);
          foreach (SPRoleDefinition roleDefinition in role.
          RoleDefinitionBindings)
          {
            Console.WriteLine("\t" + roleDefinition.Name);
          }
        }
      }
      Console.ReadKey();
    }
  }
```

Fetch a specific group

The following code displays how to fetch a specific group and display its ID.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
  {
    using (SPWeb web = site.OpenWeb())
    {
      SPGroup group = web.Groups.GetByName("Approvers");
      Console.WriteLine(group.ID);
      Console.ReadKey();
    }
  }
```

Display all users of a group

The following code displays all the users belonging to a particular group.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
  {
    using (SPWeb web = site.OpenWeb())
    {
      SPGroup group = web.Groups.GetByName("Approvers");
      foreach (SPUser user in group.Users)
      {
        Console.WriteLine("User Name: " + user.Name);
      }
      Console.ReadKey();
    }
  }
```

Create new group

The following code demonstrates how to create a new group, and assigns the permissions of Administrator for the group.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
  {
    using (SPWeb web = site.OpenWeb())
    {
      SPGroup group;
      SPUser user = web.CurrentUser;
      web.SiteGroups.Add("Admins", user, web.Author, "Group created
      by code");
      group = web.SiteGroups["Admins"];
      // Add the group's permissions
      SPRoleDefinition roleDefinition = web.RoleDefinitions.
      GetByType(SPRoleType.Administrator);
      SPRoleAssignment roleAssignment = new SPRoleAssignment(group);
      roleAssignment.RoleDefinitionBindings.Add(roleDefinition);
      web.RoleAssignments.Add(roleAssignment);
      web.Update();
    }
  }
```

Add user to group

The following code displays how to add a new user to the group.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
  {
    using (SPWeb web = site.OpenWeb())
    {
      SPGroup group;
      SPUser user = web.CurrentUser;
      group = web.SiteGroups["Admins"];
      group.AddUser(user);
      group.Update();
    }
  }
```

Remove user from group

The following code displays how to remove a user from the group. If user doesn't exist in the group, the code will throw an exception.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
  {
    using (SPWeb web = site.OpenWeb())
    {
      SPGroup group;
      SPUser user = web.CurrentUser;
      group = web.SiteGroups["Admins"];
      group.RemoveUser(user);
      group.Update();
    }
  }
```

Deleting a group from the site

The following code demonstrates how to delete a group.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
  {
    using (SPWeb web = site.OpenWeb())
    {
      SPGroupCollection collGroups = web.SiteGroups;
      collGroups.Remove("Admins");
    }
  }
```

Display all users at a site

The following code displays all the users that have permissions at the site.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
  {
    using (SPWeb web = site.RootWeb)
    {
      foreach (SPUser user in web.AllUsers)
      {
        Console.WriteLine("Username: " + user.Name);
      }
    }
  }
```

Display all users in a group

The following code displays all the users that have permissions at the site.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
  {
    using (SPWeb web = site.RootWeb)
    {
      SPGroup group = web.Groups.GetByName("Designers");
      foreach (SPUser user in group.Users)
      {
        Console.WriteLine("Username: " + user.Name);
      }
    }
  }
```

Retrieve user based on login name

The following code demonstrates how to retrieve a user based on his login name.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
  {
    using (SPWeb web = site.RootWeb)
    {
      SPUser user = web.EnsureUser(@"mujtronics\ko");
      Console.WriteLine("Name: " + user.Name);
    }
  }
```

Display all the groups that the user belongs to at a site

The following code demonstrates all the groups that the user belongs to at the site.

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
  {
    using (SPWeb web = site.RootWeb)
    {
      SPUser user = web.CurrentUser;
      SPGroupCollection groupColl = user.Groups;
      foreach (SPGroup group in groupColl)
      {
        Console.WriteLine("Group Name: " + group.Name);
      }
    }
  }
```

Running with elevated privileges

There are times when you could have a user without sufficient permissions to make changes to a list or document library. You might not want to grant these permissions permanently to a user. An Example of this is when you have created a form in a public facing website on SharePoint and you want the anonymous users to fill a form and save these values into the SharePoint list. Giving permissions to anonymous users to edit would be a security hazard, and the best approach to this is to run the code with Elevated privileges. SP.Security. RunWithElevatedPrivileges method executes a specific method as full control even if the user doesn't have full control permissions. The application pool account is used to run that section of the code.

The following code demonstrates the template in which the method can be used.

```
SPSecurity.RunWithElevatedPrivileges(delegate ()
  {
    using (SPSite site = new SPSite("http://win-kmp:46466/"))
    {
      using (SPWeb web = site.RootWeb)
      {
        //Write the code here
      }
    }
  });
```

11 SPCONTEXT

The SPContext class represents the HTTP request. SPContext can be used to get the instance of the site or web from the current context. SPContext can also be used to get information of the current site rather than creating new objects of SPSite & SPWeb. This class is widely used while coding & its a good practice to use it.

Few examples of using the SPContext class are shown below:

- SPList currentList = SPContext.Current.List;
- SPWeb currentSite = SPContext.Current.Web;
- SPSite currentSiteCollection = SPContext.Current.Site;
- SPWebApplication currentWebApplication = SPContext.Current.Site.WebApplication;
- SPListItem item = (SPListItem)SPContext.Current.Item;
- SPUser currentUser = SPContext.Current.Web.CurrentUser;
- String url = SPContext.Current.Web.Url;

12 CAML

CAML or Collaborative Application Markup Language is the XML based language used in SharePoint to define its fields and views used in sites and lists. CAML can be used for various reasons, from creating content types to querying contents in a list. There are many tools for CAML Query generation and building, two of the popular tools are Saketa – CAML Query Builder & U2U CAML Query Builder. CAML query is mostly used to filter data in lists and document library. We will be exploring the commonly used elements while querying using CAML below.

1. Where
   The where clause is used within the context of a query to specify the filter.

2. And
   And is a logical join.

3. Or
   Or is a logical join.

4. BeginsWith
   It is used to fetch the string or note begins with a particular value.

5. Contains
   It is used to search for a particular string within a column.

6. DateRangesOverlap
   Used in queries to check whether recurring events overlap.

7. Eq
   This operator makes comparison if values are equal.

8. Geq
   This operator makes comparison if values are greater than or equal.

9. Gt
   This operator makes comparison if values are greater than.

10. In
    This operator is used to check multiple values on the list items.

Example:

```
<where>
  <in>
    <fieldref name="ID">
      <values>
        <value type="Number">1</value>
        <value type=" Number ">2</value>
      </values>
    </fieldref>
  </in>
</where>
```

11. Includes
    It is used to check in case of a lookup field that allows multiple values, if the specified field contains the specified value for that field.

12. IsNotNull
    Used to fetch items that are not empty.

13. IsNull
    Used to fetch items that are empty.

14. Leq
    This operator makes comparison if values are lesser than or equal.

15. Lt
    This operator makes comparison if values are lesser than.

16. Neq
    This operator makes comparison if values are not equal.

17. NotIncludes
    It is used to check in case of a lookup field that allows multiple values, if the specified field doesn't include the specified value for that field.

18. GroupBy
    This is used for grouping the data returned through the query.

19. OrderBy
    This is used for determining the sort order for the data returned through the query.

Example:

```
using (SPSite site = new SPSite("http://win-kmp:46466/"))
  {
    using (SPWeb web = site.OpenWeb())
    {
      SPList list = web.Lists["Test List"];
      SPQuery qry = new SPQuery();
      qry.Query = "<Where><Eq><FieldRef Name='Country' /><Value
      Type='Text'>Canada</Value></Eq></Where>";
      SPListItemCollection items = list.GetItems(qry);
      foreach (SPListItem item in items)
      {
        Console.WriteLine(item["CurrencyCode"]);
      }
      Console.ReadKey();
    }
  }
```